RM-PoT: Reformulating Mathematical Problems and Solving via Program of Thoughts

Changsong Lei[§], Yu Zhang[§]
[§]Tsinghua University, China;
{leics23,yu-zhang23}@mails.tsinghua.edu.cn;

Abstract

Recently, substantial advancements have been made in training language models to carry out step-by-step reasoning for solving intricate numerical reasoning tasks. Beyond the methods used to solve these problems, the structure and formulation of the problems themselves also play a crucial role in determining the performance of large language models. We observe that even small changes in the surface form of mathematical problems can have a profound impact on both the answer distribution and solve rate. This highlights the vulnerability of LLMs to surface-level variations, revealing its limited robustness when reasoning through complex problems. In this paper, we propose RM-POT, a method that first reformulates the surface form of mathematical problems and then applies the Program of Thoughts (PoT) approach to solve them. PoT disentangles computation from reasoning, thereby enhancing the reasoning capabilities of LLMs. Experiments on various datasets demonstrate the effectiveness of our proposed method.

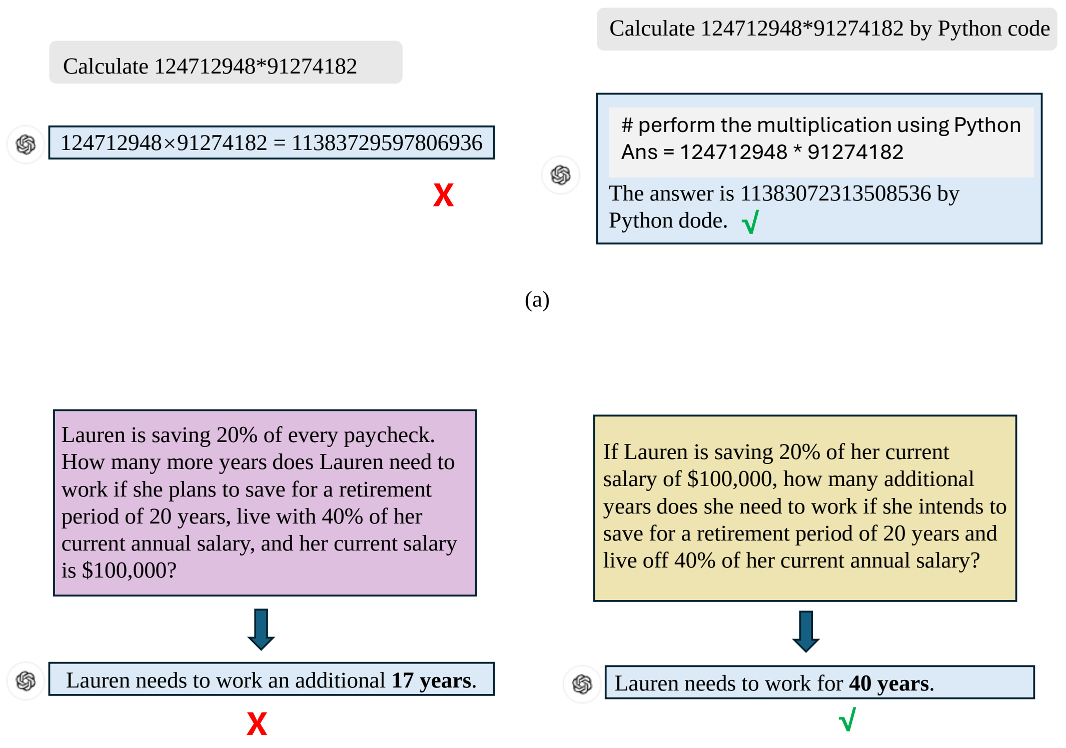
1 Introduction

Mathematical reasoning is a cornerstone of problem-solving, with applications spanning diverse fields such as physics, engineering, economics, and computer science. However, despite their success in general natural language processing tasks, existing Large Language Models (LLMs) such as GPT-4 struggle with mathematical problems that demand precision, logical reasoning, and step-by-step computation [1]. This gap arises because LLMs rely heavily on statistical patterns in natural language, which often fail to capture the formal structure and symbolic complexity of mathematical problems. [2].

Current advancements in mathematical reasoning with LLMs have primarily focused on methods like Chain-of-Thought (CoT) prompting [3, 4, 5], which encourages models to break problems into reasoning steps. Self-Consistency (SC) methods further refine CoT by introducing multiple reasoning paths to identify consistent answers. While effective, these approaches are still limited when faced with problems requiring complex computation or diverse logical forms. As shown in Fig 1(a), the LLM fails to provide the correct answer directly when posed with a complex calculation problem.

This limitation is handled by another notable approach, Program of Thoughts (PoT), which introduces intermediate code generation to represent reasoning steps explicitly, improving the interpretability of solutions and disentagle computation from the reasoning process. However, PoT alone cannot resolve inconsistencies arising from ambiguous structured problem forms. As shown in Fig 1(b), we can observe that small changes in the surface form of the mathematical problem can lead to significantly different outcomes.

To address these challenges, we propose RM-PoT, a novel two-stage framework that integrates Reformulation of Mathematical Problems (RM) and Program of Thoughts (PoT). The RM stage prompts the LLM to generate multiple paraphrased versions of the same problem, thereby mitigating


Calculate 124712948*91274182

124712948×91274182 = 11383729597806936

X

Calculate 124712948*91274182 by Python code

```
# perform the multiplication using Python
Ans = 124712948 * 91274182
```

The answer is 11383072313508536 by Python dode. √

(a)

Lauren is saving 20% of every paycheck. How many more years does Lauren need to work if she plans to save for a retirement period of 20 years, live with 40% of her current annual salary, and her current salary is $100,000?

Lauren needs to work an additional **17 years.**

X

If Lauren is saving 20% of her current salary of $100,000, how many additional years does she need to work if she intends to save for a retirement period of 20 years and live off 40% of her current annual salary?

Lauren needs to work for **40 years**.

√

(b)

Figure 1: Limitations for GPT-4 solving MWPs. (a) When GPT-4 is asked to directly provide the result of a complex arithmetic problem, the answer is incorrect.(b) When the original problem is reformulated, answers are inconsistent.

the adverse effects of poorly structured problem formulations. By generating multiple reformulations of a given problem and exposing the model to various formulations of the same task, LLMs can uncover the core mathematical structure underlying the problem [6]. This diversity in problem presentation improves the LLM's accuracy and consistency in solving the problem. In the PoT stage, the model generates Python code as intermediate reasoning steps, decoupling symbolic computations from natural language reasoning. A voting mechanism ensures consistency across reformulated solutions, enhancing accuracy and robustness.

The main contributions of this paper are as follows:

- We introduce RM-PoT, a novel framework that combines surface-level problem reformulation and explicit intermediate code generation to improve mathematical reasoning in LLMs.
- We conduct comprehensive experiments on widely-used benchmarks, including GSM8K, AQuA, and MATH, demonstrating that RM-PoT outperforms baseline approaches across different datasets.
- We provide analysis of the effectiveness of RM and PoT stages, highlighting their contributions to solving mathematical problems.

2 Related Work

2.1 Mathematical Reasoning in LLMs

In recent years, numerous studies have explored using the System-2 reasoning approach to solve mathematical problems with LLMs [3, 7, 8]. As a prominent framework, chain-of-thought(CoT) is proposed by [3]. Rather than generate the answer directly, it prompts the LLMs to produce a sequence of intermediate reasoning steps. [7] further extended CoT by self-consistency. They generate multiple reasoning steps from different angles and potentially lead to the same answer. [9] proposes program-of-thoughts, using intermediate codes to represent the reasoning process.

2.2 Specialized Models for Mathematical Tasks

To address the limitations of general-purpose LLMs, specialized approaches have been developed. For instance, MathGPT [10] integrates symbolic computation engines to solve algebraic problems, combining language modeling with symbolic reasoning. However, this hybrid approach still depends heavily on the capabilities of external tools. Other models, such as GeoSolver [11] and MathQA [12], focus on specific domains like geometry or math question-answering, using domain-specific datasets and tailored architectures. While these models perform well in their respective areas, their narrow focus limits generalization to broader mathematical tasks.

2.3 Problem Reformulation in Mathematical Problem Solving

The role of problem reformulation in enhancing the performance of language learning models (LLMs) in mathematical problem solving has gained increasing attention in recent years. Early work by [13, 14] demonstrated that altering the structure and phrasing of a problem can lead to improved reasoning outcomes in LLMs. This line of research has been extended by [15], who explored how different surface forms of mathematical problems can significantly influence the solve rate of LLMs. Further studies have investigated the impact of reformulation strategies such as paraphrasing [16] and introducing problem variants [17], suggesting that diverse representations help models better understand and process complex tasks. Building on these insights, recent works have also delved into the combination of reformulation with in-context learning techniques to optimize model performance through adaptive problem presentations.

3 Method

We propose RM-POT, a framework that leverages the potential of large language models (LLMs) to solve mathematical problems. The overall architecture of RM-POT is illustrated in Fig. 2, consisting of two stages.

- **Stage I**: We reformulate the given mathematical problems into diverse surface forms, enabling the LLM to better grasp the underlying structure of the problems.
- **Stage II**: We employ the Program of Thoughts [9] (PoT) approach, which generates Python code to solve the reformulated problems. This approach not only reveals the reasoning process of the LLM but also separates the computational steps from the reasoning process.

The details of these two stages are described in the following subsections.

3.1 Reformulate Mathematical Problems(RM)

As demonstrated in Section 1, the surface form of a problem significantly influences the performance of LLMs [6]. Therefore, we prompt the LLM to generate K different surface forms of the original problem and use a voting mechanism to determine the answer. The intuition behind this approach is that if a problem exhibits a low solve rate and ineffective reasoning paths due to its original surface form, introducing diversity in its surface forms can enhance the likelihood of finding a correct solution.

It is important to note that the original problem is reformulated into different forms using the same LLM that is used to solve the problem. This ensures that the improvement in model performance is due to the diversity of the reformulated problem forms, rather than the sharing of knowledge with other LLMs.

In this paper, we explore two ways of prompting LLM to generate reformulated problems. The naive prompt template is "*Reformulate the following math problem, try to change the sentence structure of the problem*:{input problem}" . The In-Context method begins by identifying effective examples: We conduct experiments to determine which pairs of (Original Problem, Reformulated Problem) exhibit the largest margin in solve rates. Then, we use these examples to enable in-context learning for the LLM, as depicted in Fig. 3.

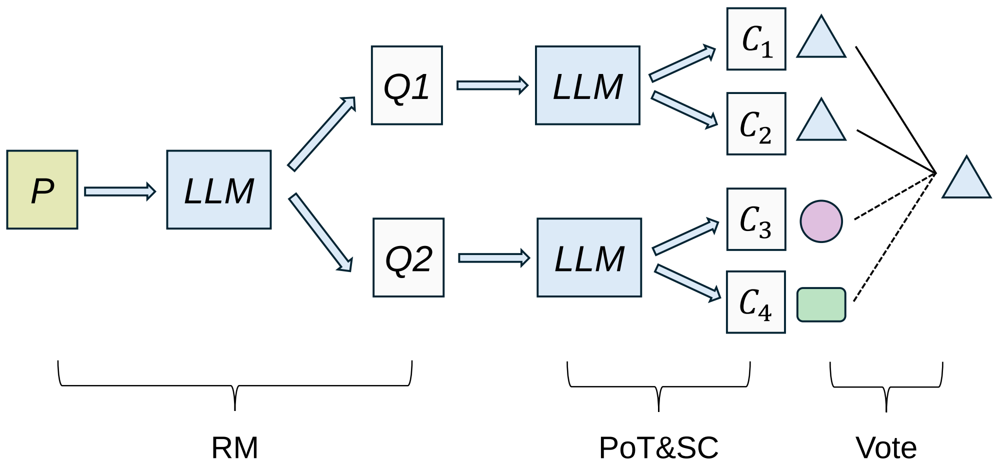


Figure 2: Overview of our proposed RM-PoT framework. It operates in two stages. First, the RM method is employed to reformulate the original problem into various forms using the LLM. Next, the PoT approach is utilized to guide the LLM in generating intermediate code, which not only illustrates the reasoning process but also computes the result. The final answer is then determined through a voting mechanism.

3.2 Program of Thoughts(PoT)

Program of Thoughts (PoT) is a method aimed at enhancing the reasoning capabilities of large language models (LLMs) [9]. It works by decomposing complex tasks into a series of intermediate steps, or "thoughts", which guide the model through a structured reasoning process. Each thought represents a logical step that incrementally leads to the final answer, enabling the model to tackle intricate problems using a step-by-step approach.

This method improves the model's ability to solve tasks that require multi-step reasoning, such as mathematical or logical problems, by fostering transparency in the reasoning process and increasing accuracy in the final result. PoT has proven effective in scenarios where direct answers are difficult, allowing LLMs to perform more reliably in problem-solving tasks.

In our proposed RM-PoT, we aim to instruct the LLM to generate intermediate Python code to solve mathematical problems. This approach can clearly show the reasoning process of LLM and improve accuracy.

3.3 Self-Consistency(SC) and Voting

In our proposed RM-PoT framework, we reformulate the original problems into K different surface forms. Morever, we utilize the Self-Consistency(SC) [7] method. Specifically, for each formulated problem, we let LLM generate $\frac{K}{N}$ reasoning paths, and thus the total number of generated answer is N. We then vote for the final answer.

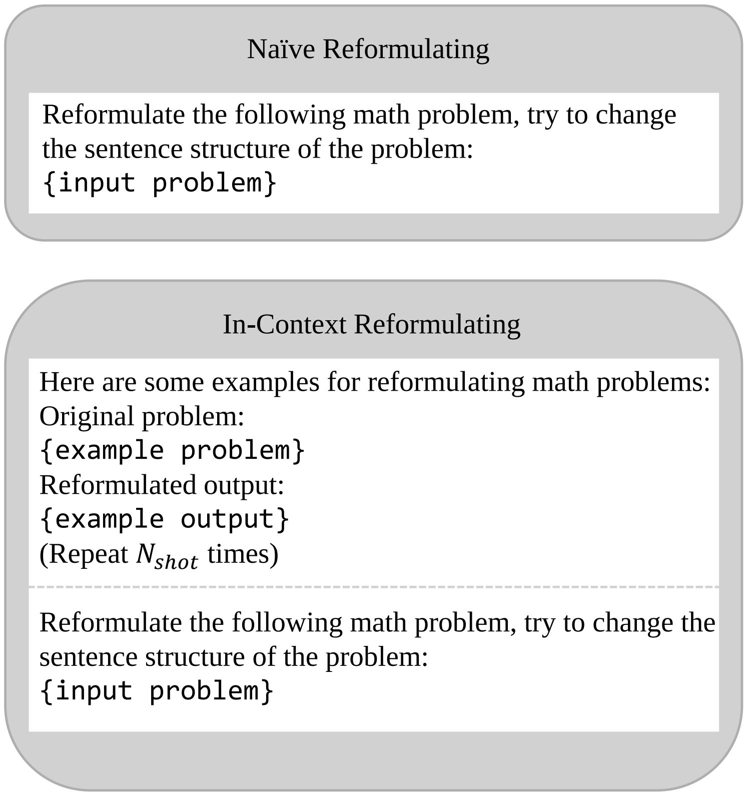
By increasing K, we maintain a fixed total number of reasoning paths N. This approach effectively isolates the effect of diversifying the reasoning paths from merely increasing their number. It also ensures a fair comparison with the self-consistency baselines.

4 Experiments

4.1 Experimental Settings

Datasets We evaluate our approach on the following public mathematics reasoning benchmarks:

- **GSM8k** [2] contains 8.5K linguistically diverse grade school-level math questions with moderate difficulties.


Naïve Reformulating
In-Context Reformulating

Reformulate the following math problem, try to change the sentence structure of the problem:
`{input problem}`

Here are some examples for reformulating math problems:
Original problem:
`{example problem}`
Reformulated output:
`{example output}`
(Repeat N_{shot} times)

Reformulate the following math problem, try to change the sentence structure of the problem:
`{input problem}`

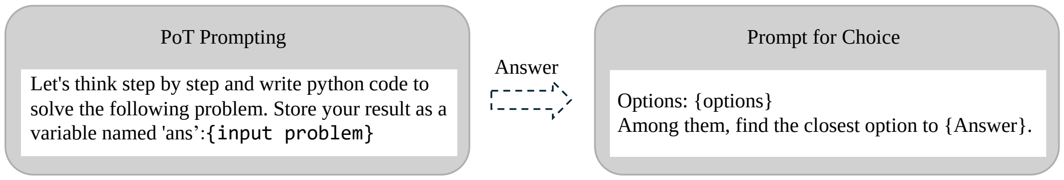
Figure 3: Naive and In-Context Reformulating prompts for LLM.

- **AQuA** [18] consists of 100K algebraic word problems, including the questions,the possible multiple-choice options, and natural language answer rationales from GMAT and GRE.

- **SVAMP** [19] contains 1K arithmetic word problems. It focuses on basic arithmetic operations such as addition, subtraction, multiplication, and division.

Large Language Model We use GLM-4-9B [20] as our base model. All experiments are conducted in zero-shot or few-shot settings, without training or fine-tuning it. For generation configs, We set the temperature $T = 0.7$, Top-p$= 0.8$ and Top-k$= 3$. The total number of reasoning paths N we sample for each problem is 16.

Implementation Details For problem solving, the PoT process consists of two steps, as illustrated in Fig. 4. First, we prompt the LLM to generate Python code and store the result in a variable with a fixed name, allowing for convenient extraction. If the problem includes options, we instruct the LLM to identify the closest match among the provided options.



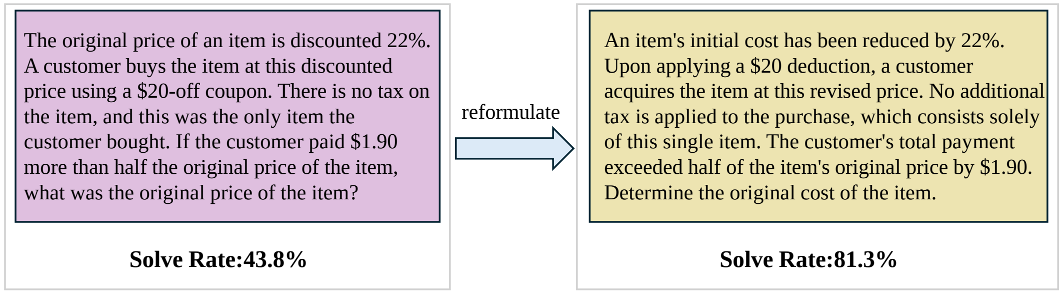
Let's think step by step and write python code to solve the following problem. Store your result as a variable named 'ans':`{input problem}`

Options: {options}
Among them, find the closest option to {Answer}.

Figure 4: Prompts for solving the problem and selecting the correct choice when options are provided.

4.2 Effectiveness of Reformulating

To verify the effectiveness of RM, we reformulate the selected problems from the AQuA dataset and calculate the solve rate difference between the original problems and their reformulated versions. An example is shown in Fig. 5 to provide readers with an intuitive understanding of the RM process. When the surface form of the original problem is altered, the solve rate increases from 43.8% to 81.3%.



The original price of an item is discounted 22%. A customer buys the item at this discounted price using a $20-off coupon. There is no tax on the item, and this was the only item the customer bought. If the customer paid $1.90 more than half the original price of the item, what was the original price of the item?

Solve Rate:43.8%

An item's initial cost has been reduced by 22%. Upon applying a $20 deduction, a customer acquires the item at this revised price. No additional tax is applied to the purchase, which consists solely of this single item. The customer's total payment exceeded half of the item's original price by $1.90. Determine the original cost of the item.

Solve Rate:81.3%

Figure 5: An example from AQuA dataset. When the original problem is reformulated, the solve rate improves significantly.

Furthermore, we compute the solve rate difference between the original problems p and the reformulated problems p_r from the AQuA dataset, defined as where $SR(p_r) - SR(p)$, where SR denotes the solve rate. The results are shown in Fig. 6. we can observe that reformulating the problems leads to an overall improvement in the solve rate.

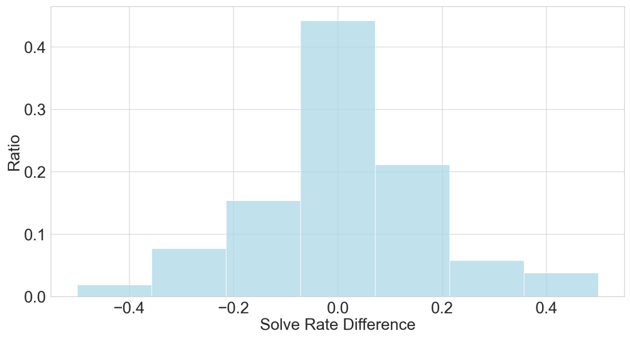


Figure 6: The solve rate difference between the original problems and the reformulated versions.

4.3 Main Results

In this subsection, we compare the performance of RM-PoT with Chain of Thoughts (CoT), vanilla self-consistency (SC), and vanilla Program of Thoughts(PoT). All results are presented in Table 1. In this evaluation, we apply the naive reformulation of the original problems. the number of reformulated problems is set to $K = 4$, and the total number of reasoning paths is $N = 16$. For SC, we also maintain $N = 16$.

The performance metric presented in the table is the accuracy of answers after voting. We can observe that RM-PoT outperforms the other three baselines across all datasets, despite the considerable likelihood of generating reformulation with a lower solve rate (see Fig. 6). We assume that this may be because the model demonstrates more consistent understanding of the problems when exposed to various reformulations, thereby avoiding misinterpretations of the original problem in certain specific cases. We will further discuss it in section 5.

Table 1: Comparison of accuracy between different methods.

	GSM8K	AQuA	SVAMP
CoT	75.6	63.2	86.3
SC	77.3	64.9	87.6
PoT	78.9	65.6	87.1
RM-PoT	**80.4**	**69.1**	**88.0**

4.4 Ablation Study

In this subsection, we vary the number of reformulated problems K across {1,2,4}, while keeping the total reasoning paths fixed at $N = 16$. Additionally, we compare the naive reformulation with the In-Context reformulation, as outlined in Fig 3. The results are shown in Table 2.

Table 2: Comparison of accuracy between different settings of K and reformulation.

	K	GSM8K	AQuA	SVAMP
Naive	1	78.2	66.0	86.9
	2	79.8	67.7	**88.4**
	4	**80.4**	**69.1**	88.0
In-Context	1	78.4	67.6	87.1
	2	80.1	69.4	89.0
	4	**80.9**	**72.2**	**89.6**

We observe that as K increases, the performance of the LLM improves as well. This further validates our assumption that exposing the LLM to different surface forms of a problem allows it to better grasp the underlying structure. Notably, on the SVAMP dataset, the LLM performs better when $K = 2$ than when $K = 4$. This may be because the problems in this dataset are relatively simple, and the process of reformulating and solving the problems involves some degree of randomness.By comparing the naive reformulation with the In-Context reformulation, we can conclude that the LLM learns the reformulation method more effectively when provided with good examples.

5 Discussion

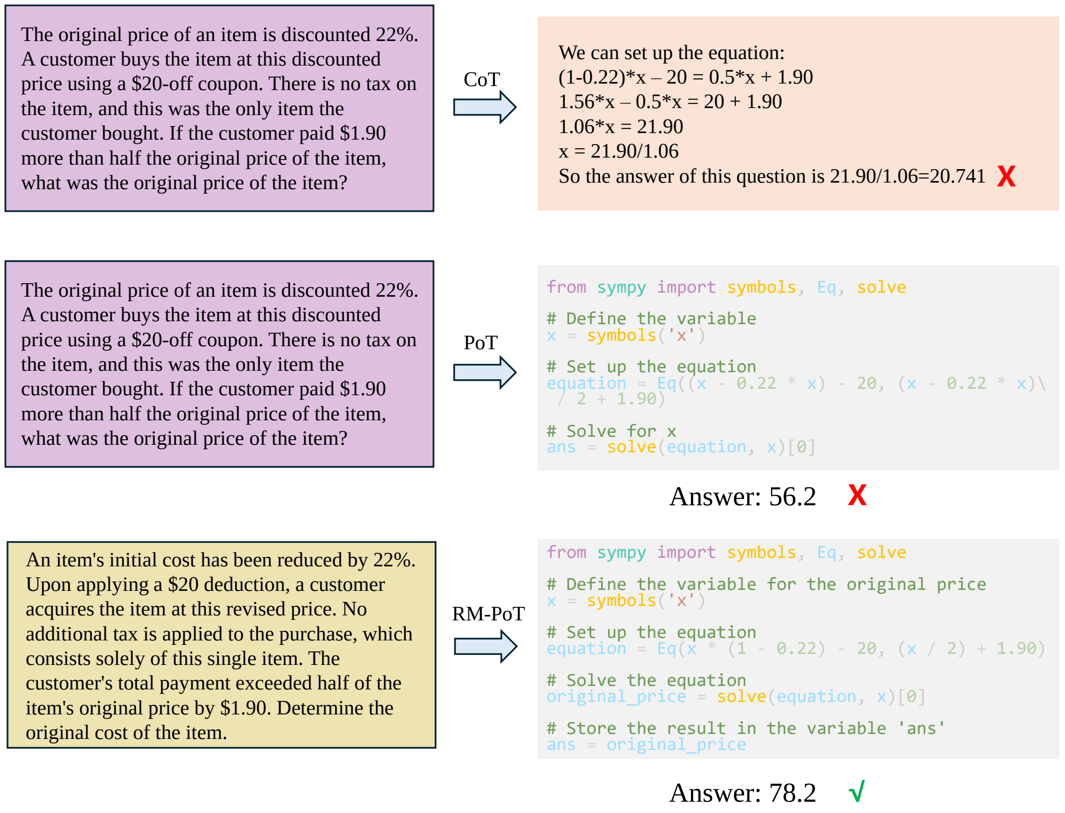
In this section, we discuss how RM-PoT helps the LLM solve math problems through the example shown in Fig. 7. When applying CoT, the LLM makes two mistakes. First, there is an error in rearranging the terms of the equation in the second step.econd, the result of 21.90/1.60 is actually 20.660, not 20.741. This demonstrates that the LLMs has certain shortcomings in computation and reasoning.

To disentangle computaion from reasoning process, we then apply the PoT method. We find that the LLM's answer is generally correct, but it misinterprets the original price as the discounted price during the understanding of the problem. However, after reformulating the original problem, the LLM correctly understands the question and provides the correct answer. Although both versions of the problem contain the term 'original price,' the LLM does not fully grasp this in the original formulation. The deeper reasons behind this remain unclear and will be explored in future work.

6 Conclusion

In this paper, we propose a framework RM-PoT to solve mathematical problems, which combines reformulating original problems and Program of Thoughts. By reformulating, the LLM can better understand the structure of the problem. Through PoT, the computation process is disentangled from reasoning, so the LLM can get precise answer. Experiments prove the effectiveness of our method.

However, the underlying reasons why naive reformulation can enhance performance are still unclear. In some cases, the reformulated problem even exhibits a lower solve rate than the original. Future



The original price of an item is discounted 22%. A customer buys the item at this discounted price using a $20-off coupon. There is no tax on the item, and this was the only item the customer bought. If the customer paid $1.90 more than half the original price of the item, what was the original price of the item?

We can set up the equation:
(1-0.22)*x – 20 = 0.5*x + 1.90
1.56*x – 0.5*x = 20 + 1.90
1.06*x = 21.90
x = 21.90/1.06
So the answer of this question is 21.90/1.06=20.741

The original price of an item is discounted 22%. A customer buys the item at this discounted price using a $20-off coupon. There is no tax on the item, and this was the only item the customer bought. If the customer paid $1.90 more than half the original price of the item, what was the original price of the item?

```
from sympy import symbols, Eq, solve

# Define the variable
x = symbols('x')

# Set up the equation
equation = Eq((x - 0.22 * x) - 20, (x - 0.22 * x)\
 / 2 + 1.90)

# Solve for x
ans = solve(equation, x)[0]
```

Answer: 56.2

An item's initial cost has been reduced by 22%. Upon applying a $20 deduction, a customer acquires the item at this revised price. No additional tax is applied to the purchase, which consists solely of this single item. The customer's total payment exceeded half of the item's original price by $1.90. Determine the original cost of the item.

```
from sympy import symbols, Eq, solve

# Define the variable for the original price
x = symbols('x')

# Set up the equation
equation = Eq(x * (1 - 0.22) - 20, (x / 2) + 1.90)

# Solve the equation
original_price = solve(equation, x)[0]

# Store the result in the variable 'ans'
ans = original_price
```

Answer: 78.2

Figure 7: An example showing how CoT, PoT, and RM-PoT solve the same problem. Both CoT and PoT provide incorrect answers, while RM-PoT gives the correct answer.

work could explore more effective ways to reformulate problems and integrate fine-tuning methods to further improve the performance of LLMs.

References

[1] Dan Hendrycks, Collin Burns, Steven Basart, Spencer Zou, and Mantas Mazeika. Measuring mathematical problem solving with the math dataset. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[2] Karl Cobbe, Vineet Kosaraju, and Mohammad Bavarian. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[3] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[4] Shizhe Diao, Pengcheng Wang, Yong Lin, Rui Pan, Xiang Liu, and Tong Zhang. Active prompting with chain-of-thought for large language models. *arXiv preprint arXiv:2302.12246*, 2023.

[5] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

[6] Zhiheng Zhou, Qianwen Huang, and Shunyu Chen. Self-consistency over paraphrases: Improving the robustness of mathematical reasoning. *arXiv preprint arXiv:2401.00234*, 2024.

[7] Xuezhi Wang, Jason Wei, Dale Schuurmans, and Quoc Le. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[8] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[9] Shunyu Chen, Linjun Zhou, and Tianyi Wang. Program of thoughts prompting: A framework for large language models to reason through programs. *arXiv preprint arXiv:2208.14859*, 2022.

[10] Nikos Scarlatos, Jacky Lin, and Ben Smith. Tree of thoughts: Integrating symbolic computation into language models for math reasoning. *arXiv preprint arXiv:2305.14786*, 2023.

[11] Sean Pedigo, Grace Williams, and Rui Zhang. Geosolver: A framework for solving geometry problems using language models. *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 192–203, 2023.

[12] Aida Amini, Oana Gabriel, and Yejin Choi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *arXiv preprint arXiv:1905.13319*, 2019.

[13] Yue Zhou, Yada Zhu, Diego Antognini, Yoon Kim, and Yang Zhang. Paraphrase and solve: Exploring and exploiting the impact of surface form on mathematical reasoning in large language models. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 2793–2804. Association for Computational Linguistics, 2024.

[14] Ethan Goh, Robert Gallo, Jason Hom, Eric Strong, Yingjie Weng, Hannah Kerman, Joséphine A Cool, Zahir Kanjee, Andrew S Parsons, Neera Ahuja, et al. Large language model influence on diagnostic reasoning: a randomized clinical trial. *JAMA Network Open*, 7(10):e2440969–e2440969, 2024.

[15] Ernest Davis. Mathematics, word problems, common sense, and artificial intelligence. *Bulletin of the American Mathematical Society*, 61(2):287–303, 2024.

[16] Zaibo Long and Jinfen Xu. Investigating teacher reformulations in efl classroom interaction: An ecological perspective. *Journal of Language, Identity & Education*, pages 1–16, 2023.

[17] Xin Wang, Hong Chen, Zihao Wu, Wenwu Zhu, et al. Disentangled representation learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[18] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017.

[19] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 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*, pages 2080–2094, Online, June 2021. Association for Computational Linguistics.

[20] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. Chatglm: A family of large language models from glm-130b to glm-4 all tools, 2024.