Get an A in Math: Progressive Rectification Prompting

Zhenyu Wu[1], **Meng Jiang**[2], **Chao Shen**[1]

[1]School of Cyber Science and Engineering, Xi'an Jiaotong University
[2]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556
zhenyuwu@stu.xjtu.edu.cn, mjiang2@nd.edu, chaoshen@xjtu.edu.cn

Abstract

Chain-of-Thought (CoT) prompting methods have enabled large language models (LLMs) to generate reasoning paths and solve math word problems (MWPs). However, they are sensitive to mistakes in the paths, as any mistake can result in an incorrect answer. We propose a novel method named Progressive Rectification Prompting (PRP) to improve average accuracy on eight MWP datasets from 77.3 to 90.5. Given an initial answer from CoT, PRP iterates a verify-then-rectify process to progressively identify incorrect answers and rectify the reasoning paths. With the most likely correct answer, the LLM predicts a masked numerical value in the question; if the prediction does not match the masked value, the answer is likely incorrect. Then the LLM is prompted to regenerate the reasoning path hinted with a set of incorrect answers to prevent itself from repeating previous mistakes. PRP achieves the best performance compared against the CoT methods. Our implementation is made publicly available at https://wzy6642.github.io/prp.github.io/.

Introduction

Math word problems (MWPs) require language comprehension, mathematical reasoning, and problem-solving skills. Studying these problems helps AI researchers develop algorithms and models that can mimic human-like reasoning and problem-solving abilities (Chen et al. 2022). Chain-of-thought (CoT) prompting methods help large language models (LLMs) break down complex problems into manageable parts, allowing them to focus on each part individually (Kojima et al. 2022). The LLMs become decent zero-shot reasoners by simply adding "*Let's think step by step*" to generate reasoning paths and predict answers to the MWPs (Shi et al. 2023; Wang et al. 2023a,b; Zheng et al. 2023).

When analyzing the performance of existing methods, we found that the average accuracy on eight standard datasets (e.g., MultiArith, GSM8K) was 77.3, far below A-level grades. Because they have three drawbacks: (1) lack of verification that checks if the answer is correct, (2) lack of rectification that finds the correct answer being aware of mistakes, and (3) lack of an effective method that progressively refines reasoning path, which are essential "exam skills."

First, to distinguish correct and incorrect answers, existing methods repeatedly solve a problem and use a majority vote strategy to determine the most consistent answer as the correct answer. This is known as self-consistency (Wang et al. 2023b). However, since it solves the same problem multiple times, this repeated independent process leads to same mistakes, making the frequent answer still incorrect. Second, existing methods such as progressive-hint prompting (Zheng et al. 2023) modify reasoning paths by adding "*(Hint: The answer is near* $[\mathsf{H}]$*)*" after the given problem, where $[\mathsf{H}]$ is the slot of previous answers. It is evident that when previous answers are incorrect, LLMs may still generate an incorrect answer in response to the hint. Third, existing CoT prompting methods exhibit high sensitivity to mistakes in intermediate reasoning steps (Kojima et al. 2022; Chen et al. 2022; Wang et al. 2023a; Shi et al. 2023). Even a tiny mistake in the reasoning process could alter the entire problem-solving process, resulting in an incorrect answer. It is nontrivial to achieve multi-step precise reasoning.

To address the three drawbacks of existing methods, our research is inspired by a guide on math study skills and exam success written by Gall et al. in 1990. First, *substitute verification* has been commonly used in math exams to verify the correctness of an answer. Let us look at a specific example. Given an equation $2+3=y$, after solving it, we find that the answer y equals 5. Next, we introduce a masked condition to formulate the masked equation $X+3=y$ and substitute the answer 5 into this equation. Solving the masked equation, if X equals 2, it indicates that the answer 5 align with the original equation. So the answer was more likely to be correct. Otherwise, the answer could probably be incorrect. Compared to repeatedly checking, such as solving the same question multiple times, the substitute verification can effectively prevent the repetition of mistakes and improve the accuracy of answer verification. Second, relying on existing progressive hints such as "*the answer is near* $[\mathsf{H}]$" would limit the exploration of other potential answers when the hint answer was incorrect. Suppose that $[\mathsf{H}]$ has been found less likely correct by substitute verification. A negation hint on it, like "*the answer is likely* ***not*** $[\mathsf{H}]$", will help LLMs eliminate or less consider such answers, so the LLMs will actively rectify their reasoning paths to avoid mistakes. Third, the *dual process theory* in psychology (Evans 2003) tells us that humans have two cognitive systems to progressively re-

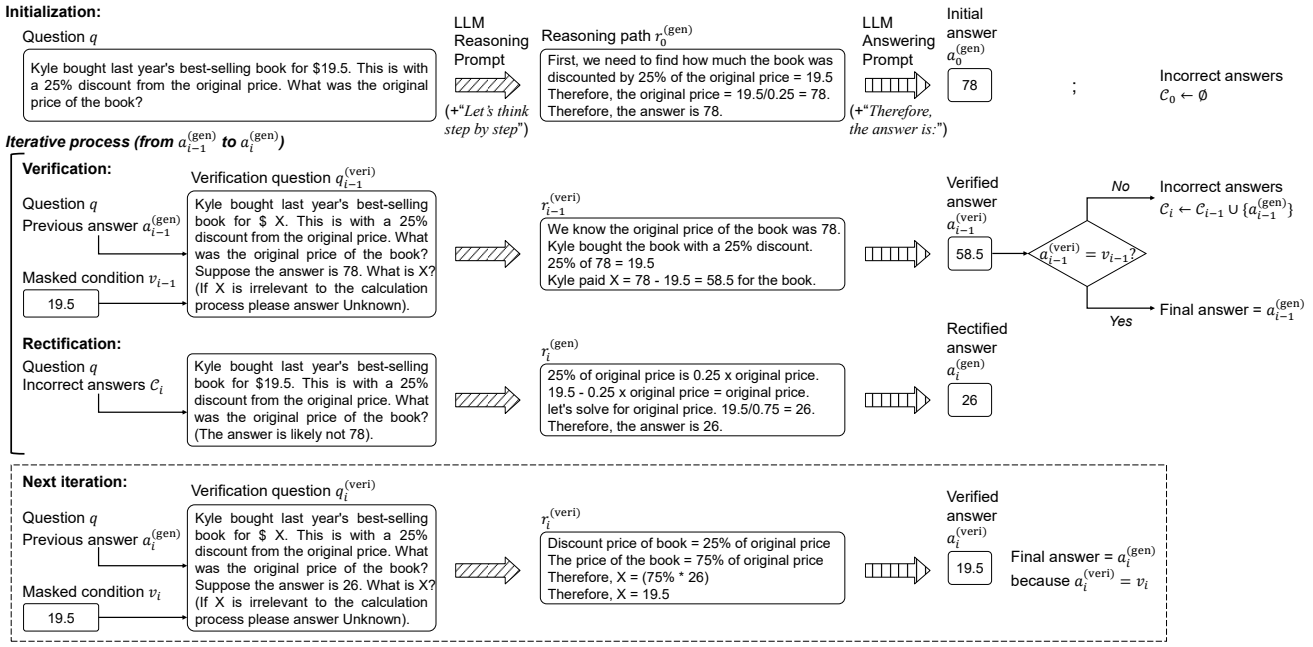

Figure 1: Overview of Progressive Rectification Prompting (PRP) method. PRP first generates an initial answer. PRP then iterates a verify-then-rectify process to progressively rectify the LLM-generated answer to find the correct one.

fine their answers, plans, and solutions. One provides initial responses based on intuition; the other provides a deliberate and reflective approach to progressive refine those initial responses. Existing CoT prompting methods possess only the capability of the first system, while lacking the capacity for progressive refinement of answers through the second type.

In this paper, we propose a novel zero-shot prompting method to implement and integrate the above ideas to improve the performance of LLMs on MWPs. We name it Progressive Rectification Prompting (PRP). Figure 1 illustrates PRP with an example from the GSK8K dataset. In PRP, an initial answer is generated by a standard zero-shot prompt (Kojima et al. 2022). Then PRP feeds the question and initial answer into an iterative *verify-then-rectify* process. It progressively rectifies the LLM-generated answer to find the correct one. The verify-then-rectify process consists of a *verification* module and a *rectification* module. The verification module uses the substitute verification method to verify the correctness of the answer. It masks a numerical value in the question, takes the previous generated answer as a conclusion, and uses it as a new condition. If the masked value is predicted incorrectly, the answer is added to the set of potentially incorrect answers. The rectification module designs a hint that uses the set of potentially incorrect answers as feedback to rectify previous answers. In Figure 1, the initial numerical answer was 78. Next, PRP used a regular expression to match all numerical values within the question. Then it randomly selected one of these values (i.e., 19.5 in this example) and replaced its occurrence in the question with a special token X. This converted the known condition in the original question into an unknown condition, resulting in the masked question. Subsequently, we rewrote the masked question using a simple template to form the verification question: "[Q] Suppose the answer is [A]. What is X? (If X is irrelevant to the calculation process please answer Unknown)", where [Q] was the slot for the masked question, and [A] was the slot for previous generated answer. If the answer did not match the masked condition, the previous generated answer would be considered less likely correct and added to a set of potentially incorrect answers. In rectification, we added the phrase "(The answer is likely not [H])" after the given question, where [H] was the slot for the set of potentially incorrect answers. The LLM avoided repeating previous mistakes when re-answering the question using the set of potentially incorrect answers as feedback. In most cases, the LLM got the correct answer with a single rectification. But to deal with complex arithmetic questions, PRP had to iterate the verify-then-rectify process to progressively rectify the answer.

Experiments on text-davinci-003 demonstrate that the proposed PRP method improves over existing prompting methods by a striking margin across eight MWP datasets. Our method attains an average score of 90.5, significantly higher than 77.3 from the best of zero-shot CoT, and even higher than 81.0 from the best of few-shot CoT. Our PRP equips LLMs with high-level math exam skills.

The main contributions are summarized as follows:

- We propose a novel zero-shot prompting method that enables LLMs to progressively rectify the generated answer and accurately solve math word problems. It has an iterative verify-then-rectify process to avoid repeating previous mistakes and achieve continuous improvement.

- We conduct extensive experiments on eight math word problem datasets under zero-shot and few-shot CoT settings. Notably, our method achieves the state-of-the-art performance and attains an A-level grade on average.

Related Work

Math Word Problem Solving

Our work is related to existing efforts in solving math word problems (MWPs). Traditional methods used statistical learning-based approaches to extract entities, quantities, and operators from a question and generated an arithmetic equation to find the answer (Hosseini et al. 2014; Roy, Vieira, and Roth 2015; Zhou, Dai, and Chen 2015; Mitra and Baral 2016). Recent methods based on sequence-to-sequence (Seq2Seq) model and recurrent neural networks directly transformed the question into an arithmetic equation (Wang, Liu, and Shi 2017; Wang et al. 2019) . However, their generated equations could be invalid or unsolvable. Besides, recent efforts fine-tuned pre-trained language models on a variety of downstream tasks (Shen et al. 2021; Liang et al. 2022, 2023), which significantly improved the validity of generated equations and brought substantial performance improvements over Seq2Seq models. These methods require a significant amount of human annotations, lacking the ability to generalize to new MWP datasets. In this work, we aim to directly prompt the LLMs to answer arbitrary MWPs without human annotation and task-specific fine-tuning. Our method can generate reasoning paths that enable researchers to investigate model behavior.

Chain-of-Thought Prompting Methods

Our work is related to Chain-of-Thought (CoT) prompting methods, which enable LLMs to generate reasoning paths and solve MWPs. Two types of CoT prompting methods have been proposed: zero-shot prompting (Kojima et al. 2022) and few-shot prompting. By adding "*Let's think step by step*" after the question and feeding the modified question to the LLMs, the LLMs can generate complex reasoning paths. However, zero-shot CoT prompting suffers from missing-step errors. To mitigate these errors, Plan-and-Solve (PS) prompting method instructed the LLMs to devise a plan for breaking down the entire task into smaller subtasks, and then carry out the subtasks according to the plan (Wang et al. 2023a). All these methods are based on manually writing instructions, to eliminate human labor, Zhang et al. proposed Auto-Instruct to automatically improve the quality of instructions provided to LLMs. Manual-CoT (Wei et al. 2022), as a type of few-shot promopting, designed effective manual demonstrations to elicit multi-step reasoning ability of LLMs. Program of Thought (PoT) (Chen et al. 2022) used LLMs to generate programming language statements, and used a program interpreter to execute the generated program to get the final answer. To leverage the benefit of demonstration examples and minimize manual effort, Zhang et al. designed Auto-CoT. By sampling questions with diversity and generating reasoning path to automatically construct demonstrations. Yu et al. introduced IfQA, a dataset for counterfactual reasoning, which requires models to identify the right information for retrieval and inference. These methods are sensitive to mistakes in reasoning paths, and any mistake can result in an incorrect answer. Our method iterates a verify-then-rectify process to progressively identify incorrect answers and rectify reasoning paths.

Notation	Definition
q	Math word problem
$q_i^{(\text{veri})}$	Verification question
$r_i^{(\text{gen})}$	LLM-generated reasoning path for question q
$a_i^{(\text{gen})}$	LLM-generated answer for question q
$r_i^{(\text{veri})}$	LLM-generated reasoning path for question $q_i^{(\text{veri})}$
$a_i^{(\text{veri})}$	LLM-generated answer for question $q_i^{(\text{veri})}$
v_i	Masked condition
$\mathcal{C}_i$	The set of potentially incorrect answers
[Q]	Slot for question
[R]	Slot for reasoning path
[A]	Slot for answer
[H]	Slot for set of potentially incorrect answers

Table 1: Notations and their definitions.

Answer Selection

Several studies have trained models to evaluate candidate answers and select the best answer as the final response. For example, Kushman et al. trained a classifier to select the best answer from candidate answers. Roy and Roth trained a relevance classifier and a lowest common ancestor operation classifier. The distributional output of these classifiers was used in a joint inference procedure to determine the final answer. Shen et al. jointly trained a candidate expression generator and a candidate expression ranker to get better answers. Cobbe et al. fine-tuned GPT-3 as a scorer to calculate solution-level verification score and choose the highest score answer as the final answer. All these methods require massive human annotations. In contrast, our method automatically verifies the correctness of LLM-generate answers and selects the answer that has been verified.

Proposed Method

Overview

We propose a novel zero-shot prompting method named Progressive Rectification Prompting (PRP) for solving math word problems. Figure 1 illustrates the PRP method. Given a question q, PRP prompts the LLM to generate the final answer. Specifically, it first prompts the LLM to generate an initial answer $a_0^{(\text{gen})}$ and initializes the set of potentially incorrect answers as an empty set $\mathcal{C}_0 = \varnothing$. Then, it iterates the verify-then-rectify process up to K iterations to progressively rectify the LLM-generated answer. This process consists of a verification module and a rectification module. In the i-th iteration, the verification module uses the substitute verification method to verify the correctness of the previous generated answer $a_{i-1}^{(\text{gen})}$. If the answer $a_{i-1}^{(\text{gen})}$ is verified likely to be incorrect, add $a_{i-1}^{(\text{gen})}$ to the set of potentially incorrect

answers $\mathcal{C}_{i-1}$ to obtain the updated set $\mathcal{C}_i$. Otherwise, take $a_{i-1}^{(\text{gen})}$ as the final answer. The rectification module uses the set of potentially incorrect answers $\mathcal{C}_i$ as feedback to rectify previous answers and generate the rectified answer $a_i^{(\text{gen})}$. If the number of iterations exceeds the maximum iteration K, take the last LLM-generated answer $a_K^{(\text{gen})}$ as the final answer. In the following sections, we will elaborate on the details of each component. Table 1 presents a list of the notations used throughout this paper.

Initialization

During initialization, PRP initializes the set of potentially incorrect answers as an empty set $\mathcal{C}_0 = \varnothing$ and prompts the LLM to generate an initial answer $a_0^{(\text{gen})}$ for the given question q. Specifically, we first construct a reasoning generation prompt: "Q: [Q]. A: Let's think step by step", where [Q] is the slot for question q. We then feed the above prompt to the LLM, which subsequently generates a reasoning path $r_0^{(\text{gen})}$.

To extract the answer from the reasoning path, we add the answer extraction instruction after the reasoning path to devise the answer generation prompt: "[R] Therefore, the answer (expressed in Arabic numerals and without units) is:", where [R] is the slot for reasoning path $r_0^{(\text{gen})}$. Finally, we feed the answer extraction prompt to the LLM to generate the initial answer $a_0^{(\text{gen})}$ for the question q.

Iterative Verify-then-Rectify Process

We propose a novel iterative verify-then-rectify method that progressively rectifies the LLM-generated answer over K iterations by cyclic execution of the verification and rectification modules. The iteration process would terminate early if the LLM-generated answer is verified likely to be correct. Here we take the i-th iteration as an example to illustrate the verify-then-rectify process.

Verification Module The verification module uses substitute verification method to verify the correctness of the previous generated answer $a_{i-1}^{(\text{gen})}$. It comprises several substeps.

Firstly, we utilize the condition mask method (Weng et al. 2022) to create a masked question. Specifically, we first use a regular expression to match all numerical values within the question q. We then randomly select one of these values v_{i-1} and replace its occurrence in the question q with a special token X, resulting in the masked question.

Secondly, we rewrite the masked question using a simple template to form the verification question $q_{i-1}^{(\text{veri})}$: "[Q] Suppose the answer is [A], what is X? (If X is irrelevant to the calculation process please answer Unknown)", where [Q] is the slot for masked question, and [A] is the slot for previous generated answer $a_{i-1}^{(\text{gen})}$.

Thirdly, we feed the reasoning generation prompt "Q: [Q]. A: Let's think step by step" into the LLM to generate a reasoning path $r_{i-1}^{(\text{veri})}$ for the verification question $q_{i-1}^{(\text{veri})}$, where [Q] is the slot for question $q_{i-1}^{(\text{veri})}$. Furthermore, we feed the answer generation prompt "[R] Therefore, the answer (expressed in Arabic numerals and without units) is:" into the LLM to generate the answer $a_{i-1}^{(\text{veri})}$ for the verification question $q_{i-1}^{(\text{veri})}$. Where [R] is the slot for reasoning path $r_{i-1}^{(\text{veri})}$.

Finally, we check if $a_{i-1}^{(\text{veri})}$ is equal to v_{i-1}. If they are equal, it indicates that the previous generated answer $a_{i-1}^{(\text{gen})}$ is most likely correct. We select $a_{i-1}^{(\text{gen})}$ as the final answer and exit the loop. Otherwise, the previous generated answer $a_{i-1}^{(\text{gen})}$ is likely incorrect, and we add $a_{i-1}^{(\text{gen})}$ to the set of potentially incorrect answers $\mathcal{C}_{i-1}$ to obtain the updated set $\mathcal{C}_i$.

Rectification Module The rectification module uses a set of potentially incorrect answers $\mathcal{C}_i = \{a_0^{(\text{gen})}, \cdots, a_{i-1}^{(\text{gen})}\}$ as feedback to generate a rectified answer $a_i^{(\text{gen})}$. Specifically, we first devise an answer rectification prompt: "Q: [Q] (The answer is likely not [H]) A: Let's think step by step", where [Q] is the slot for the question q, and [H] is the slot for the set of potentially incorrect answers $\mathcal{C}_i$. We then feed the above prompt into the LLM to generate a rectified reasoning path $r_i^{(\text{gen})}$. Finally, we feed the prompt "[R] Therefore, the answer (expressed in Arabic numerals and without units) is:" into the LLM to generate the rectified answer $a_i^{(\text{gen})}$ for the question q. Where [R] is the slot for reasoning path $r_i^{(\text{gen})}$.

Answer Selection The process of verify-then-rectify can be iterated until specific stopping conditions are met. The process terminates under two situations. The first is when the answer $a_{i-1}^{(\text{gen})}$ is verified likely to be correct. In this case, we select answer $a_{i-1}^{(\text{gen})}$ as the final answer. The second situation is when the number of iterations exceeds the maximum iteration K. In this case, we choose the last LLM-generated answer $a_K^{(\text{gen})}$ as the final answer.

Experiments

Experimental Setup

Datasets. We conduct comprehensive experiments on eight math word problem datasets, including AddSub (Hosseini et al. 2014), SingleOp (Roy, Vieira, and Roth 2015), MultiArith (Roy and Roth 2015), SingleEq (Koncel-Kedziorski et al. 2015), SVAMP (Patel, Bhattamishra, and Goyal 2021), GSM8K (Cobbe et al. 2021), GSM-IC2-1K (Shi et al. 2023), and GSM-ICM-1K (Shi et al. 2023). Table 3 provides the detailed descriptions of each dataset.

Baselines. We compare our method with six baseline methods: Direct (Kojima et al. 2022), Zero-Shot-CoT (Kojima et al. 2022), Plan-and-Solve (PS) (Wang et al. 2023a), Manual-CoT (Wei et al. 2022), Auto-CoT (Zhang et al. 2023b), and Progressive-Hint Prompting (PHP-CoT) (Zheng et al. 2023). The Direct baseline concatenates a question with the prompt "The answer is" as the LLM input.

Implementation. We use text-davinci-003 as the backend large language model, which is one of the most widely-used LLMs with public APIs[1]. The few-shot baselines, including Manual-CoT (Wei et al. 2022), Auto-CoT (Zhang et al.

[1]Public API available at https://openai.com/api/.

Setting	Method (text-davinci-003)	Dataset								Average
		AddSub	MultiArith	SVAMP	GSM8K	SingleEq	SingleOp	GSM-IC2-1K	GSM-ICM-1K	
Zero-Shot	Direct	89.3	25.8	65.2	15.0	84.6	92.1	22.8	9.0	50.5
	Zero-Shot-CoT	84.8	87.0	74.3	60.8	89.5	89.1	70.7	62.5	77.3
	PS	88.1	87.2	72.0	58.2	89.2	89.5	70.9	63.5	77.3
	PRP (Ours)	**94.7**	**96.3**	**86.2**	**73.6**	**96.5**	**96.1**	**93.1**	**87.1**	**90.5**
Few-Shot	Manual-CoT	87.8	91.5	76.7	56.9	91.3	93.7	73.9	60.6	79.1
	Auto-CoT	90.6	95.1	77.8	58.9	90.9	94.4	74.3	65.2	80.9
	PHP-CoT	91.1	94.0	81.3	57.5	93.5	94.5	75.3	60.9	81.0

Table 2: Accuracy comparison on eight math word problem datasets. The best and second best results are boldfaced and underlined, respectively. All indicators are presented in percentages.

Dataset	# Problems	Avg.# Words	# IC
SingleEq	508	27.4	0.0%
MultiArith	600	31.8	0.0%
SingleOp	562	20.9	0.0%
AddSub	395	31.5	30.9%
SVAMP	1,000	31.8	36.7%
GSM8K	1,319	46.9	6.2%
GSM-IC2-1K	1,000	41.8	100.0%
GSM-ICM-1K	1,000	61.4	100.0%

Table 3: Statistics of datasets. # IC Indicates the percentage of problems with irrelevant context in the statement.

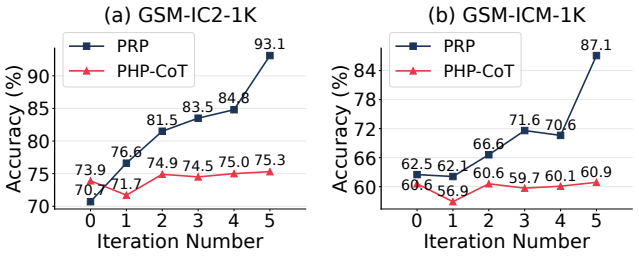

Figure 2: Accuracy (%) at different number of iterations.

2023b), and PHP-CoT (Zheng et al. 2023) employ demonstration examples as suggested in the original papers. Regarding the evaluation metric, we use accuracy to evaluate the performance of MWP solving.

Experimental Results

PRP attains an A-level grade on average. Table 2 reports the accuracy comparison of PRP with existing zero-shot and few-shot methods on MWP datasets. Notably, PRP achieves state-of-the-art performance with an average accuracy of 90.5 on eight MWP datasets. Compared to other zero-shot prompting methods, PRP demonstrates a remarkable improvement in accuracy, surpassing them by at least 13.2% on all datasets. Specifically, PRP achieves a substantial accuracy gain of 24.6% over Zero-Shot-CoT on the GSM-ICM-1K dataset. Even when compared to the competitive zero-shot baseline PS, the PRP maintains an impressive performance. PRP outperforms PS on all eight MWP datasets, with an average accuracy improvement of 13.2%. These results demonstrate that, in contrast to existing zero-shot prompting methods, which solve the problem only once and are sensitive to mistakes in the reasoning path, the PRP method progressively rectifies the answer generated by the LLM to find the correct one. As a result, PRP equips the LLM with high-level math exam skills.

While comparing with few-shot prompting methods, PRP achieves an accuracy improvement of at least 9.5% across all datasets. Notably, PRP enhances problem-solving accuracy for the GSM8K, GSM-IC2-1K, and GSM-ICM-1K datasets by 16.1%, 17.8%, and 26.2% respectively when compared to PHP-CoT. These results demonstrate that PRP significantly enhances the LLM's ability to solve MWPs without the need for manually designed demonstrations.

Iterative verify-then-rectify process progressively improves accuracy. Figure 2 demonstrates the accuracy improvements of both PRP and PHP-CoT as the number of iterations increases. Notably, PRP exhibits a significantly higher rate of improvement compared to PHP-CoT. Specifically, for the GSM-IC2-1K dataset, PRP achieves a remarkable accuracy improvement of 22.4% after five iterations, resulting in an accuracy of 93.1%, compared to using the initial answer as the final answer, which only yields an accuracy of 70.7%. In contrast, PHP-CoT, which relies on progressive hints, shows a much smaller improvement in accuracy. After five iterations, PHP-CoT achieves an accuracy improvement of 1.4%, resulting in an accuracy of 75.3%, compared to using the initial answer as the final answer, which yields an accuracy of 73.9%. PHP-CoT relies on progressive hints such as "the answer is near to [H]" which can limit the exploration of other potential answers when the hint answer [H] is incorrect. In contrast, PRP uses an iterative verify-then-rectify process to progressively identify incorrect answers and rectify the reasoning paths. This iterative process ensures a constant improvement in accuracy and allows PRP to outperform PHP-CoT in terms of accuracy enhancement.

The more complex problems in the dataset, the more iterations are needed. Figure 4(b) illustrates the average iteration number of PRP across all eight MWP datasets. For datasets such as SingleOp, MultiArith, and SingleEq, the average number of iterations is less than 2.5. This is because, as shown in Table 3, the problem statements in these

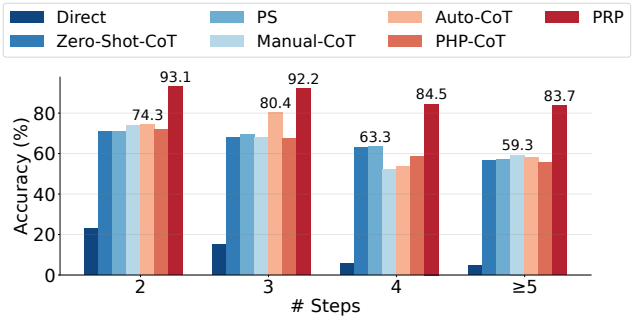

Figure 3: Accuracy on GSM-IC-2K with respect to the number of required reasoning steps. The GSM-IC-2K dataset is formed by merging the GSM-IC2-1K dataset and the GSM-ICM-1K dataset. # Steps indicating the number of reasoning steps in the standard answer.

Method	GSM8K	SVAMP
Zero-Shot-CoT + SC	70.7	81.7
PRP (Ours)	**73.6**	**86.2**

Table 4: Accuracy comparison of PRP to Zero-Shot-CoT with self-consistency (SC) on GSM8K and SVAMP.

datasets are shorter and contain no irrelevant context. As a result, the PRP method can quickly obtain the final answer within a few iterations. In contrast, the PRP method requires more iterations on the SVAMP, GSM8K, GSM-IC2-1K and GSM-ICM-1K datasets. This can be attributed to longer problem statements and more irrelevant context in the problems. Specifically, PRP requires an average of 3.59 and 4.1 iterations on the GSM-IC2-1K and GSM-ICM-1K datasets, respectively. This is because each question in these two datasets contains irrelevant context, and PRP requires more iterations to gradually eliminate incorrect answers to obtain more correct one. These findings suggest that PRP demonstrates a high efficiency in obtaining the final answer for simpler problems. However, when faced with more complex problems, PRP needs to iterate the verify-then-rectify process multiple times to progressively rectify the answer and achieve accurate results.

PRP can effectively solve difficult MWPs. To explore the relationship between the accuracy of model predictions and problem difficulty, we combined two datasets, GSM-IC2-1K and GSM-ICM-1K, into a merged dataset named GSM-IC-2K. The difficulty of problems was classified into four levels based on the number of reasoning steps[2]. Figure 3 illustrates the accuracy of solving problems at different difficulty levels. PRP outperforms current state-of-the-art prompting method by 18.8%, 11.8%, 21.2%, and 24.4% for problems of increasing difficulty levels, respectively. The results demonstrate that PRP notably enhances accuracy in solving MWPs, particularly for challenging problems.

[2]The number of reasoning steps of a problem is given by the number of sentences in its standard answer. (Cobbe et al. 2021)

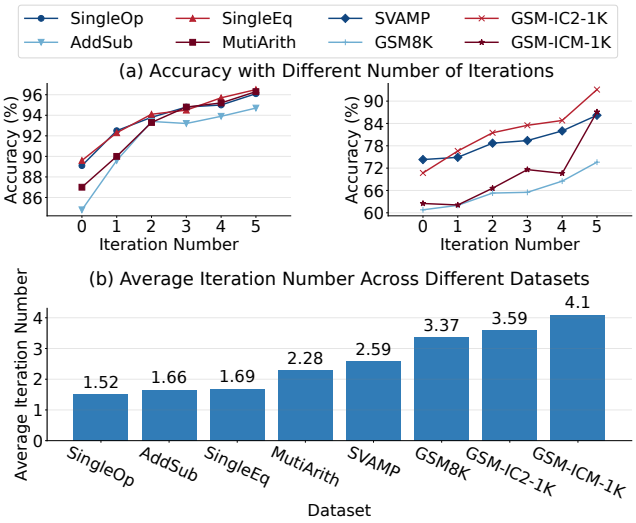

Figure 4: Break-down analysis of PRP. (a) Accuracy (%) of PRP method on different datasets with different number of iterations. (b) The average number of iterations for PRP method across different datasets.

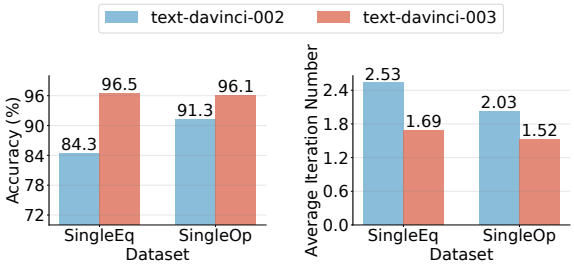

Figure 5: Accuracy and average number of iterations when solving MWPs using PRP with different LLMs.

Ablation Studies

Progressive rectification outperforms self-consistency. Self-consistency (SC) (Wang et al. 2023b) is the process of repeatedly solving a problem M times and using a majority vote strategy to determine the most consistent answer as the final answer. We evaluate Zero-Shot-CoT prompting with SC on the GSM8K and SVAMP datasets. We set the temperature to 0.7 and set M to 10 for the SC experiments. Table 4 demonstrates that PRP outperforms Zero-Shot-CoT prompting with SC on both the GSM8K and SVAMP datasets, achieving accuracy improvements of 2.9% and 4.5%, respectively. This superiority of PRP can be attributed to its progressive rectification method, which prevents the repetition of previous mistakes and enhances the accuracy of the final answer. In contrast, SC solves the same problem multiple times, this repeated independent process can lead to same mistakes, making the frequent answer still incorrect.

PRP performs better with more powerful LLMs. Figure 5 illustrates the accuracy and average number of iterations when solving MWPs using the PRP method with dif-

QUESTION	Amy is collecting candy for the car pool. She has 5 chocolate bars, 7 times as many M&Ms as chocolate bars, and 6 times as many marshmallows as M&Ms. Amy then arranges the candies into baskets. Ada ate 25 more cookies than Amy. If Amy fills each basket with 10 candies, how many baskets will Amy fill?
ANSWER	$(5 + 5 \times 7 + 5 \times 7 \times 6)/10 =$**25**
Direct	Amy will fill 10 baskets. ✗
Zero-Shot-CoT	Amy has 5 chocolate bars, 7 times as many M&Ms as chocolate bars, and 6 times as many marshmallows as M&Ms. 5 chocolate bars = 35 M&Ms and 210 marshmallows Amy has 35 M&Ms, 210 marshmallows, and 5 chocolate bars. Amy will fill 10 candies in each basket, so she will make: 35/10 = 3 baskets of M&Ms 210/10 = 21 baskets of marshmallows 5/10 = 0.5 baskets of chocolate bars Therefore, Amy will fill 24.5 baskets. Ada ate 25 more cookies, so Ada ate 25 baskets of cookies. ✗
PS	Plan: 1. Calculate the total number of candies that Amy has 2. Calculate the number of baskets that Amy can fill Solution: 1. Amy has 5 chocolate bars, 7 M&Ms for every chocolate bar, and 6 marshmallows for every M&M. Therefore, Amy has 5 x 7 x 6 = 210 candies in total. 2. Amy can fill each basket with 10 candies. Therefore, Amy can fill 210 candies in 21 baskets (210/10 = 21). ✗
Manual-CoT	Amy has 5 chocolate bars, 7 x 5 = 35 M&Ms and 6 x 35 = 210 marshmallows. So she has 5 + 35 + 210 = 250 candies. Ada ate 25 more than Amy, so Ada ate 25 + 250 = 275. Amy will fill 275 candies into 10 candies per basket. 275 / 10 = 27.5. The answer is 28 baskets. ✗
Auto-CoT	Amy has 5 chocolate bars, 7 times as many M&Ms as chocolate bars, and 6 times as many marshmallows as M&Ms. That means she has 5 + (7 * 5) + (6 * 7) = 65 candies. Amy then arranges the candies into baskets. If Amy fills each basket with 10 candies, she can fill 65/10 = 6.5 baskets. Since she can fill half a basket, she can only fill 6 baskets. The answer is 6. ✗
PHP-CoT	Amy has 5 chocolate bars, 7 * 5 = 35 M&Ms, and 6 * 35 = 210 marshmallows. So she has 5 + 35 + 210 = 250 candies. Ada ate 25 more than Amy, so Amy has 250 - 25 = 225 candies. If she fills each basket with 10 candies, then she will have 225 / 10 = 22.5 baskets. The answer is 23. ✗
PRP (Ours)	Amy has 5 chocolate bars, 7 x 5 = 35 M&Ms, and 6 x 35 = 210 marshmallows. So in total she has 250 candies. She will fill 10 candies in each basket, so Amy will be able to fill 250/10 = 25 baskets. ✓

Table 5: Case study of answers generated by different methods. The gold answer is indicated in bold.

ferent LLMs. For instance, on the SingleEq dataset, using the text-davinci-003 model leads to a 12.2% increase in accuracy and a reduction of 0.84 in the average number of iterations, compared to the text-davinci-002 model. Similarly, on the SingleOp dataset, using the text-davinci-003 model results in a 4.8% accuracy improvement and a 0.51 reduction in average number of iterations compared to text-davinci-002 model. It is worth noting that the text-davinci-002 is finetuned with supervised instruction tuning, while the text-davinci-003 is finetuned with reinforcement learning (Zheng et al. 2023). The improved performance with text-davinci-003 can be attributed to its enhanced power, making it better at understanding and employing the given prompt.

Maximum iteration number. Figure 4(a) shows that as the number of iterations increases, the accuracy improves across all datasets. We set the maximum iteration number K to 5. Note that the bigger maximum iteration number K may lead to better performance, but here we set it to 5 to achieve a trade-off between efficiency and effectiveness.

Case Study

PRP exhibits robustness in handling irrelevant context. A real case from GSM-ICM-1K is presented in Table 5. It is evident that apart from PRP, other methods cannot accurately answer the given question. Manual-CoT and PHP-CoT generate incorrect answers by incorporating irrelevant context into the problem-solving process. Auto-CoT and PS generate incorrect answers due to semantic misunderstandings. Zero-Shot-CoT generates an incorrect answer due to miscalculations. As Direct does not generate intermediate reasoning steps, it is not possible to analyze the reasons for its mistakes. In contrast, PRP exhibits robustness in handling irrelevant context and preventing miscalculations. Additionally, PRP has the ability to uncover hidden details in the problem statement, such as the fact that "Chocolate bars, M&Ms, and marshmallows are all candies".

Conclusion

In this paper, we present a novel zero-shot prompting method for solving math word problems. We name it progressive rectification prompting (PRP), which first prompts a large language model to generate an initial answer, then iterates a verify-then-rectify process to progressively identify incorrect answers and rectify the reasoning paths. Notably, it attains an A-level grade on average (90.5), significantly higher than 77.3 from the best of zero-shot CoT, and even higher than 81.0 from the best of few-shot CoT.

Acknowledgments

We thank the anonymous reviewers for their insightful feedback and constructive comments. This work was partially supported by National Key R&D Program of China (2020AAA0107702), National Natural Science Foundation of China (U21B2018, 62161160337, 62132011, 62376210, 62006181, U20B2049), Shaanxi Province Key Industry Innovation Program (2021ZDLGY01-02), Fundamental Research Funds for the Central Universities under grant (xtr052023004, xtr022019002). Chao Shen is the corresponding author. Co-author Meng Jiang consulted on this project on *unpaid weekends* for personal interests, and appreciated collaborators and family for their understanding.

References

Chen, W.; Ma, X.; Wang, X.; and Cohen, W. W. 2022. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *arXiv preprint arXiv:2211.12588*.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; Hesse, C.; and Schulman, J. 2021. Training Verifiers to Solve Math Word Problems. *CoRR*, abs/2110.14168.

Evans, J. S. 2003. In two minds: dual-process accounts of reasoning. *Trends in Cognitive Sciences*, 7(10): 454–459.

Gall, M. D.; et al. 1990. *Tools for Learning: A Guide to Teaching Study Skills.* ERIC.

Hosseini, M. J.; Hajishirzi, H.; Etzioni, O.; and Kushman, N. 2014. Learning to Solve Arithmetic Word Problems with Verb Categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 523–533. Doha, Qatar: Association for Computational Linguistics.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large Language Models are Zero-Shot Reasoners. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems*, volume 35, 22199–22213. Curran Associates, Inc.

Koncel-Kedziorski, R.; Hajishirzi, H.; Sabharwal, A.; Etzioni, O.; and Ang, S. D. 2015. Parsing Algebraic Word Problems into Equations. *Transactions of the Association for Computational Linguistics*, 3: 585–597.

Kushman, N.; Artzi, Y.; Zettlemoyer, L.; and Barzilay, R. 2014. Learning to Automatically Solve Algebra Word Problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 271–281. Baltimore, Maryland: Association for Computational Linguistics.

Liang, Z.; Zhang, J.; Wang, L.; Qin, W.; Lan, Y.; Shao, J.; and Zhang, X. 2022. MWP-BERT: Numeracy-Augmented Pre-training for Math Word Problem Solving. In *Findings of the Association for Computational Linguistics: NAACL 2022*, 997–1009. Seattle, United States: Association for Computational Linguistics.

Liang, Z.; Zhang, J.; Wang, L.; Wang, Y.; Shao, J.; and Zhang, X. 2023. Generalizing Math Word Problem Solvers via Solution Diversification. In *AAAI*.

Mitra, A.; and Baral, C. 2016. Learning To Use Formulas To Solve Simple Arithmetic Problems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2144–2153. Berlin, Germany: Association for Computational Linguistics.

Patel, A.; Bhattamishra, S.; and Goyal, N. 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*, 2080–2094. Online: Association for Computational Linguistics.

Roy, S.; and Roth, D. 2015. Solving General Arithmetic Word Problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1743–1752. Lisbon, Portugal: Association for Computational Linguistics.

Roy, S.; Vieira, T.; and Roth, D. 2015. Reasoning about Quantities in Natural Language. *Transactions of the Association for Computational Linguistics*, 3: 1–13.

Shen, J.; Yin, Y.; Li, L.; Shang, L.; Jiang, X.; Zhang, M.; and Liu, Q. 2021. Generate & Rank: A Multi-task Framework for Math Word Problems. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, 2269–2279. Punta Cana, Dominican Republic: Association for Computational Linguistics.

Shi, F.; Chen, X.; Misra, K.; Scales, N.; Dohan, D.; Chi, E.; Schärli, N.; and Zhou, D. 2023. Large Language Models Can Be Easily Distracted by Irrelevant Context. In *Proceedings of the 40th International Conference on Machine Learning*.

Wang, L.; Xu, W.; Lan, Y.; Hu, Z.; Lan, Y.; Lee, R. K.-W.; and Lim, E.-P. 2023a. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2609–2634. Toronto, Canada: Association for Computational Linguistics.

Wang, L.; Zhang, D.; Zhang, J.; Xu, X.; Gao, L.; Dai, B. T.; and Shen, H. T. 2019. Template-Based Math Word Problem Solvers with Recursive Neural Networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press. ISBN 978-1-57735-809-1.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q. V.; Chi, E. H.; Narang, S.; Chowdhery, A.; and Zhou, D. 2023b. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*.

Wang, Y.; Liu, X.; and Shi, S. 2017. Deep Neural Solver for Math Word Problems. In *Proceedings of the 2017 Confer-*

ence on Empirical Methods in Natural Language Processing, 845–854. Copenhagen, Denmark: Association for Computational Linguistics.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; ichter, b.; Xia, F.; Chi, E.; Le, Q. V.; and Zhou, D. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems*, volume 35, 24824–24837. Curran Associates, Inc.

Weng, Y.; Zhu, M.; He, S.; Liu, K.; and Zhao, J. 2022. Large Language Models are reasoners with Self-Verification. *arXiv preprint arXiv:2212.09561*.

Yu, W.; Jiang, M.; Clark, P.; and Sabharwal, A. 2023. IfQA: A Dataset for Open-domain Question Answering under Counterfactual Presuppositions. In Bouamor, H.; Pino, J.; and Bali, K., eds., *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 8276–8288. Singapore: Association for Computational Linguistics.

Zhang, Z.; Wang, S.; Yu, W.; Xu, Y.; Iter, D.; Zeng, Q.; Liu, Y.; Zhu, C.; and Jiang, M. 2023a. Auto-Instruct: Automatic Instruction Generation and Ranking for Black-Box Language Models. In Bouamor, H.; Pino, J.; and Bali, K., eds., *Findings of the Association for Computational Linguistics: EMNLP 2023*, 9850–9867. Singapore: Association for Computational Linguistics.

Zhang, Z.; Zhang, A.; Li, M.; and Smola, A. 2023b. Automatic Chain of Thought Prompting in Large Language Models. In *The Eleventh International Conference on Learning Representations (ICLR 2023)*.

Zheng, C.; Liu, Z.; Xie, E.; Li, Z.; and Li, Y. 2023. Progressive-Hint Prompting Improves Reasoning in Large Language Models. arXiv:2304.09797.

Zhou, L.; Dai, S.; and Chen, L. 2015. Learn to Solve Algebra Word Problems Using Quadratic Programming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 817–822. Lisbon, Portugal: Association for Computational Linguistics.