From A and B to A+B: Can Large Language Models Solve Compositional Math Problems?

Xisheng Xiao[†]**, Hanlin Zhao**[†] [*]
College of Mathematics and Informatics, South China Agricultural University, China
xishengxiao.mail@gmail.com, hanlin9908@gmail.com

Abstract

Large language models (LLMs) have demonstrated strong performance in solving math problems, and there is growing research on evaluating their robustness. Unlike previous studies that create problem variants by adding perturbations to a single problem, this paper focuses on the interaction between problems. Specifically, we combine two original problems with a logical connection to get a new math problem, and measure the LLM's performance on it to evaluate its compositional generalization, which is an important and essential reasoning capability in human intelligence. The result of experiments that cover 14 different LLMs shows that even when the mathematical essence remains unchanged, a simple form of combination can significantly reduce the performance of LLMs, revealing the limitation of their generalization ability. Additionally, we propose an automated pipeline with 98.2% accuracy to assist in annotating datasets (1 manual, 2 synthetic). The extensive experiments conducted on these datasets further verify the conclusion and obtain some important findings. Finally, we analyze the impact of factors such as difficulty and length on LLMs' performance, offering insights for future research. Code and data are avaliable at https://github.com/goooood-coder/NCSP.

1 Introduction

Math reasoning is the key to the development of artificial intelligence (Tenenbaum, 2018) and thus serves as an important aspect for evaluating large language models (LLMs) (Guo et al., 2023; Chang et al., 2024; Ahn et al., 2024). Current mainstream evaluation methods focus on constructing benchmarks of various difficulty levels, ranging from elementary school-level benchmarks (Cobbe et al., 2021) to olympic-level (Huang et al., 2024; Zheng et al., 2022), to assess a model's problem-solving capabilities. Although model scores on these benchmarks have been steadily increasing, the question remains contentious: have LLMs truly learned to solve math problems, or have they merely taken shortcuts through pattern matching (Patel et al., 2021; Mirzadeh et al., 2024; Shi et al., 2023) or even data leakage (Golchin and Surdeanu, 2024)? For instance, studies by Zhou et al. (2024b) and Mirzadeh et al. (2024) demonstrate that altering the entity names or adding irrelevant conditions can confuse models and disrupt their performance. This widespread situation has raised concerns about the generalization and robustness of LLMs.

	Problem	Solution from Llama3.1-70B-Inst
Problem$_1$	What is the only integer value of n for which $\frac{n+1}{13-n}$ is a positive prime number?	**Step 1: Understand the problem** ... The final answer is: $\boxed{12}$ ✔
Problem$_2$	Let M and N denote the greatest and least positive two-digit multiples of <u>*13.*</u> What is the value of $M + N$?	**Step 1: Identify the range of two-digit multiples of 13** ... The final answer is: $\boxed{104}$ ✔
Merged Problem	What is the only integer value of n for which $\frac{n+1}{13-n}$ is a positive prime number? <u>*Define p is the only integer value of n.*</u> <u>*q is 1 more than p.*</u> Let M and N denote the greatest and least positive two-digit multiples of <u>*q.*</u> What is the value of $M + N$?	**Step 1: Understand the problem** ... **Step 6: Identify *n*** ... *n=12.* ✔ ... **Step 8: Define *p* and *q*.** ... so *q=p+1=12+1=13.* ✔ **Step 9: Determine the positive two-digit multiples of *q*.** The least positive two-digit multiple of *13* is *13 * 8=104.* ✘ The greatest positive two-digit multiple of *13* is *13 * 7 = 91.* ✔ **Step 10: Calculate *M + N*.** *M=104* and *N=91*, so *M + N = 104 + 91 = 195.* ✘ The final answer is: $\boxed{195}$ ✘

Table 1: An example of a compositional math problem, where Llama3.1-70b-it successfully solves the seed problems but fails on the compositional one.

In response to these issues, we propose a novel evaluation perspective for LLMs' robustness evaluation. Unlike previous works (Mirzadeh et al., 2024; Li et al., 2024), which generate various kinds of problems variants from **single** problems, our approach considers the interaction between **multiple** problems, focusing more on the compositional generalization (Xu and Wang, 2024; An et al., 2023;

[†] Equal contribution.
[*] Corresponding author.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 13069–13090
November 4-9, 2025

Li et al., 2021) in robustness. Specifically, we combine two existing math problems into a new one by applying a simple logical connection. As shown in the example of Figure 1, we replace the number 13 in problem$_2$ with the variable q, and then substitute q with an equivalent condition, which is the problem$_1$. For humans, if they have mastered both Problem$_1$ and Problem$_2$ individually, this combined problem can be easily solved as well. Such compositional generalization, which refers to understanding unseen combinations of seen primitives, is an essential reasoning capability in human intelligence (An et al., 2023). Therefore, by evaluating the model's performance on both the original and the combined problem, we can assess its compositional generalization ability.

We conducted experiments on 14 LLMs with parameters ranging from 6B to 671B. Compared with the original dataset, even powerful models like o3-mini and Deepseek-r1 exhibited a relative score degradation of 5.3% and 3.7%, respectively. This result confirms that even when the mathematical essence remains unchanged, a simple form of combination can significantly reduce the performance of LLMs, highlighting the limitations in the compositional generalization of existing LLMs.

To automatically combine all kinds of math problems, we propose a **N**umerical-based **C**omposition **S**ynthesis **P**ipeline (NCSP), which uses LLMs and Tool-Integrated Reasoning (TIR) (Gou et al., 2024; Tahmid and Sarker, 2024) to verify problems, achieving an accuracy of 98.2% in the generated problems. Using the pipeline, we ultimately synthesized three datasets and manually verified one of them. Specifically, our synthetic data not only allows us to check whether the final results are correct, but also enables us to verify the correctness of intermediate variables.

The extensive experiments conducted on these datasets further verify the reliability of our conclusion and explore the relationship between problem features and compositional generalization ability. Our findings are as follows: (1) Two unrelated questions posed to LLMs had only a very slight impact if they have on any connection; (2) Error accumulation is another important reason causing the score drop on compositional problems; (3) Even if the probability that LLMs can answer the subproblems is almost 100%, they may still provide incorrect answers to the combined problem; (4) Problems with characteristics such as "high difficulty, low confidence, different types, long question length, and easy-to-difficult transitions" pose a greater challenge to compositional generalization.

Based on the findings in this paper, we aim to further investigate LLMs in math to improve their compositional generalization capabilities.

2 Related Work

Through data collection and cleaning of resources from textbooks, websites, and other materials (Yue et al., 2024), many benchmark datasets have been proposed to train and validate the math problem-solving abilities of LLMs. These datasets can be categorized based on difficulty into elementary-school level (Cobbe et al., 2021), high-school level (Hendrycks et al., 2021), college-level (Sawada et al., 2023), and olympic-level (Huang et al., 2024; Zheng et al., 2022) datasets. However, these benchmarks mainly focus on the final results of individual problems and do not intuitively reflect the math robustness of LLMs (Zhou et al., 2024a).

To address this issue, many works have introduced various modifications to the original benchmarks to assess how models perform when faced with subtle perturbations (Li et al., 2024), such as semantic perturbations (Wang et al., 2023; Zhou et al., 2024b), problem reversal (Yu et al., 2024; Berglund et al., 2024), and irrelevant distractions (Shi et al., 2023; Li et al., 2023). Most of these works provide an automated synthesis pipeline. However, math problems often involve precise numerical design and logical interconnections, making the accuracy of automated synthesis relatively low (as shown in Table 3 and Table 4), which often requires manual annotation and modification.

Some studies use compositional problems to assess the robustness of LLMs. Hosseini et al. (2024) focus on why LLM performance declines, emphasizing "model types" like different LLM sizes and math-specific models. In comparison, our work examines both "model types" and "problem types," considering factors like problem length, quantity, and order. Miner et al. (2024) explore automatic problem synthesis with "if-else" combinations, while we focus on "numerical calculation" combinations. Both methods generalize well, but Miner et al.'s approach risks giving hints by merging answers into problem statements. Additionally, we evaluated reasoning models to validate the generality of our findings and used the more diverse and challenging MATH dataset instead of GSM8k to demonstrate the scalability of our approach.

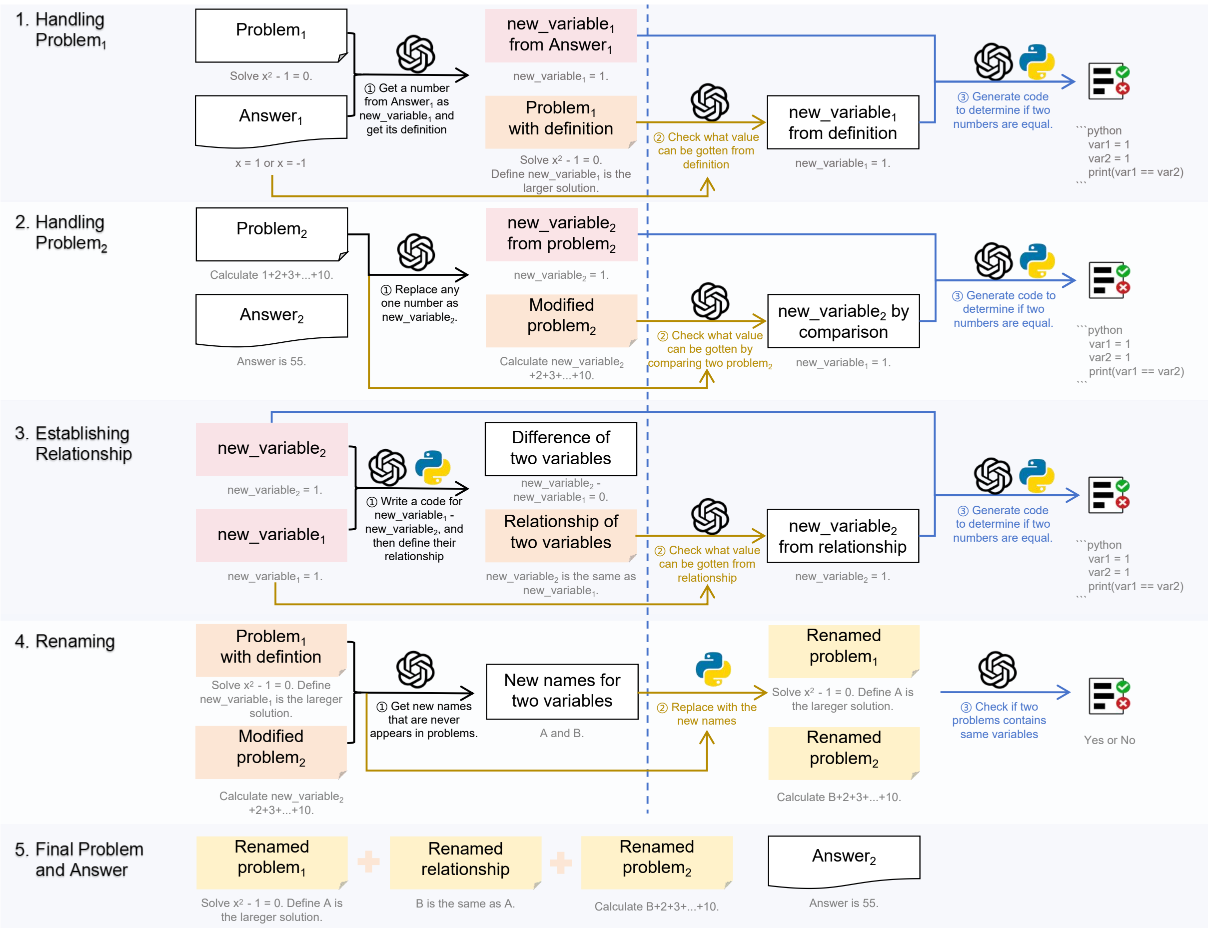


Figure 1: The overall pipeline structure of NCSP, including four steps with LLM and Python interpreter. The main processing steps are on the left side of the dashed line, while the right side verifies the result of LLM processing. Examples are provided in gray text beneath each entity.

3 Compositional Problem Generation

The compositional problem is not a direct concatenation, but a need to establish a logical connection between subproblems, such that in the new problem, if LLMs want to solve problem 2, they must first solve problem 1. However, math problems come in diverse types, making it hard to find one compositional method that works for all of them.

We observed that most math problems involve numbers, such as integers or floating-point values, which often play a critical role in mathematical reasoning. These numbers are governed by precise relationships derived from numerical computations. Based on this insight, we propose a novel framework **NCSP** (Numerical-based Composition Synthesis Pipeline). As illustrated in Figure 1, NCSP begins by masking an arbitrary number in Problem 2 and introducing an equivalent condition. This condition, together with the answer from Problem 1, is used to recover the masked number.

For the sake of clarity, we define the two problems being processed as p_1 and p_2, with their corresponding answers being a_1 and a_2. Our goal is to derive the compositional problem p_{12} and its answer a_{12} by LLM $\mathcal{M}$ and Python interpreter Py. The prompts (Sahoo et al., 2024; Zhao et al., 2023) of instruction $I = \{I_1, I_1', ..., I_4, I_4', I_c\}$ for each step are provided in Appendix D.

3.1 Handling Problem$_1$

Task. Given instruction I_1, problem p_1, and answer a_1, our goal is to extract a numeric value v_1 from a_1, which we denote as a new symbol ‘`new_variable1`’, and provide its definition d_1:

$$o_1 \sim \mathbb{P}_{\mathcal{M}}(\cdot | I_1 \oplus p_1 \oplus a_1), \tag{1}$$

where $o_1 = v_1 \oplus d_1$, and $\oplus$ refers to concatenation. As the example shown in Figure 1, v_1 is the integer ‘1’, which is the larger answer to the problem p_1, and d_1 is the corresponding definition ‘Define new_variable1 is the larger solution’.

In this step, we account for the possibility that

a_1 may not be a number, but instead a choice (e.g., A, B, C), or may consist of multiple numbers, such as coordinates or matrices. Detailed rules and multiple examples are provided to help the LLM better extract the number and give its definition.

Validation of d_1. The inherent nature of LLMs means their outputs can be unstable, so we employ additional steps to verify the correctness of the results. First, we provide the modified problem $\overline{p}_1 = p_1 \oplus d_1$ and the answer a_1 to let the LLM output the value of 'new_variable1', denoted as v_1', where the instruction is I_1'. Next, we give a prompt I_c and two instances of new_variable1 as the input of LLM, writing Python code to verify if they are equal.

$$v_1' \sim \mathbb{P}_{\mathcal{M}}(\cdot|I_1' \oplus p_1 \oplus a_1 \oplus d_1), \tag{2}$$

$$o_1' = Py(\mathbb{P}_{\mathcal{M}}(\cdot|I_c \oplus v_1 \oplus v_1')), \tag{3}$$

where $o_1' \in \{true, false, \delta\}$ is the execution result of code, and δ refer to other output text except $true$ or $false$. Although we have constrained the output of the code to include $true$ or $false$, there may be instances where errors occur or the output is not in the correct format. In the verification of this step and all subsequent steps, we will only keep the cases where the output is $true$.

3.2 Handling Problem$_2$

Task. Given p_2, our goal is to identify an arbitrary numeric value v_2 from p_2 and replace it with the variable symbol 'new_variable2'. The output of the task is $o_2 = v_2 \oplus \overline{p}_2$, where $\overline{p}_2$ is the masked problem. To improve accuracy, we suggest the model prioritize integers and fractions.

$$o_2 \sim \mathbb{P}_{\mathcal{M}}(\cdot|I_2 \oplus p_2). \tag{4}$$

Validation of $\overline{p}_2$. Given p_2 and the $\overline{p}_2$, we ask the LLM to output the value of new_variable2, denoted as v_2' . Similar to 3.1, we write Python code to validate if two instances of new_variable2 are equal. If the results $o_2' \in \{true, false, \delta\}$ is true, it indicates that the $\overline{p}_2$ and v_2 is correct. Additionally, we have observed cases where information is lost in the modified p_2, so we filter out cases where the length of p_2 is greater than the $\overline{p}_2$ by comparing the number of characters.

$$v_2' \sim \mathbb{P}_{\mathcal{M}}(\cdot|I_2' \oplus p_2 \oplus \overline{p}_2), \tag{5}$$

$$o_2' = Py(\mathbb{P}_{\mathcal{M}}(\cdot|I_c \oplus v_2 \oplus v_2')). \tag{6}$$

3.3 Establishing Relationships

Task. Given v_1 and v_2, we allow the model to generate Python code to subtract the two values and compute the difference o_3. Based on the sign of o_3, we can determine their relationship $r \in \{gt, eq, lt\}$, where gt indicates $v_1 > v_2$, eq indicates $v_1 = v_2$, and lt indicates $v_1 < v_2$. As shown in the example in Figure 1, the execution result of the subtraction code is 0, which indicates that new_variable2 is equal to new_variable1.

$$o_3 = Py(\mathbb{P}_{\mathcal{M}}(\cdot|I_3 \oplus v_1 \oplus v_2)). \tag{7}$$

Validation of Relationship Definition. Given v_1 and the relationship r, we ask the LLM to output the value of new_variable2, denoted as v_2''. Then have the LLM write Python code to check whether the two values of new_variable2 are equal. If the judgment $o_3' \in \{true, false, \delta\}$ is true, it indicates that the relationship definition is correct.

$$v_2'' \sim \mathbb{P}_{\mathcal{M}}(\cdot|I_3' \oplus v_1 \oplus r), \tag{8}$$

$$o_3' = Py(\mathbb{P}_{\mathcal{M}}(\cdot|I_c \oplus v_2 \oplus v_2'')). \tag{9}$$

3.4 Renaming

Task. Given the first problem with definition $\overline{p}_1$, and the modified problem $\overline{p}_2$, we aim to rename the variable strings new_variable1 and new_variable2 to more conventional, unused variable names. Let these new variable symbols be denoted by s_1 and s_2, respectively. The task can be formulated as:

$$o_4 \sim \mathbb{P}_{\mathcal{M}}(\cdot|I_4 \oplus \overline{p}_1 \oplus \overline{p}_2), \tag{10}$$

where $o_4 = s_1 \oplus s_2$ is the output of LLM. After that we replace new_variable1 with s_1 and replace new_variable2 with s_2 in $\overline{p}_1$, $\overline{p}_2$, and r, obtaining their renamed version, denoted as $\hat{p_1}$, $\hat{p_2}$, and $\hat{r}$. This renaming step helps ensure the variables do not appear artificial and adhere to standard naming conventions.

Validation of Renaming. To ensure that the renaming process is valid, we define the validation as follows: Given the renamed problems $\hat{p_1}$ and $\hat{p_2}$, we verify whether any variable names used in $\hat{p_1}$ appear in $\hat{p_2}$, or whether any variable in $\hat{p_2}$ appears in $\hat{p_1}$. In other words, we check if the two problems share any variables and confirm that no conflict occurs. It can be formulated as:

$$o_4' \sim \mathbb{P}_{\mathcal{M}}(\cdot|I_4' \oplus \hat{p_1} \oplus \hat{p_2}), \tag{11}$$

	Verify	Sub-P	Size	P-Len.	S-Len.
SEED	Human	1	200	59	445
VCMD	Pipeline, Human	2	500	152	743
ECMD	Pipeline	2	30,818	142	788
TCMD	Pipeline	3	6,878	199	912

Table 2: The statistics of experimental datasets, where P and S represent problem and solution, respectively. The solution used to calculate the average length is sampled from Llama-3.1-70b-it.

where $o_4' \in \{true, false, \delta\}$ represents the validation output confirming that the renaming has been correctly applied.

3.5 Final Problem and Answer

After the above steps, we have obtained all the elements required for the final problem p_{12}: the renamed problems $\hat{p_1}$ and $\hat{p_2}$, and their relationship $\hat{r}$. By combining these three, we get the final compositional problem and its answer:

$$p_{12} = \hat{p_1} \oplus \hat{r} \oplus \hat{p_2}, \tag{12}$$

$$a_{12} = a_2. \tag{13}$$

For the comparison and computation of numeric values within the pipeline, we adopted the TIR (Gou et al., 2024; Tahmid and Sarker, 2024) by using Python interpreter to obtain more reliable results. Python has well-established symbolic math libraries such as SymPy[1] , which can handle comparisons and calculations involving fractions, decimals, and other cases, e.g., $\frac{1}{2}$ and 0.5. More details and specific prompts used at each step are given in the Appendix D. It is worth mentioning that NCSP has many other implementation methods, such as masking two or more variables to construct more complex conditions. We chose this approach because it introduces the least amount of change and better reflects the generalization ability of LLMs.

4 Experiment

4.1 Datasets

Table 2 shows the dataset used in our experiment. Firstly, we synthesized a dataset called **VCMD** (Verified Compositional Math Dataset), consisting of 500 problems, utilizing the Llama-3.1-70b-it (Meta-AI, 2024) model. The dataset was created by evenly selecting 200 questions from the **MATH** (Hendrycks et al., 2021) dataset's test set as seed data, ensuring an equal representation across five difficulty levels and seven question types. Each seed question was then randomly combined with other questions five times, ensuring that every question was sampled an equal number of times. Each synthetic question was manually verified twice, and the error problems were repaired.

[1]https://www.sympy.org/

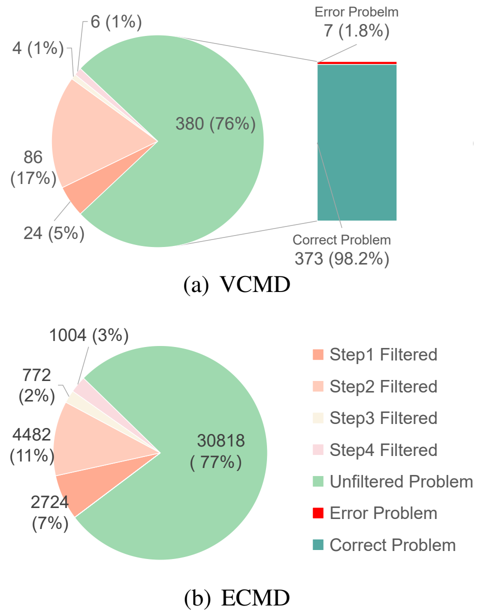


Figure 2: Component diagram of verification results. The pie chart (a) shows the composition of VCMD detected by pipeline and human, while the chart (b) shows the composition of ECMD detected only by pipeline.

As shown in Figure 2(a), our automated synthesis pipeline filtered out 120 failed combinations, leaving 380 valid questions, of which 373 were verified by humans as correct, achieving an accuracy rate of **98.2%**. Most of the errors observed were primarily due to two questions sharing the same variable (e.g., both containing the variable "x").

To further explore the factor contribution of compositional problems, we combined the 200 seed questions in pairs, resulting in a total of 39,800 combination tasks (200×199). After automatic verification, the final dataset size was 30,818 valid problems, which we named the **ECMD** (Extensive Compositional Math Dataset).

Finally, we selected the seed problems where Llama-3.1-8b-it and Llama-3.1-70b-it all answered correctly in 8 samples as seed data. Every three questions were randomly combined into a new one, resulting in **TCMD** (Triplet Compositional Math Dataset), with size of 6,878.

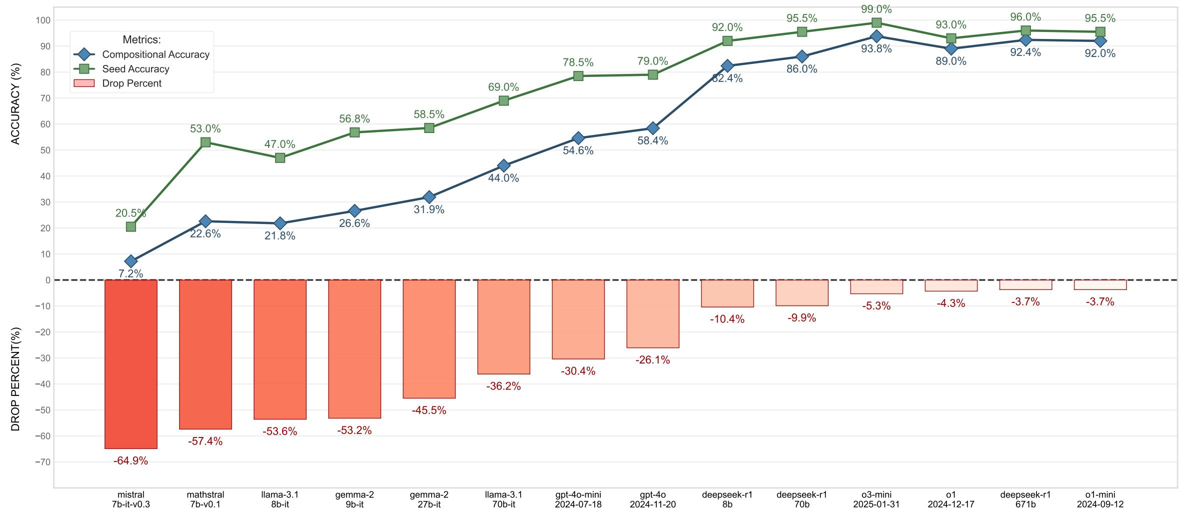


Figure 3: Comparison of various models on the combination problem dataset and on the original dataset.

4.2 Models

We conducted experiments on a range of state-of-the-art models, with the following rationale:

Different Sizes Models: The *Llama-3.1* series (Meta-AI, 2024) and the *Gemma-2* series (Gemma-Team, 2024), including *Llama-3.1-8b-it*, *Llama-3.1-70b-it*, *Gemma-2-9b-it*, and *Gemma-2-27b-It*, are selected to explore the effect of model size on performance.

Math-Specialized Models: *Mathstral-7b-v0.1* (Mistral-AI., 2024) are models specifically fine-tuned for math problems, chosen for comparing with *Mistral-7b-it-v0.3* (Jiang et al., 2023).

Closed-Source Models: *gpt-4o-mini-2024-07-18* (OpenAI, 2024b) and *gpt-4o-2024-11-20* (OpenAI, 2024b) are powerful closed-source models.

Reasoning Models: The closed-source *o1* series (OpenAI, 2024c), including *o1-mini-2024-09-12*, *o1-2024-12-17*, and *o3-mini-2025-01-31*, as well as the open-source *DeepSeek-r1* series (DeepSeek-AI, 2025), including *DeepSeek-r1-671b*, *DeepSeek-r1-distill-llama-8b*, and *DeepSeek-r1-distill-llama-70b*, are reasoning models that have advanced reasoning capabilities by performing extended thinking before generating answers.

4.3 Overall Performance of Various LLMs on Compositional Math Problems

We evaluate the LLMs using pass@1 Accuracy (acc) and the Accuracy score drop percent (ρ), which can be formulated as:

$$acc = \frac{\Sigma_{(x,y)\in D}\Pi[\mathcal{M}(x), y]}{|D|} \cdot 100\%, \tag{14}$$

$$\rho = \frac{acc_2 - acc_1}{acc_1} \cdot 100\%, \tag{15}$$

where $\Pi[\cdot]$ is the operator to check equal, acc_2 is the Accuracy on the target dataset and acc_1 is the Accuracy on the corresponding original dataset.

We conducted extensive experiments on the VCMD, and the main experimental result is shown in Figure 3. There are some notable observations:

(1) Overall, while the frontier reasoning LLMs like o1-mini continue to excel in compositional datasets, other LLMs exhibited significant score reduction, and the magnitude of the decline is related to their original capability.

(2) In terms of absolute Accuracy, Mathstral-7b-v0.1 has the highest decline by 30% and o1-mini has the lowest decline, nearly 3.5%. SOTA model on the original dataset is o3-mini, which achieves 99%, but also shows a decrease of 5.2%.

(3) In terms of relative decrease, the largest and smallest decreases are 64.9% for Misral-7b-it-v0.3 and 3.7% for o1-mini, respectively. The open-source model DeepSeek-r1-671b also showcases a small relative drop of about 3.7%.

In fact, the score drop was expected, but the extent of the decline was somewhat surprising. For humans, after learning to solve individual problems A and B, it is natural to learn how to solve the A+B, which is the simple combination of these two problems. However, this simple "generalization ability" seems to be discounted in LLMs. This result underscores the significant value of using combinatorial math problems for evaluating LLMs. In fact, this performance degradation is attributed to generalization ability and error accumulation, as

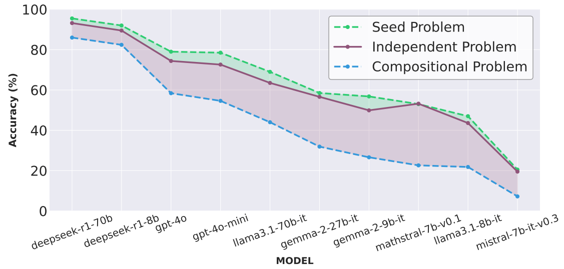


Figure 4: The performance of LLMs on independent problems pair without logical relationship.

we will demonstrate in the following experiments.

4.4 Do We Really Need Establishing Logical Relationship?

From the above experiments, we concluded that compositional problems challenge the generalization ability of LLMs. However, the necessity of our approach remains uncertain. Here is the issue that needs to be addressed: Do we really need to combine them? If we do not use our pipeline to combine the problems, but instead directly give the two problems to the LLMs to answer together, does the difficulty remain the same? As shown in Figure 4, the area of the green region, which represents the score drop of independent problem pairs compared to the original problems, is much smaller than that of the red region, which represents the score growth compared to VCMD. On the one hand, this result suggests that long context without logical connections does not interfere with LLMs as much as expected. On the other hand, it strongly supports the necessity of our method for establishing connections between problems.

4.5 Is the Generalization Ability Really Leading to the Performance Decline?

To further investigate what affects the performance decline of LLMs in combinatorial problems, we conducted in-depth research from multiple perspectives. Assume that the probability of correctly answering an original problem $p_i \in D_{seed}$ is given by P_i, while the probability of correctly answering a compositional problem $p_{ij} \in D_{composition}$ is denoted as $\overline{P_{ij}}$. Due to **error accumulation**, the likelihood probability of the model answering both questions correctly is $P_i \times P_j$, which is obviously less than or equal to P_i or P_j. It means that we should consider the influence of error accumulation.

Theoretically Best Performance Experiment. To evaluate the performance decline caused by the generalization ability after eliminating error accumulation, we propose a new metric, Theoretically Best Performance (*tbp*):

$$tbp = \frac{\sum_{(i,j)\in D_{composition}} P_i \times P_j}{|D_{composition}|}, \qquad (16)$$

Thus, the relative drop percentage rdp caused by generalization is defined as:

$$rdp = \frac{\Delta}{tbp} = \frac{acc_{compositon} - tbp}{tbp}, \qquad (17)$$

where Δ is the absolute degradation value and $acc_{compositon}$ is the average accuracy of the compositional problems.

According to this assumption, we sample 8 responses and calculate the avg@8 Accuracy for each problem to derive an accurate estimation of P_1, P_2, and $\overline{P_{ij}}$. The results shown in Figure 5 demonstrate that the relative degradation rdp caused by generalization across different models ranges from -2.6% to -23.4%, indicating that generalization ability significantly impacts model performance. Notably, compare with the Accuracy of seed problem shown in Figure 3, the score drop caused by error accumulation ($tbp - acc_{seed}$) has a greater impact than generalization. This means that our proposed compositional problems are capable of amplifying the 'difficulty level' of problems by accumulating their error probabilities. This characteristic enables the creation of more challenging math problems for both evaluation and training.

In fact, *tbp* is a theoretical value, not an actual calculation. It has a strong premise that "p_1 *and* p_2 *should be independent of each other*". Since LLMs are based on attention mechanisms, when solving the latter part of a problem, LLMs may leverage cues from their own responses to the earlier part. As a result, the *tbp* derived under the independence assumption may exhibit slight deviations from the actual theoretical best performance. This means that *the actual attenuation caused by generalization may be greater.* We also evaluated the *tbp* of Mistral-7B-it-v0.3, and found that its *tbp* was even lower than the $acc_{composition}$. We believe this is due to its relatively low overall performance, which is greatly influenced by context, thereby leading to an imprecise estimation.

Perfect Subproblem Experiment. The fundamental motivation we have always claimed is that, for

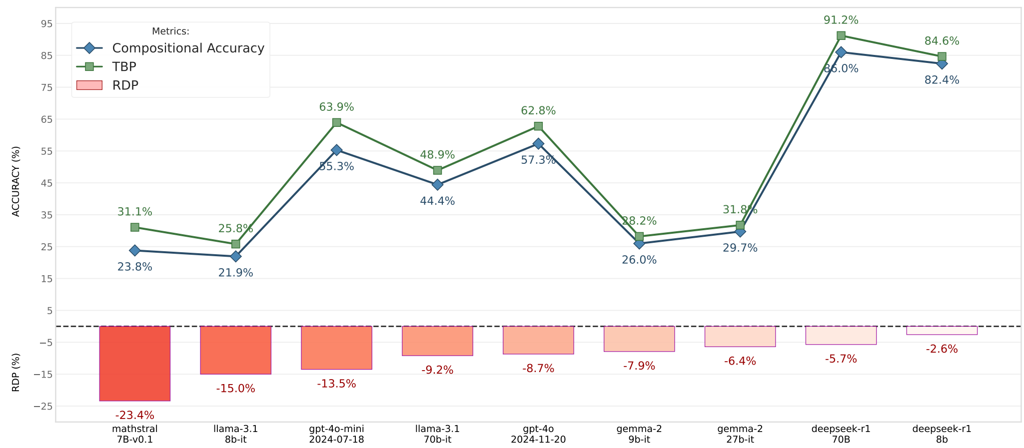


Figure 5: The theoretically best performance and the corresponding relative drop percentage of LLMs on VCMD.

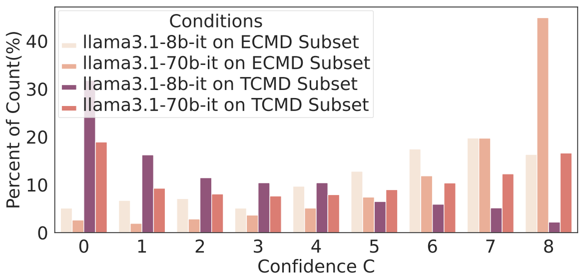


Figure 6: Confidence distribution chart, where the confidence of their sub-problems are all equal to 8.

humans, as long as they master the problems, they can also solve the combination of these problems. Here, we assume that 'master a problem' means an accuracy close to 100%:

$$p_1 \approx p_2 \approx 1, \tag{18}$$

In this scenario, the interference from error accumulation is minimal, and the measured performance can be considered an accurate reflection of the model's intrinsic generalization ability:

$$p_1 \times p_2 \approx 1. \tag{19}$$

$$p_1 \times p_2 \approx p_1 \approx p_2, \tag{20}$$

According to this setup, we first sampled 8 answers for the seed problem and defined the number of correct answers as the confidence level of the model in answering this problem, denoted as $C \in [0, 8]$. Then, we evaluated all the cases from ECMD and TCMD, which consist of seed problems with $C = 8$. As shown in Figure 6, even though LLMs have very high confidence in the subproblems, confidence tends to decay when faced with compositional problems. Most of the problems in ECMD are on the side where $C > 4$, while TCMD consists of three sub-problems, making it more difficult. As a result, the majority of the problem confidence in TCMD is distributed on the side where $C < 4$.

4.6 Dominant Factor Analysis

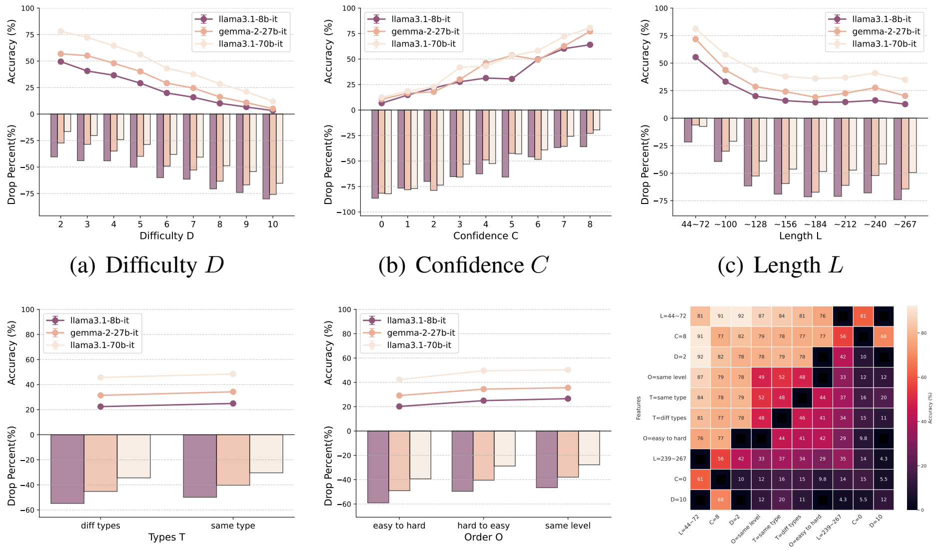
In fact, a math problem can have many characteristics, such as difficulty, type, and length, which may influence the combined problem's outcome. To identify the dominant factors affecting LLMs, we evaluate LLMs with ECMD by sampling 8 answers for each problem. We recorded the accuracy score of each subset and the proportion of score decay compared to the corresponding seed problems.

Difficulty. We define the difficulty of two seed problems as $D_1 \in [1, 5]$ and $D_2 \in [1, 5]$, where the combined problem is $D = D_1 + D_2$. As shown in Figure 7(a), as difficulty increases, the performance of the models gradually declines. Even the Llama-3.1-70b-it achieves only a score of 12.01% on the most difficult $D = 10$ problem, which is 66.13% lower than the highest score.

Subproblem Confidence. The smaller confidence level between the two sub problems, $C \in [0, 8]$, can be considered as a more personalized difficulty indicator. As shown in Figure 7(b), the drop in confidence values exhibits a greater degree of change and trend than difficulty does. The difference between the highest and lowest scores is 68%.

Length. The token count of a problem is regarded as its length L. As shown in Figure 7(c), in general, longer problems tend to have lower scores, though the correlation is not particularly strong.

Type. Different combinations of problem types, such as geometry and algebra, can enhance the



(a) Difficulty D (b) Confidence C (c) Length L

(d) Types T (e) Order O (f) Feature Cross Heatmap

Figure 7: The impact of features on LLMs' generalization ability. Figure (f) is a heatmap with intersecting features from Llama-3.1-70b-it, where cells without numbers represent problems with such features do not exist.

problem's diversity, which may also pose a challenge. We tested two types T: "same type" and "different type". The results, shown in Figure 7(d), are in line with expectations. However, LLMs do not seem to be very sensitive to this feature.

Order. We compared the easy-to-difficult problem with the difficult-to-easy problem to analysis the influence of order O. Results shown in Figure 7(e) suggest that LLMs tend to perform better on problems that progress from difficult to easy. This could be because, with fewer contextual clues at the beginning, LLMs' attention mechanisms more easily focus on the correct reasoning path, allowing them to tackle the complex part first and solve the simpler parts later with fewer errors.

Feature Interaction Analysis. Figure 7(f) shows a heatmap of the score performance based on various feature combinations for Llama3.1-70b-it, where the highest and lowest values were selected for each feature category. Obviously, confidence, length, and difficulty are all indicators that have a strong impact. The influence of confidence is the strongest among them. As long as $C = 8$, both subsets with $D = 10$ and subsets with $L = 239 \sim 267$ receive relatively high scores.

The results indicate that the combination of features such as "high difficulty, low confidence, different types, long question length, and easy-to-difficult" poses a greater challenge to the model, making it more suitable for verifying the model's combined generalization ability.

4.7 Case Study

We have observed an interesting phenomenon RAWR (Right Answer, Wrong Reasoning): intermediate variable results were incorrect, but the final answer was still correct. We identified two scenarios where RAWR occurs: (1) Match learned text patterns and forcefully apply them without any logic. As the first case is shown in Appendix C, gpt-4o got right answer because it matching the similar pattern of $(82 + 18)^3$. (2) The intermediate results provide a weak constraint on the final result. As shown in the second case, it will get the correct answer as long as $Q \in [\frac{90}{13}, \frac{92}{13}]$. The existence of RAWR suggests that it is better to introduce intermediate checkpoints when evaluate math problem, which is exactly what our combinational problems can naturally achieve.

5 Conclusion

This paper introduces a novel pipeline NCSP with 98.2% accuracy to combine math problems into a new one and to evaluate LLMs' generalization ability on the compositional problems. We annotate three compositional math problems datasets and evaluate 14 LLMs, the results of extensive experiments reveal the limitations of LLMs' generalization ability. Additionally, we analyze the key factors affecting compositional generalization and offer guidance for problem synthesis.

6 Limitations

In this work, the compositional problems are synthesized from existing problems, which, compared to problems created by humans, do not lead to significant new math breakthroughs for the models. If used as training data, it might be more suitable for improving the model's generalization ability. Furthermore, to ensure reliability, the pipeline calls the LLM multiple times for result verification. In the future, with models that have better performance, it may be possible to complete multiple-step reasoning in one go, streamlining the process. Additionally, the compositional problems is more complex and difficult, thus more suitable to use as the training data for reasoning models, like gpt-o1 and Deepseek-r1. We will implement it in the future work.

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, EACL 2024: Student Research Workshop, St. Julian's, Malta, March 21-22, 2024*, pages 225–237. Association for Computational Linguistics.

Shengnan An, Zeqi Lin, Qiang Fu, Bei Chen, Nanning Zheng, Jian-Guang Lou, and Dongmei Zhang. 2023. How do in-context examples affect compositional generalization? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11027–11052, Toronto, Canada. Association for Computational Linguistics.

Lukas Berglund, Meg Tong, Maximilian Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. 2024. The reversal curse: Llms trained on "a is b" fail to learn "b is a". In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3):39:1–39:45.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

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

Gemma-Team. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Shahriar Golchin and Mihai Surdeanu. 2024. Time travel in llms: Tracing data contamination in large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zishan Guo, Renren Jin, Chuang Liu, Yufei Huang, Dan Shi, Supryadi, Linhao Yu, Yan Liu, Jiaxuan Li, Bojian Xiong, and Deyi Xiong. 2023. Evaluating large language models: A comprehensive survey. *CoRR*, abs/2310.19736.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Arian Hosseini, Alessandro Sordoni, Daniel Toyama, Aaron C. Courville, and Rishabh Agarwal. 2024. Not all LLM reasoners are created equal. *CoRR*, abs/2410.01748.

Zhen Huang, Zengzhi Wang, Shijie Xia, and Pengfei Liu. 2024. Olympicarena medal ranks: Who is the most intelligent AI so far? *CoRR*, abs/2406.16772.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. 2024. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 2961–2984. Association for Computational Linguistics.

Yafu Li, Yongjing Yin, Yulong Chen, and Yue Zhang. 2021. On compositional generalization of neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4767–4780. Association for Computational Linguistics.

Zekun Li, Baolin Peng, Pengcheng He, and Xifeng Yan. 2023. Do you really follow me? adversarial instructions for evaluating the robustness of large language models. *CoRR*, abs/2308.10819.

Meta-AI. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Stephen Miner, Yoshiki Takashima, Simeng Han, Ferhat Erata, Timos Antonopoulos, Ruzica Piskac, and Scott J. Shapiro. 2024. Scheherazade: Evaluating chain-of-thought math reasoning in llms with chain-of-problems. *CoRR*, abs/2410.00151.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*.

Mistral-AI. 2024. Mathstral.

Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. 2024. Orca-math: Unlocking the potential of slms in grade school math. *CoRR*, abs/2402.14830.

OpenAI. 2024a. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

OpenAI. 2024b. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

OpenAI. 2024c. Learning to reason with llms.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2080–2094. Association for Computational Linguistics.

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*.

Tomohiro Sawada, Daniel Paleka, Alexander Havrilla, Pranav Tadepalli, Paula Vidas, Alexander Kranias, John J. Nay, Kshitij Gupta, and Aran Komatsuzaki. 2023. ARB: advanced reasoning benchmark for large language models. *CoRR*, abs/2307.13692.

Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Nan Rosemary Ke, Michael Mozer, Yoshua Bengio, Sanjeev Arora, and Anirudh Goyal. 2024. Ai-assisted generation of difficult math questions. *CoRR*, abs/2407.21009.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 31210–31227. PMLR.

Saad Tahmid and Sourav Sarker. 2024. Qwen2.5-32b: Leveraging self-consistent tool-integrated reasoning for bengali mathematical olympiad problem solving. *CoRR*, abs/2411.05934.

Josh Tenenbaum. 2018. Building machines that learn and think like people. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, page 5. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM.

Haoyu Wang, Guozheng Ma, Cong Yu, Ning Gui, Linrui Zhang, Zhiqi Huang, Suwei Ma, Yongzhe Chang, Sen Zhang, Li Shen, Xueqian Wang, Peilin Zhao, and Dacheng Tao. 2023. Are large language models really robust to word-level perturbations? *CoRR*, abs/2309.11166.

Ziyao Xu and Houfeng Wang. 2024. SPOR: A comprehensive and practical evaluation method for compositional generalization in data-to-text generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 604–621, Bangkok, Thailand. Association for Computational Linguistics.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Xiang Yue, Tianyu Zheng, Ge Zhang, and Wenhu Chen. 2024. Mammoth2: Scaling instructions from the web.

In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Hanlin Zhao, Yan Chen, Jiajian Xie, and Kangshun Li. 2023. Swp: A sliding window prompt for emotion recognition in conversation. In *International Conference on Artificial Neural Networks*, pages 293–304. Springer.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2022. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

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

Zihao Zhou, Qiufeng Wang, Mingyu Jin, Jie Yao, Jianan Ye, Wei Liu, Wei Wang, Xiaowei Huang, and Kaizhu Huang. 2024b. Mathattack: Attacking large language models towards math solving ability. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 19750–19758. AAAI Press.

A Datasets Comparison

As shown in Table 3, the accuracy of some existing methods for automatically synthesizing math problems is much lower than ours. Moreover, the LLMs they use for data synthesis is gpt-4o (OpenAI, 2024b) or gpt-4-turbo (OpenAI, 2024a), the cost of which is more than ours. The accuracy of MATH[2] (Shah et al., 2024), GSM-PLUS (Li et al., 2024), and MATHCHECK (Zhou et al., 2024a) comes from the report in the paper, while ORCA (Mitra et al., 2024) is used for training and does not report the accuracy of the problem. Therefore, we randomly sampled 50 samples and evaluated their accuracy. Compare with all of other methods, we do not have such high requirements for LLM capability in synthesizing data.

To better illustrate the difference between our NCSP and current methods for evaluating mathematical robustness, we list some of the approaches used in existing works, as shown in Table 4. In fact, all of these methods generate new problems by creating variants of individual questions. In contrast, our approach considers interactions between problems, establishing connections among two or more questions to generate new ones. This represents the fundamental distinction.

B Experiment Details

Data synthesis. We use vLLM-0.6.4 (Kwon et al., 2023) as our inference backend for data synthesis and model evaluation. It takes 4 hours * 8 A100 to generate 40k compositional problems. When synthesizing data, we set the $seed$ as 42 and the $temperature$ to 1e-2. In the response sampling phase of the evaluation, we set the $temperature$ to 0.8, and set 1e-2 to evaluate whether the answer is correct. Specifically, we set the max_token for all models to 4k, except for the gpt-o1 series and deepseed-r1 series, which are set to 16k. We followed the template used in the Simple Eval[2] project to check if the answers are equal, while we used LLM to extract the final answer instead of pattern matching.

Annotation. We first use a pipeline for synthesis and then manually annotate it. Each problem in VCMD should be verified and annotated at least twice, and the annotators are the authors of this work, each of whom has at least a master's degree. Due to the simplicity of our synthesis method, it does not require highly specialized math knowledge, and it takes us a week to finish the annotation.

C Case

In this section, we show 3 cases from o1-mini-2024-09-12, gpt-4o-2024-11-20, and Deepseek-r1, respectively.

[2]A project created by OpenAI.

Method	Seed Set	Annotation	Pipeline Acc	Use For ?
ORCA (Mitra et al., 2024)	GSM8K	gpt-4o (Q+A)	72%	Training
MATH^2 (Shah et al., 2024)	MATH	gpt-4-turbo (Q+A), Human (Q+A)	77%	Evaluation
GSM-PLUS (Li et al., 2024)	GSM8K	gpt-4-turbo (Q+A), Human (Q+A)	81.15%	Evaluation
MATHCHECK (Zhou et al., 2024a)	GSM8K, GeoQA, UniGeo, Geometry3K	gpt-4-turbo (Q+A), Human (Q+A)	84.61%	Evaluation
Ours NCSP	MATH	Llama3.1-70b-it (Q+A) Human (Q+A)	98.2%	Evaluation

Table 3: Comparison of the success rate of automatic math problem synthesize method.

Method	Pertubation	Description
ORCA (Mitra et al., 2024)	multi-agent-based	Use a suggestor agent to provide the modifing suggestions and then use a editor agent to modified the problems based on it.
MATH^2 (Shah et al., 2024)	skill-based	Use LLMs to extract the skills from seed sets, e.g., arithmetic, and then create new problems based on these skills.
GSM-PLUS (Li et al., 2024)	numerical variation	Alter the numerical data or its types (e.g., from integer to decimal).
	arithmetic variation	Reverse or introduce additional operations, such as addition, subtraction, multiplication, and division, to math problems.
	problem understanding	Rephrase the text description of the math problems.
	distractor insertion	Inserte topic-related but useless sentences to the problems.
	critical thinking	Focus on doubt ability when the question lacks necessary statements.
MATHCHECK (Zhou et al., 2024a)	MATHCHECK-GSM	129 groups checklist matrix, which can be used to evaluate mathematical textual reasoning ability comprehensively.
	MATHCHECK-GEO	A checklist matrix of 60 groups to evaluate the multi-modal geometry reasoning ability.
Ours NCSP	Compositional Problem	Combine two exist problems into a new one with a logical connection to get a new math problems.

Table 4: Comparison of the automatic math problem synthesize method.

Case 1 (About RAWR From gpt-4o)

Origin Problem1:
A regular octagon $ABCDEFGH$ has an area of one square unit. What is the area of the rectangle $ABEF$?
[asy] pair A,B,C,D,I,F,G,H; A=(-1,1); B=(1,1); H=(-2.41,-0.41); C=(2.41,-0.41); G=(-2.41,-2.41); D=(2.41,-2.41); F=(-1,-3.82); I=(1,-3.82); draw(A–B–C–D–I–F–G–H–cycle,linewidth(0.7)); label("A",A,N); label("B",B,N); label("C",C,E); label("D",D,E); label("E",I,S); label("F",F,S); label("G",G,W); label("H",H,W); [/asy]

Origin Problem2:
What is the value of $82^3 + 3(82^2)(18) + 3(82)(18^2) + 18^3$? (No calculators! And see if you can do it in your head!)

Combinational Problems:
A regular octagon $ABCDEFGH$ has an area of one square unit. What is the area of the rectangle $ABEF$?

[asy] pair A,B,C,D,I,F,G,H; A=(-1,1); B=(1,1); H=(-2.41,-0.41); C=(2.41,-0.41); G=(-2.41,-2.41); D=(2.41,-2.41); F=(-1,-3.82); I=(1,-3.82); draw(A–B–C–D–I–F–G–H–cycle,linewidth(0.7)); label("A",A,N); label("B",B,N); label("C",C,E); label("D",D,E); label("E",I,S); label("F",F,S); label("G",G,W); label("H",H,W); [/asy]

Define K is the result of the area. X is $\frac{5}{2}$ more than K.

What is the value of $82^3 + \underline{X}(82^2)(18) + \underline{X}(82)(18^2) + 18^3$? (No calculators! And see if you can do it in your head!)

Solution From gpt-4o:
Step 1: Understand the problem setup We are tasked with determining the area of rectangle $ABEF$ within a regular octagon $ABCDEFGH$ whose total area is 1 square unit. Then, we calculate K, analyze another variable X, and work out a polynomial expression based on X. Let's proceed step by step.
—
Step 2: Find K, the area of rectangle $ABEF$ #### Geometry of the octagon: 1. A regular octagon can be split into a central square and 8 identical isosceles right triangles. 2. Let s be the side length of the octagon. The area of the entire octagon is given as 1 square unit.
Determine the dimensions of the rectangle $ABEF$: - The rectangle $ABEF$ includes two sides of the central square and spans the lengths of adjacent isosceles right triangle bases. - The base of each isosceles right triangle equals the side length s of the octagon. - Thus, the length of rectangle $ABEF$ is $2s$ (the bases of two adjacent triangles) and its width is s (the side of the central square).
Area of rectangle $ABEF$: The area of $ABEF$ is:

$$\text{Area of } ABEF = \text{length} \times \text{width} = (2s)(s) = 2s^2$$

Total area of the octagon: The total area of the octagon is given as 1, which equals the sum of the central square and the 8 isosceles right triangles:

$$\text{Total area} = s^2 + 8 \times \frac{1}{2}s^2 = s^2 + 4s^2 = 5s^2$$

Thus:

$$5s^2 = 1 \quad \Longrightarrow \quad s^2 = \frac{1}{5}$$

Substitute s^2 into the area of $ABEF$:

$$\text{Area of } ABEF = 2s^2 = 2 \times \frac{1}{5} = \frac{2}{5}$$

So, $K = \frac{2}{5}$. **× The correct value of K should be $\frac{1}{2}$**
—
Step 3: Determine X The problem states that X is $\frac{5}{2}$ more than K:

$$X = K + \frac{5}{2} = \frac{2}{5} + \frac{5}{2}$$

To add these fractions, find a common denominator:

$$\frac{2}{5} = \frac{4}{10}, \quad \frac{5}{2} = \frac{25}{10} \quad \Longrightarrow \quad X = \frac{4}{10} + \frac{25}{10} = \frac{29}{10}$$

Thus, $X = \frac{29}{10}$. **× The correct value of X should be 3**
—
Step 4: Simplify the given polynomial expression
The problem asks us to calculate:

$$82^3 + X(82^2)(18) + X(82)(18^2) + 18^3$$

Step 4.1: Factorize the expression
Notice that the expression has a binomial expansion structure:
✓ Here, LLMs derive the correct solution path from wrong reasoning because it matches the pattern of $(82+18)^3$

$$82^3 + X(82^2)(18) + X(82)(18^2) + 18^3 = (82 + 18)^3$$

Step 4.2: Simplify $82 + 18$

$$82 + 18 = 100$$

Step 4.3: Expand 100^3

$$100^3 = 100 \times 100 \times 100 = 1{,}000{,}000$$

—
Final Answer:

$$\boxed{1{,}000{,}000}$$ **✓The Answer is Correct!**

Case 2 (About RAWR From gpt-o1-mini)

Origin Problem1:
Ryan has 3 red lava lamps and 3 blue lava lamps. He arranges them in a row on a shelf randomly, then turns 3 random lamps on. What is the probability that the leftmost lamp on the shelf is red, and the leftmost lamp which is turned on is also red?

Origin Problem2:
If $a = 7$ and $b = 13$, what is the number of even positive integers less than ab?

Combinational Problems:
Ryan has 3 red lava lamps and 3 blue lava lamps. He arranges them in a row on a shelf randomly, then turns 3 random lamps on. What is the probability that the leftmost lamp on the shelf is red, and the leftmost lamp which is turned on is also red?
Define P is the result of the problem. Q is $\frac{133}{20}$ more than P.
If $a = \underline{Q}$ and $b = 13$, what is the number of even positive integers less than ab?

Solution From gpt-o1-mini:
To solve the problem, we need to determine the probability P and subsequently find the number of even positive integers less than ab, where $a = Q$ and $b = 13$.
Step 1: Calculate P
Given: - Ryan has 3 red and 3 blue lava lamps. - He arranges them randomly in a row and turns on 3 lamps at random. - P is the probability that: - The **leftmost lamp** on the shelf is **red**. - The **leftmost lamp that is turned on** is also **red**.
Approach: 1. **Probability that the first lamp is red ($P(A)$):**

$$P(A) = \frac{3}{6} = \frac{1}{2}$$

2. **Probability that the leftmost turned-on lamp is red given that the first lamp is red ($P(B|A)$):** - **Case 1:** The first lamp is turned on.

$$P(\text{first lamp turned on}) = \frac{\binom{5}{2}}{\binom{6}{3}} = \frac{10}{20} = \frac{1}{2}$$

If turned on, it's already red.
- **Case 2:** The first lamp is not turned on.

$$P(\text{leftmost turned on lamp is red}|\text{first lamp not on}) = \frac{13}{20}$$

- Combining both cases:

$$P(B|A) = \frac{1}{2} \times 1 + \frac{1}{2} \times \frac{13}{20} = \frac{33}{40}$$

3. **Calculate P:**

$$P = P(A) \times P(B|A) = \frac{1}{2} \times \frac{33}{40} = \frac{33}{80} \quad \times\ P \text{ should be } \frac{7}{20}$$

Step 2: Calculate Q and ab
Given: - Q is $\frac{133}{20}$ more than P.

$$Q = P + \frac{133}{20} = \frac{33}{80} + \frac{133}{20} = \frac{113}{16} \quad \times\ Q \text{ should be } 7$$

- $a = Q = \frac{113}{16}$
- $b = 13$
Calculate ab:

$$ab = \frac{113}{16} \times 13 = \frac{1469}{16} \approx 91.8125 \quad \times\ ab \text{ should be } 91$$

Step 3: Number of Even Positive Integers Less Than ab
Since $ab \approx 91.8125$, the even positive integers less than ab are:

$$2, 4, 6, \ldots, 90$$

There are $\frac{90}{2} = 45$ such numbers.

Final Answer:
45 ✓**It will get the correct answer as long as $Q \in (\frac{90}{13}, \frac{92}{13}]$.**

Case 3 (Badcase From Deepseek-r1)

Origin Problem1: ✓**Deepseek-r1 solve the problem successfully.**
Find the 3×3 matrix $\mathbf{M}$ such that

$$\mathbf{M} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} y \\ z \\ x + y + z \end{pmatrix}$$

for all real numbers x, y, and z.

Origin Problem2: ✓**Deepseek-r1 solve the problem successfully.**
Given that a particular positive integer is a four-digit palindrome, what is the probability that it is a multiple of 99? Express your answer as a common fraction.

Combinational Problems:
Find the 3×3 matrix $\mathbf{M}$ such that

$$\mathbf{M} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} y \\ z \\ x + y + z \end{pmatrix}$$

for all real numbers x, y, and z.
Define a is the element in the first row and the second column of the matrix $\mathbf{M}$. b is 3 more than a.
Given that a particular positive integer is a b-digit palindrome, what is the probability that it is a multiple of 99? Express your answer as a common fraction.

Solution From Deepseek-r1:
To find the 3×3 matrix $\mathbf{M}$ such that

$$\mathbf{M} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} y \\ z \\ x + y + z \end{pmatrix}$$

we analyze the linear combinations of x, y, and z for each component of the resulting vector. The matrix $\mathbf{M}$ is determined to be:

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

The element a in the first row and the second column is 1. Therefore, $b = a + 3 = 4$. ✓**a and b is correct.**
Next, we consider 4-digit palindromes. A 4-digit palindrome has the form $ABBA$, where A ranges from 1 to 9 and B ranges from 0 to 9. There are $9 \times 10 = 90$ such palindromes.
A number is divisible by 99 if it is divisible by both 9 and 11. For divisibility by 11, all 4-digit palindromes are automatically divisible by 11 because the difference between the sums of the digits in odd and even positions is zero. For divisibility by 9, the sum of the digits $2A + 2B$ must be a multiple of 9, which simplifies to $A + B \equiv 0 \pmod 9$. For each A (1 to 9), there is exactly one B (0 to 9) such that $A + B \equiv 0 \pmod 9$. This gives 9 valid pairs (A, B). Therefore, there are 9 four-digit palindromes divisible by 99.
The probability that a 4-digit palindrome is a multiple of 99 is $\frac{9}{90} = \frac{1}{10}$.

The final answer is $\boxed{\frac{1}{10}}$. × **Final answer should be $\frac{1}{9}$**

D Instruction and Prompt Template

I_c Generate Code to Check Equal

Task Description:

Write a Python program to compare two given values and determine if they are equal. Follow these guidelines:

1. Use the ‘sympy‘ library to handle symbolic comparisons, ensuring that equivalent expressions (e.g., $\frac{2}{4}$ and $\frac{1}{2}$) are recognized as equal.
2. For values involving irrational constants (e.g., π, e), perform comparisons up to **two decimal places** for practical equivalence.
3. Include clear intermediate steps in the program, such as evaluating or simplifying the values where appropriate.
4. Wrap the final comparison outcome in a ‘\boxed‘ command for clarity.
5. Provide both the Python code and the results of running the code.

Output Format:

$'''python$
{The Python code that compares the two given values, including print statements for intermediate steps and the $\boxed{final\_comparison\_outcome}$.}
$'''$

$'''output$
{The output of the Python program.}
$'''$

—
Example1

—
...

—
Example3

—
Task:

Value1: {VALUE1}
Value2: {VALUE2}

I_1 Modify p_1

Given a math problem and the final answer, your task is to output the modified math problem. Follow the steps below:

Step1: Identify a specific integer, float, or fraction within ‘final_answer‘ and name it as $new\_variable1$; There are several situations:

1. If the ‘final_answer‘ contains unknown variables:
   (a) If the ‘final_answer‘ is an expression, choose one coefficient as $new\_variable1$, for example, 2x + 3, you can choose the coefficient of x as $new\_variable1$, which is 2, and in the case of sin(x), there is a hidden coefficient 1 and a hidden amplitude 1, you can choose either one as $new\_variable1$;
   (b) If the ‘final_answer‘ is an equation, you can choose one solution as $new\_variable1$, for example, y = 2x + 1, you can define the value of y as $new\_variable1$ when given x = 1, which is 3;
   (c) If the ‘final_answer‘ is a symbol of an option, such as ’A’, ’B’, ’C’, etc, use their order in the alphabet as a variable, such as ’A’ = 1, ’B’ = 2, ’C’ = 3, etc;
   (d) If the ‘final_answer‘ contains 2 or more items, e.g. multiple choice questions, choose the smallest or the largest one, and then apply the corresponding situation;
2. If the ‘final_answer‘ has no unknown variables, there are several situations:
   (a) If the ‘final_answer‘ itself is a numerical value, like ’four’, ’4’, ’$2+\sqrt{2}$’, ’3π’, and ’$\frac{3}{4}$’, use it directly as $new\_variable1$;
   (b) If the ‘final_answer‘ contains 2 or more numerical values, use the largest or the smallest one as $new\_variable1$;
   (c) If the ‘final_answer‘ is an interval or ratio, choose one boundary and \infty is not allowed, for example, [2,\infty), you can define the lower bound as $new\_variable1$, which is 2;
   (d) If the ‘final_answer‘ is a ratio, choose one part of the ratio, for example, 3:4; you can define the first part of the simplified ratio as $new\_variable1$, which is 3;
   (e) If the ‘final_answer‘ is a non-base 10 number, for example, 1001_2, you can define ’the number of digits in the base 2 representation’ as $new\_variable1$, which is 4;
   (f) If the ‘final_answer‘ is an angle or degree, choose the corresponding radian value, for example, 30\cric or 30°, define the corresponding radian value of final answer as $new\_variable1$, which is \pi/6 or π/6.

All in all, find a way to identify a specific numerical value as $new\_variable1$ without unknown, and make sure the reader can get the value of $new\_variable1$ from the ‘final_answer‘ through your definition.

Step2: Output the value of $new\_variable1$, keep the exact value or math symbol, and simplify the fraction if necessary, for example, keep the π as π, keep the $\sqrt{2}$ as $\sqrt{2}$, and simplify $\frac{6}{8}$ as $\frac{3}{4}$, without rounding to a decimal point.

Step3: Output the definition of $new\_variable1$ without mentioning the real value.

—

Output Format:
<Analysis>
{Identified a specific integer, float, or fraction as $new\_variable1$}
</Analysis>

<The_Value_of_New_Variable1>
{The value of $new\_variable1$, no other text, and output 'None' if you can not find a suitable $new\_variable1$}
</The_Value_of_New_Variable1>

<The_Definition_of_New_Variable1>
{The definition of $new\_variable1$ without mentioning the real value, and output 'None' if you can not find a suitable $new\_variable1$}
</The_Definition_of_New_Variable1>

—

Example1

—

...

—

Example6

—

Task:

Original Problem:
{PROBLEM1}

'final_answer' of Problem:
{FINAL_ANSWER1}

I_1' Verify $new\_variable1$

{PROBLEM1}
Assume that the final answer of the problem is {FINAL_ANSWER1}.
{DEFINITION_OF_NEW_VARIABLE1}

Then what is the value of $new\_variable1$?

Please output the value of $new\_variable1$ directly, wrapping it in \boxed{ }, for example, \boxed{3}.

I_2 Modify p_2

Given a math problem, please identify a specific integer, float, or fraction within it and replace it with $new\_variable2$. The specific steps are as follows:

Step 1: Identify and list all the numerical values in the problem. There are several situations:

1. Containing unknown variables is not allowed, for example, in the case of $2a + 3 = 5$, '2', '3', and '5' are valid, while $2a + 3$ isn't.

2. Containing math symbols is allowed, do not simplify or round them to decimals. For example, keep 3π as 3π, keep $\sqrt{2}$ as $\sqrt{2}$, keep $7!$ as $7!$, and keep $\frac{6}{8}$ as $\frac{6}{8}$.

3. Containing units is not allowed. For example, \$$120\backslash circ$\$ or 120° should be replaced by \${new_variable2}$\backslash circ$\$ where $\{new\_variable2\} = 120$; 10% can become $\{new\_variable2\}\%$ where $\{new\_variable2\} = 10$; and 1000_2 should choose the number of base to be defined as $1000_{new\_variable2}$, etc.

4. Choose part of the expression like coefficient, numerator, or denominator is allowed. For example, the expression $7x + 3$ can be replaced by $\{new\_variable2\} \cdot x + 3$.

Step 2: There are many types of numerical values, and choose one following the priority:

1. Integers like '1' > fractions like $\frac{1}{2}$ or $1/2$ > decimals like 0.5 > numbers in words like 'one', 'two', 'three' > numbers in other bases like 1000_2.

2. Small numbers are preferred when there are multiple numerical values of the same type.

3. If there are no numerical values in the problem, output 'None' in the tag <The_Value_of_new_variable2>.

Output Format:

<Identify_new_variable2>
{Identified a specific integer, float, or fraction as $new\_variable2$}
</Identify_new_variable2>

<The_Value_of_new_variable2>
{The value of $new\_variable2$, no other text, and output 'None' if there are no numerical values in the problem}
</The_Value_of_new_variable2>

<The_Definition_of_new_variable2>
{The definition of $new\_variable2$ without mentioning the real value, and output 'None' if there are no numerical values in the problem}
</The_Definition_of_new_variable2>

<Modified_Problem>
{The modified problem with the new variable symbol $new\_variable2$ without mentioning the real value of $new\_variable2$, and output 'None' if there are no numerical values in the problem}
</Modified_Problem>

—
Example 1:

—
...
—
Example 6:

—
Task:
Original Problem:
{PROBLEM2}

Output:

$I_2^{'}$ Verify $new\_variable2$

Given two math problems, Problem 1 and Problem 2, where a numerical value in Problem 1 has been replaced by a variable $new\_variable2$ to form Problem 2, your task is to identify the value of $new\_variable2$.

Output the results with the following format:

<The_Value_of_new_variable2>
The value of $new\_variable2$, if identified, or 'None' if no value can be determined.
</The_Value_of_new_variable2>

—
Example1

—
...

—
Example3

—
Task:
Original Problem 1:
{PROBLEM2}

Problem 2:
MODIFIED_PROBLEM2

Output:

I_3 Establishing Relationship

Given the values of $new\_variable1$ and $new\_variable2$, your task is to calculate the difference $new\_variable2 - new\_variable1$ and establish the relationship between them. Follow these steps:

Step 1: Write a Python program that calculates the difference between $new\_variable2$ and $new\_variable1$. The program should follow these guidelines:

1. Instead of writing functions, write programs directly.
2. Avoid using decimal values, and ensure that all fractions and square roots are simplified using functions from the 'sympy' library.
3. If there are any intermediate variables, print them in the Python programs.
4. Print the $\boxed{final\_answer}$ at the end.
5. Provide the code output following the Python code.

Step 2: Describe the relationship between $new\_variable1$ and $new\_variable2$ based on the calculated difference.

—

Output Format:

$'''python$
{The Python code that computes '$new\_variable2 - new\_variable1$', including print statements for intermediate variables and the final answer.}
$'''$

$'''output$
{The output of the Python code, including intermediate variables and the final answer.}
$'''$

—
Example1

—
...

—
Example3

—
Task:
The values of $new\_variable1$:
{new_variable1}

The values of $new\_variable2$:
{new_variable2}

$I_3^{'}$ Verify Relationship

Task Description:
Write a Python program to calculate the value of '$new\_variable2$' based on the given value of '$new\_variable1$' and the specified relationship. Follow these guidelines for your program:

1. Avoid using floating-point numbers for intermediate steps; instead, use the 'sympy' library to handle fractions, square roots, and other symbolic representations.
2. Clearly print intermediate steps where appropriate.

3. Ensure that the output clearly shows the value of $new\_variable2$ in its most simplified form.

4. The output should include both the Python program and the corresponding output produced by running the program.

Output Format:

$'''python$
{The Python code that computes $new\_variable2$, including print statements for intermediate calculations and the final result.}
$'''$

$'''output$
{The output of the Python program, showing intermediate steps and the final value of $new\_variable2$.}
$'''$

<The_Value_of_new_variable2>
{The value of '$new\_variable1$' in the problem.}
</The_Value_of_new_variable2>

—
Example 1

—
...

—
Example 4

—
Task:
The value of $new\_variable1$:
{NEW_VARIABLE1}

The relationship:
{RELATIONSHIP}

I_4 Renaming

Given a math problem, your task is to find a new variable name that never appears in the problem for $new\_variable1$ and $new\_variable2$. Note that:

1. The new variable names should be different from all the variables in the problem.

2. The new variable names should wraped in $$, for example, m, α.

—
Output Format:
<The_Symbol_of_new_variable1>
(the new variable name for $new\_variable1$, no other text)
</The_Symbol_of_new_variable1>

<The_Symbol_of_new_variable2>
(the new variable name for $new\_variable2$, no other text)
</The_Symbol_of_new_variable2>

—
Example:

—
Task:
Problem:
{MODIFIED_PROBLEM1}
{MODIFIED_PROBLEM2}

Output:

I_4' Verify Renaming

Given Math Problem 1 and Math Problem 2, confirm whether Problem 2 uses the same variable symbols or object names as Problem 1, focusing on avoiding potential confusion if the two problems were combined.

1. If the same variable symbols (e.g. x, $alpha$, etc) are present in both problems, regardless of whether they have different roles, output 'yes'.
2. If the same objects or entities (e.g. Xiaoming's speed, the number of cakes, Triangle ABC, etc) are mentioned and relevant to the problems, output 'yes'.
3. Otherwise, output 'no'.

Output Format:
<IsContain> yes or no </IsContain>

<Analysis> A brief analysis </Analysis>

—
Example1:

—
...

—
Example3:

—
Task
Problem 1:
{RENAMED_PROBLEM1}

Problem 2:
{RENAMED_PROBLEM2}

Output: