FILL IN THE BLANK: EXPLORING AND ENHANCING LLM CAPABILITIES FOR BACKWARD REASONING IN MATH WORD PROBLEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

While *forward reasoning* (i.e., find the answer given the question) has been explored extensively in the recent literature, backward reasoning is relatively unexplored. We examine the *backward reasoning* capabilities of LLMs on Math Word Problems (MWPs): given a mathematical question and its answer, with some details omitted from the question, can LLMs effectively retrieve the missing information?

In this paper, we formally define the backward reasoning task on math word problems and modify three datasets to evaluate this task: GSM8k, SVAMP and MultiArith. Our findings show a significant drop in the accuracy of models on backward reasoning compared to forward reasoning across four SOTA LLMs (GPT4, GPT3.5, PaLM-2, and LLaMa). Utilizing the specific format of this task, we propose three novel techniques that improve performance: *Rephrase* reformulates the given problem into a forward reasoning problem, *PAL-Tools* combines the idea of Program-Aided LLMs to produce a set of equations that can be solved by an external solver, and *Check your Work* exploits the availability of natural verifier of high accuracy in the forward direction, interleaving solving and verification steps. Finally, realizing that each of our base methods correctly solves a different set of problems, we propose a novel Bayesian formulation for creating an ensemble over these base methods aided by a verifier to further boost the accuracy by a significant margin. Extensive experimentation demonstrates that our techniques successively improve the performance of LLMs on the backward reasoning task, with the final ensemble-based method resulting in a substantial performance gain compared to the raw LLMs with standard prompting techniques such as chain-of-thought.

1 INTRODUCTION

Large language models (LLMs) (Brown et al., 2020; OpenAI, 2023; Anil et al., 2023) have shown remarkable versatility, excelling in various tasks like sentence completion, question answering, and summarization. They have been successfully applied to mathematical reasoning, specifically in solving *Math Word Problems* (Kushman et al., 2014; Roy & Roth, 2018), where the goal is to produce the answer given an elementary school-level mathematics question. We refer to this task as *Forward Reasoning*. This problem has received significant attention in the recent literature (Lu et al., 2022), and specific datasets (Cobbe et al., 2021; Roy & Roth, 2015; Patel et al., 2021) have been proposed to serve as benchmarks for this task. Performance of powerful LLMs such as GPT-4 OpenAI (2023) with techniques such as Chain-of-Thought (Wei et al., 2022) and Self-Verification (Zhou et al., 2023) on some of these datasets are more than 90% (Lu et al., 2022).

In this work, we are interested in a slightly different problem, which is the problem of *backward reasoning*: given an MWP, with one of the numerical quantities omitted from the question, and the answer to the original question, what is the value of the omitted numerical quantity? While this problem of backward reasoning has been studied in the literature in the context of improving the performance of forward reasoning (Weng et al., 2022), to the best of our knowledge, there is no existing work that explicitly aims to solve this problem analyzing its hardness and providing solutions thereof. We believe this is an interesting problem because (1) It is a matter of study that even for humans, whether forward reasoning and backward reasoning have different complexities (Ramful & Olive, 2008; Rivera, 2008), and we would like to ask the same question in the context of LLMs (2)

Assuming we establish that backward reasoning is a harder problem, how can we design techniques to improve performance on this task, that specifically exploit the problem structure of backward reasoning and the availability of the forward direction answer? (3) The backward reasoning problem can be seen as a special case of abduction, with a unique answer, and it is interesting to explore this connection, since LLMs have not been explored as much for this important class of abductive reasoning problems (Bhagavatula et al., 2020; Qin et al., 2020; 2022).

In order to establish that backward reasoning is indeed a significantly harder problem than forward reasoning, we create backward reasoning problems from three benchmark datasets (Cobbe et al. (2021); Roy & Roth (2015); Patel et al. (2021)), and test the performance of various LLMs on this task. Table 1 presents the results (see Section 3 for details). We observe a significant drop in the performance of all the LLMs compared to forward reasoning, the drop being as high as 40% in most cases. Having established the difficulty of our proposed task, we propose a set of techniques for improving the performance on this task

Firstly, we propose to rephrase the backward reasoning as a forward reasoning problem since LLMs are better at forward reasoning compared to backward reasoning. This can be done by substituting some unknown quantity x in the blank, and then asking LLM to produce the value of x, given the answer. Next, we combine the idea of Program-Aided-Language Models, with an external solver since backward reasoning problems have a specific mathematical structure, which is not amenable to simple forward reasoning. We refer to this method as *PAL-Tools*. Thirdly, we can use the LLM as a verifier to check whether the solution to the backward problem is correct. Verification can be done iteratively until the backward reasoning model produces an answer which is declared correct by the verifier. We refer to this approach as *Check your Work*. Finally, we make use of Bayesian modeling to propose a novel method for ensembling base methods, which exploits the accuracy of each of the base methods computed on a small hold-out set, to compute answer probability. Further, we use a high-accuracy verifier based on forward reasoning and compute the probability of the final answer given the ensemble probability and the confidence of the verifier in the answer.

Our results show the benefit obtained by each of our base methods, with Verify and Reprompting performing best overall. Finally, we obtain a further improvement of up to 10% using our Bayesian model, which ensembles techniques using a verifier. The overall gain in accuracy obtained by our methods touches 45% for some of the datasets, compared to directly running the original backward problem on the LLM. We perform additional analysis on the models to explain our results.

To summarize our contributions: (1) we create three different datasets for backward reasoning, establish the difficulty of the backward reasoning problem, and a task of interest in its own right (2) We propose three different base methods, and one ensemble-based method via the use of a verifier, to improve the performance on the backward task. (3) We perform additional analysis giving further insights into the performance of the proposed models.

2 RELATED WORK

A mathematical word problem (MWP) (Lu et al., 2022) consists of a description in natural language that expresses the relation between various entities and quantities, followed by a query for an unknown quantity, as shown in Figure 1. One can answer the question by representing the relationship between the entities and quantities through a set of equations and then solving these equations. Solving MWPs necessitates a semantic understanding of the natural language description. Initial works (Kushman et al., 2014; Koncel-Kedziorski et al., 2015; Roy & Roth, 2018) to solve MWPs involve parsing the natural language description and utilizing statistical learning techniques to identify suitable templates for generating answers. Subsequently, following the triumph of sequence-to-sequence (Seq2Seq) neural models (Sutskever et al., 2014) in machine translation and other NLP tasks, the encoder-decoder framework (Wang et al., 2017; Ling et al., 2017; Li et al., 2020; Shen et al., 2021; Jie et al., 2022) is employed to directly translate the natural language description in MWPs into equations.

Recently, the strongest performance on MWPs has been given by large pre-trained language models like GPT-4 OpenAI (2023), PaLM (Anil et al., 2023), and GPT-3 (Brown et al., 2020). These models leverage the power of few-shot in-context examples and employ prompting methods like CoT (Wei et al., 2022), all without requiring any modifications to their parameters. One class of

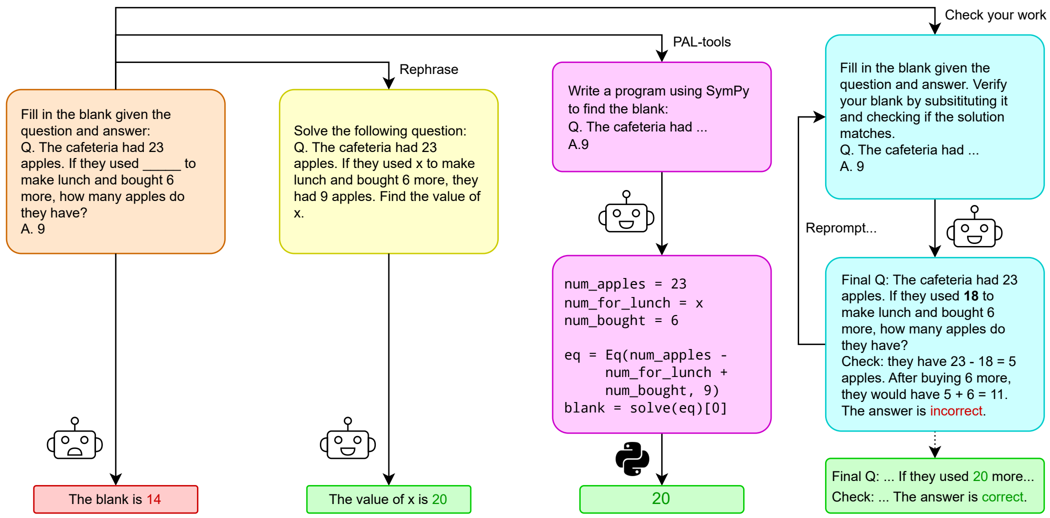

Figure 1: A summary of the prompting techniques we propose

Table 1: Performance of various models on the Math backward reasoning task, compared to their accuracies on the forward reasoning task of solving the original problem. Figures with † are taken from Zheng et al. (2023)

	GSM8k		SVAMP		MultiArith	
Model	forward	backward	forward	backward	forward	backward
GPT-4	92.8	38.6	90.5†	43.9	97.8†	54.8
GPT-3.5-turbo	58.4	10.8	79.1	20.4	97.0	13.8
PaLM-2	60.5	15.2	73.7	11.2	95.7	6.3
LLaMa-2-70B	37.0	6.8	70.3	20.3	89.2	11.0

techniques (Madaan et al., 2023; Welleck et al., 2023) using LLMs involves verifying the answer provided by the Language Model, either using the model itself or external verifiers such as compilers or proof checkers. If the answer is incorrect, the model is re-prompted, optionally with suggestions on improving its output. This prompting continues until the model generates the correct output. Other techniques, such as Progressive Hint Prompting (Zheng et al., 2023) iteratively pass the model's previous answers to itself as hints. Iterative prompting techniques like (Wang et al., 2023) do not use a verifier; instead, they sample multiple hypotheses from the model and select the answer using majority voting.

Our work can be seen as a special case of abductive reasoning with a unique answer. Abductive reasoning (Bhagavatula et al., 2020; Qin et al., 2020; 2022) involves inferring which of several explanations is the most plausible. Prior work on abductive reasoning has focused mostly on text-based reasoning under constraints. In the context of arithmetic reasoning tasks Weng et al. (2022) has utilized abductive reasoning to enhance forward reasoning accuracy. In contrast, our work addresses backward reasoning as an independent problem. Our primary interest lies in analyzing the inherent complexities of backward reasoning and devising more effective solutions to tackle it.

3 Backward Reasoning Task

A forward or the typical Mathematical Word Problem (MWP) consists of a question Q_f, which we call a forward question, and its corresponding answer A_f. The forward question is a textual representation of the MWP. It is typically composed of one or more sentences and encompasses various elements, including numbers, operations, and textual information, all represented by tokens within the question. A *numeric token* is defined as a token that encodes a numeric quantity, such as

5, 37, or 'half.' These tokens encode the values 5, 37, and 0.5, respectively. We define a backward MWP as a tuple (Q, A_f), where Q is obtained from the forward question Q_f by replacing one numerical quantity with a blank. The goal of solving the backward MWP is to find out the value of the numerical quantity that was blanked out using backward reasoning. By backward reasoning, we mean the process of using the provided answer A_f and the context provided by the question Q to deduce the missing numerical quantity to arrive at the given answer. Since there is a unique answer for every question, we measure accuracy on this task by the number of questions on which the model is able to provide the correct numeric value of the blank.

Table 1 compares the performance of four state-of-the-art (SOTA) language models on forward and backward reasoning tasks. We observe a significant drop in backward reasoning accuracy compared to forward reasoning accuracy across all models. The experiments were conducted using the chain of thought prompts defined in Wei et al. (2022). The few-shot examples used in the chain of thought prompts were modified for the backward reasoning task following the procedure described above. It is evident from the results that Large Language Models (LLMs) are not as proficient in backward reasoning compared to forward reasoning, indicating the difficulty of the task as defined above.

4 Proposed Approaches

Rephrasing: Our first base method to tackle the challenging backward reasoning problem involves a problem transformation through rephrasing. This transformation effectively converts the complex backward reasoning task into a more manageable forward reasoning problem. Consequently, we employ the LLM to solve this transformed forward reasoning problem instead of the original and inherently more difficult backward reasoning challenge.

Given a backward MWP (Q, A_f), we ask the language model to produce a rephrased question R, which incorporates the forward answer A_f into the question Q and changes the objective of the question from finding the answer A_f to finding the value of the blank. We then ask the language model to solve the rephrased problem R instead of the original backward problem.

Our experiments in Table 2 show that converting the backward MWP into an algebraic MWP by replacing the blank with x and asking the LLM to find the value of x as shown in Figure 1 gives us better results than converting to a simple MWP which asks us to find the value of the unknown quantity.

PAL-Tools: The second base method that we propose combines Program-aided language models (PAL) (Gao et al., 2023) with the techniques of framing equations and solving them using SymPy (Meurer et al., 2016), (He-Yueya et al., 2023) (referred to hereafter as *tools*). We observe that PAL performs very poorly at the backward reasoning task due to the non-sequential nature of the task. When PAL is used for backward reasoning, LLMs must first rearrange the terms of the equation instead of merely finding a formula to use given the variables in the problem. By only using the framing equations, we present the backward reasoning problem to the LLM in a structured mathematical form, making it easier for them to understand and generate code or solutions based on the given equations.

Recently, Schick et al. (2023) have shown that LLMs can utilize external tools and APIs to solve various downstream tasks. Motivated by this, we leverage the LLM's ability to call SymPy functions to overcome their inability to manipulate equations to isolate the unknown variable. Our prompt frames the problem as an equation in x and asks the model to generate code that solves the equation using Sympy's `solve` method, as shown in Figure 1. Parallel work by Zhou et al. (2023), which uses GPT-4's code interpreter to both solve and verify solutions with code, shows similar improvements on the forward task.

Reprompting and Verification: Our third base method *Reprompting and Verification* is based on recently proposed framework of Self-Refine (Madaan et al., 2023). Self-Refine is an iterative prompting technique that cycles between refinement and feedback until a predetermined condition is attained. We modify the Self-Refine to perform backward reasoning on MWP. We use PAL-tools as the base model within Self-Refine and refine the programs generated by PAL-tools with feedback from the language model as was done in (Madaan et al., 2023).

Table 2: Improvements in accuracy with various prompting strategies

Strategy	Shots	GSM8k_B	SVAMP_B	$\text{MultiArith}_\text{B}$
CoT	8-Shot	10.77	20.40	14.50
PAL	4-Shot	9.27	20.90	18.17
Tools	3-Shot	31.45	43.50	71.83
Rephrase (Linguistic)	8-Shot	19.65	32.60	40.50
Rephrase (Algebraic)	8-Shot	36.12	37.80	71.67
PAL-tools	2-Shot	37.26	41.00	79.67
PAL-tools (Rephrased)	2-Shot	37.89	42.10	77.67
PAL-tools	4-Shot	37.11	42.70	80.50
PAL-tools (Rephrased)	4-Shot	**48.74**	**51.10**	84.50
Self-Refine (Rephrased)	2-Shot	40.17	49.70	77.50
Check your Work (Rephrased)	8-Shot	41.82	47.40	**84.83**

Our method works as follows:

1. We start by instructing the LLM to solve the backward MWP (Q, A_f) and provide the numeric value of blank in question Q.
2. In this step, we verify the answer generated by the LLM. To do this, we substitute the blank in Q with the LLM's answer, creating a modified question, denoted as Q'. We then instruct an LLM to generate the answer A' of this newly formed forward question.
3. We compare the answer A' obtained from Q' with the original answer A_f. If A' matches with A_f our method stops.
4. If, upon comparison, the answers A' and A_f do not match, we go back to step 1 and ask LLM to generate alternate hypotheses by adding the feedback in obtained step 2.

Note that the second step above, which involves verifying the answer, can be performed either by the same or a more powerful LLM. We refer to this method as *Check your work.* An illustration of this process is provided in Figure 1.

5 Ensembling and Verification

In this section, we present a simple approach for ensembling the set of base methods described previously followed by use of verifier to further improve the accuracy. We assume that we are solving the backward reasoning problem, where given a backward question Q, we are interested in finding the correct answer A to fill in the blank in the backward question Q.

5.1 Computing the Prior Distribution

To create an ensemble, we use the idea of frequency voting. Assume that we are given a set of models $\{M_1, M_2, \cdots, M_k\}$. Given a model M_i, we run the model M_i on the question Q r times, to get a multi-set of answers $\{A_{ij}\}_{j=1}^{r}$.

We compute the prior probability $P(A_l \mid Q)$ as simply the fraction of times A_l appears as the answers in the union of multiset of answers produced by each model for question Q : $P(A_l = A \mid Q) = \sum_{i,j} \mathbb{1}[A_{ij} = A]/kr$. This generalizes the idea of majority voting to compute a prior distribution over all the answers produced by a set of models in a given set of runs. This ensemble can be used for (1) Simple Voting: Output the answer with the highest probability (2) Sampling: Output an answer based on its prior probability (3) Prior: Use this as a prior distribution, to compute another posterior with additional knowledge as described below.

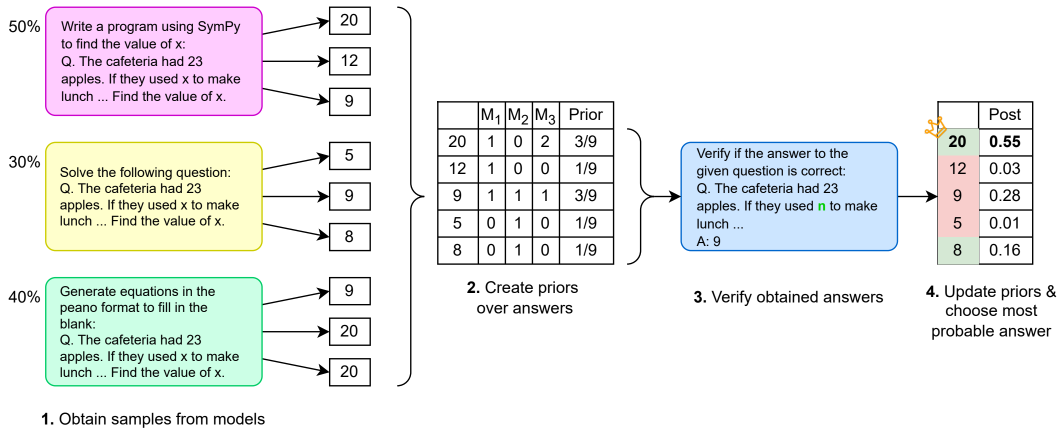

Figure 2: An illustrative example of how the ensembling of base models works together with a verifier.

5.2 Improvements using a Verifier

Next, let us assume we have a verifier V, such that given a question Q and an answer A_l, V gives a Boolean output Z_l. This output is equal to 1 if A_l is the correct answer to the question according to the verifier, and 0 otherwise. Let $P_1(Z_l|A_l\!=\!A,Q)$ denote the distribution over V's outputs when $A_l\!=\!A$. Similarly, let $P(Z_l|A_l\!\neq\!A,Q)$ denote the distribution over V's outputs when $A_l\!\neq\!A$. We estimate these distributions by computing the accuracy of the verifier on the holdout set S', and supplying a set of answers produced by the k models, each run r times on each $Q \in S'$, along with the gold answer A. Please note that we could also use the probability $P_1(Z_l|A_l\!=\!A,Q)$ provided by the verifier, but this probability may not be well calibrated (Jiang et al., 2021; Zhao et al., 2021), i.e., the model's probability estimates may not accurately reflect the true likelihood of the answer being correct. Therefore, we opted to use the holdout set to estimate these probabilities.

The probability that A_l is the correct answer for a question Q, given the prior probability and the verifier output Z_l, by application of Bayes Rule, can be written as

$$P(A_l\!=\!A \mid Z_l, Q) = \frac{P(Z_l \mid A_l\!=\!A, Q)\; P(A_l\!=\!A \mid Q)}{P(Z_l \mid A_l\!=\!A, Q)P(A_l\!=\!A \mid Q) + P(Z_l \mid A_l\!\neq\!A, Q)P(A_l\!\neq\!A \mid Q)} \quad (1)$$

The first term in the numerator captures the prior probability over the verifier output given the correct answer and is computed based on the holdout set as described earlier. The second term is simply the prior probability as computed in section 5.1. The denominator is simply the normalization constant: the first term is the same as in the numerator. In the second term, the first sub-term captures the prior probability of verifier output given an incorrect answer and is obtained from the validation set. The second sub-term can be computed simply by subtracting the prior distribution over answers. Figure 2 illustrates the process of how ensembling works using an example.

6 Experiments

6.1 Setup

In our experimental setup, we work with three primary datasets: GSM8k (Cobbe et al., 2021), MultiArith (Roy & Roth, 2015), and SVAMP (Patel et al., 2021). We transform the examples in these datasets into backward tasks, resulting in the creation of three new datasets: GSM8k_B, SVAMP_B, and $\text{MultiArith}_\text{B}$. We have experimented with four SOTA LLMs: GPT-4, GPT-3.5-Turbo (OpenAI, 2023), PaLM-2 (Anil et al., 2023) and LLaMa-2 (Touvron et al., 2023). The first three models are utilized via their official model APIs, while for LLaMa-2, we use the 70-billion-parameter model quantized to 4-bit using GPTQ (Frantar et al., 2022). Apart from Table 1, all other tables use GPT-3.5-Turbo as the base model, unless mentioned otherwise.

Table 3: Results of Ensembling

	$GSM8k_B^\dagger$	$SVAMP_B^\dagger$	$MultiArith_B^\dagger$
CoT	35.67	37.78	69.60
Tools	41.81	48.11	72.00
PAL-Tools	48.55	45.00	81.50
Ensemble	**65.33**	**66.67**	**92.60**

We work with 4 different prompting techniques: Chain of Thought (Wei et al., 2022), PAL (Gao et al., 2023), Tools (He-Yueya et al., 2023) and SELF-REFINE (Madaan et al., 2023). Prompts and in-context examples for the prompting techniques are taken from their original works. The in-context examples are modified for the backward setting as discussed in Section 4. Examples of the prompts used are given in Appendix A.2.

6.2 APPROACHES

We test our approaches using GPT-3.5-Turbo as the base model. The results are highlighted in Table 2. We classify the prompts into four categories:

1. **Baseline prompts**: Prompts whose in-context examples were adapted to the backward task without changing the technique.
2. **Rephrased prompts**: Prompts that solve the rephrased question using chain-of-thought reasoning. Used in proposed base method rephrasing.
3. **PAL-Tools prompts**: Prompts that use PAL-tools and their variants.
4. **Adaptive prompts**: Prompts used for resampling another hypothesis from the LLM if the verification condition is not satisfied in the base method Reprompting and Verification.

For rephrasing, we find that models perform better when the rephrased problem has the blank replaced with x compared to the baseline prompt This is because the relationship between the missing value and the equations that models need to frame in order to solve the forward problem is explicit. Also, for the baseline prompt, the model requires inferring the relationship between the forward answer and the equations they need to frame to obtain it, which may introduce ambiguity and reduce accuracy.

For programmatic techniques, we see that PAL-Tools is more performant than Pal or Tools taken separately. We hypothesize that LLMs encounter a larger amount of code in their training data compared to the Peano solutions that He-Yueya et al. (2023) use in their prompt, and this implies that they may find it easier to frame equations in a programming language rather than in a domain-specific language specified via few-shot examples. We find that 4-shot PAL-tools with rephrasing performs the best on $GSM8k_B$ and $SVAMP_B$, while Check your Work prompting marginally outperforms PAL-tools on $MultiArith_B$. We also see that more in-context examples strongly assist in rephrasing, as 4-shot rephrased PAL-Tools strongly outperforms 2-shot rephrased PAL-Tools when compared to the base version without any rephrasing.

Finally, our proposed Check your Work method does marginally better than with our designed prompts than prompts given the original Self-Refine paper (Madaan et al., 2023), with the added benefit of much lower cost. While Self-Refine's feedback prompts are quite large, containing extensive feedback, check your work only appends the final question and chain of thought for verifying the final question to the few-shot examples. This reduces cost as well as inference time. We also see that Self-Refine outperforms the comparable PAL-tools variant, indicating that feedback from the language model does assist in correcting reasoning.

6.3 ENSEMBLING

For our ensemble experiments, we selected the 100 examples from the datasets as holdout sets to compute the prior probabilities of the models and the verifier's accuracy. The models we included in the ensemble are rephrased versions of three of our strongest single-prompt models: Chain-of-Thought, Tools, and PAL-Tools. We then evaluated the ensemble on the non-holdout set, which is

Table 4: Improvements using Rephrasing

Model	Shots	Rephrased	$GSM8k_B$	$SVAMP_B$	$MultiArith_B$
CoT	8-shot	No	10.77	20.4	14.50
		Yes	**36.12**	**37.8**	**71.67**
			(↑25.35)	(↑17.4)	(↑57.17)
PAL	4-shot	No	9.27	20.9	18.17
		Yes	**21.38**	**37.0**	**55.50**
			(↑12.11)	(↑16.1)	(↑37.33)
Tools	3-shot	No	31.45	43.5	71.83
		Yes	**41.43**	**48.5**	**73.00**
			(↑9.98)	(↑5.0)	(↑1.17)
PAL-tools	4-shot	No	37.11	42.7	80.50
		Yes	**48.74**	**51.1**	**84.50**
			(↑11.63)	(↑8.4)	(↑4)

	model	
actual	positive	negative
positive	75.94	24.05
negative	7.39	92.61

Table 5: Confusion matrix for verifying problems and their solutions on $GSM8k_B$, normalized across rows

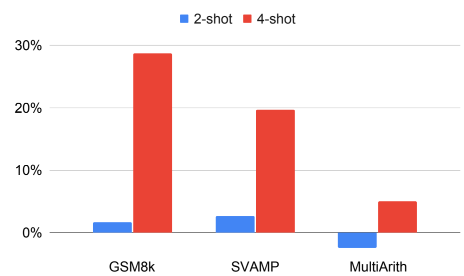

Figure 3: Relative performance increase rephrasing brings to PAL-tools when 4 shots are used compared to 2.

denoted with a † symbol in Table 3 to show the results. We observed that the accuracy on backward MWP via ensembling surpasses the forward accuracy of Chain-of-Thought by up to 6%.

7 Analysis

How much does rephrasing help? Since rephrasing is a strategy that can be applied across multiple techniques, we analyze the extent of accuracy gains obtained via rephrasing by applying it independently to four techniques: CoT, PAL, Tools, and PAL-Tools. The results are shown in Table 4. Rephrasing improves the accuracy of every technique that it is applied to. We see larger gains with rephrasing in weaker methods, such as CoT. We also see that rephrasing has higher gains in datasets where the problems are harder, such as in GSM8k compared to SVAMP.

Is verifying easier than solving? In the third step of the ensembling method, we try to verify whether the blank provided is correct by solving the resulting forward problem after substituting the blank. There are two settings in which we can verify this: 1) We ask the model to solve this new question, and compare whether the answer obtained is same as original answer a. 2) We give the original answer a to the model and ask it to check whether it is the answer obtained for new question.

To find which method is better at correctly verifying the blank, we check the accuracy of GPT-3.5-turbo on GSM8k in setting 2. In the first pass, we provide the correct blank and in the second pass, we provide an incorrect blank formed by multiplying $z \in \{2 \dots 10\}$ with the correct blank. The confusion matrix obtained is shown in Figure 5. It is observed that the accuracy of setting 2 is higher than the forward reasoning accuracy of GPT-3.5-turbo. Hence we use that as the verification method for ensembling.

Carla just gave birth to identical octuplets. She dresses 3/4 of them in purple and _____ in blue. If all the blue wearers and 1/3 of the purple wearers also wear bows, what is the percentage chance a baby wearing a bow is wearing purple?
Ian has a board that is 40 feet long. He decides to make a cut so he can have _____ pieces. The longer piece is 4 times longer than the shorter piece. How long is the longer piece?

Table 7: Examples of questions in GSM8k_B that don't require the answer to find the value of the blank

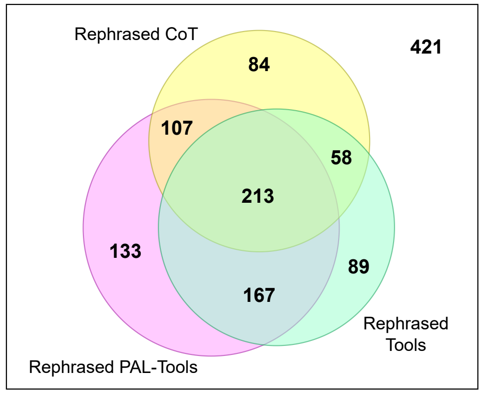

Figure 4: Venn diagram of the subsets of problems of GSM8k_B different prompts are able to solve

Does the verifier assist in ensembling? We compare the accuracies obtained by using majority voting with and without the verifier. We find that using a verifier improves the accuracy on $\text{GSM8k}_\text{B}^\dagger$ and $\text{SVAMP}_\text{B}^\dagger$ by 7% and on $\text{MultiArith}_\text{B}^\dagger$ by 0.6%. Since the verifier has a higher accuracy than any of the models we consider, its inclusion inevitably increases the accuracy of any set of methods we choose. Even if we use a noisy verifier, updating our priors based on its results using Bayes' rule ensures that the priors are not changed significantly.

Do some prompts subsume others? Let prompts M_1, M_2, M_3 be able to solve problems $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$ respectively, where $\mathcal{D}_i \subseteq D$. If we choose to ensemble these prompts together, then $|\bigcup_j \mathcal{D}_j| > |\mathcal{D}_i|$ for the ensemble to do better. Figure 4 gives an overview of the subsets of GSM8k three prompts, namely Rephrased Chain of Thought, Rephrased PAL-Tools and Rephrased Tools can solve in a single try. We see that even though there is significant overlap between the prompts, the probability of any one of them giving the right answer is 66.9 %, provided we sample from each prompt once. The venn diagram also shows that the subsets of problems different prompts cover is quite disjoint in nature. No prompt can solve all the problems that another prompt can solve.

Can every blank's answer be determined? There may be cases where the blank does not directly contribute to the answer, or is irrelevant. In such a case, inferring the value of the blank is not possible given the answer. Even though Cobbe et al. (2021) claim that less than two percent of problems have breaking errors, We sample 50 random examples from GSM8k_Bthat our strongest model solves incorrectly and find that no such problems in the sample we analyse, leading us to believe that the probability of such problems existing in our dataset is little to none.

Do all blanks need answers to be solved? In the 50 examples we analyse above, there are 10 examples where the value of the blank can be obtained simply from reading the question, as the question makes implicit assumptions or provides further information that can be used to fill in the blank. Two examples are presented in figure 3. It is surprising that even our strongest model is unable to find the answer to such questions, either as a consequence of its poor reasoning abilities or because we make the dependency between requiring the answer to fill in the blank explicit.

8 Conclusion

We show that backward reasoning is still a hard problem for large language models, by showing that their accuracy drops significantly on this task compared to forward reasoning. To augment this, we propose multiple methods that can improve their accuracy in this setting and are extensible across preexisting models. We also propose a Bayesian ensembling technique that can combine multiple models in the presence of a noisy verifier. These techniques combined bring backward performance on par with forward reasoning performance. Finally, we analyze the fallacies and pitfalls of each of these techniques and show areas for future improvements.

REFERENCES

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Chandra Bhagavatula, Ronan Le Bras, Chaitanya Malaviya, Keisuke Sakaguchi, Ari Holtzman, Hannah Rashkin, Doug Downey, Wen tau Yih, and Yejin Choi. Abductive commonsense reasoning. In *ICLR*, 2020.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

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. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Optq: Accurate quantization for generative pre-trained transformers. In *ICLR*, 2022.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *ICML*, 2023.

Joy He-Yueya, Gabriel Poesia, Rose E. Wang, and Noah D. Goodman. Solving math word problems by combining language models with symbolic solvers. *arXiv preprint arXiv:2304.09102*, 2023.

Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. How can we know when language models know? on the calibration of language models for question answering. *Transactions of the Association for Computational Linguistics*, 9:962–977, 2021.

Zhanming Jie, Jierui Li, and Wei Lu. Learning to reason deductively: Math word problem solving as complex relation extraction. 2022.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597, 2015.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. Learning to automatically solve algebra word problems. In *ACL*, pp. 271–281, 2014.

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. In *Findings EMNLP*, pp. 2841–2852, 2020.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *ACL*, pp. 158–167, 2017.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. A survey of deep learning for mathematical reasoning. *arXiv preprint arXiv:2212.10535*, 2022.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E Granger, Richard P Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J Curry, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: Symbolic computing in python. May 2016.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *NAACL*, 2021.

Lianhui Qin, Vered Shwartz, Peter West, Chandra Bhagavatula, Jena D. Hwang, Ronan Le Bras, Antoine Bosselut, and Yejin Choi. Back to the future: Unsupervised backprop-based decoding for counterfactual and abductive commonsense reasoning. In *EMNLP*, 2020.

Lianhui Qin, Sean Welleck, Daniel Khashabi, and Yejin Choi. COLD decoding: Energy-based constrained text generation with langevin dynamics. In *NeurIPS*, 2022.

Ajay Ramful and John Olive. Reversibility of thought: An instance in multiplicative tasks. *The Journal of Mathematical Behavior*, 27(2):138–151, 2008.

FD Rivera. On the pitfalls of abduction: Compolicities and complexities in patterning activity. *For the learning of mathematics*, 28(1):17–25, 2008.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *EMNLP*, 2015.

Subhro Roy and Dan Roth. Mapping to declarative knowledge for word problem solving. *Transactions of the Association for Computational Linguistics*, 6:159–172, 2018.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Jianhao Shen, Yichun Yin, Lin Li, Lifeng Shang, Xin Jiang, Ming Zhang, and Qun Liu. Generate & rank: A multi-task framework for math word problems. In *Findings of EMNLP*, pp. 2269–2279, 2021.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NeurIPS*, 2014.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, 2023.

Yan Wang, Xiaojiang Liu, and Shuming Shi. Deep neural solver for math word problems. In *EMNLP*, pp. 845–854, 2017.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. In *ICLR*, 2023.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. *arXiv preprint arXiv:2212.09561*, 2022.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *ICML*, pp. 12697–12706, 2021.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv:2304.09797*, 2023.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. *arXiv preprint arXiv:2308.07921*, 2023.

A Appendix

A.1 Dataset

We consider three datasets of interest: GSM8K (Cobbe et al., 2021), MultiArith (Roy & Roth, 2015) and SVAMP (Patel et al., 2021). All these datasets consist of grade-school arithmetic word problems along with their answers.

A.1.1 Generation Methodology

Given a source forward dataset

$$D = \{(Q_i, A_i)_{i=1}^{n} \mid Q_i \in \Sigma^*, A_i \in \mathbb{R}\}$$

we present a method to create a backward dataset

$$D'_k = \{(Q'_i, A_i, (B_i^0, \dots, B_i^k))_{i=1}^{n} \mid Q'_i \in \Sigma^*, A_i, B_i^j \in \mathbb{R}\}$$

To convert Q_i (Source question) to Q'_i (blanked out question) and extract blanks $B_i^0 \dots B_i^k$, we split Q_i into it's constitutent tokens based on a delimiter, usually space. We then consider all numeric tokens, which are defined as tokens that encode a number. Numeric tokens may be alphanumeric, such as \$42, 80% or 3.14, or they may be alphabetic, such as three, twice or half. Using this heuristic for numeric tokens, we ignore the first numeric token and extract the next k tokens sequentially. If we are unable to extract k tokens, then we skip that question and answer pair. It is worth noting that for the datasets we use, $k = 1$, that is we only consider the problem of backwardly inferring one missing number in the question, given the answer. Solving the $n > 1$ case would require first checking if a unique solution exists and is a topic for future work.

Dataset	*N*	Example
GSM8k	1272*	Kylar went to the store to buy glasses for his new apartment. One glass costs $5, but every second glass costs only 60% of the price. Kylar wants to buy 16 glasses. How much does he need to pay for them?
GSM8k$_B$	1272	Q : Kylar went to the store to buy glasses for his new apartment. One glass costs $5, but every second glass costs only _____% of the price. Kylar wants to buy 16 glasses. How much does he need to pay for them? A : 64
SVAMP	1000	28 children were riding on the bus. At the bus stop 82 children got on the bus while some got off the bus. Then there were 30 children altogether on the bus. How many more children got on the bus than those that got off?
SVAMP$_B$	1000	Q : 28 children were riding on the bus. At the bus stop _____ children got on the bus while some got off the bus. Then there were 30 children altogether on the bus. How many more children got on the bus than those that got off?" A : 2
MultiArith	600	Lana picked 36 tulips and 37 roses to make flower bouquets. If she only used 70 of the flowers though, how many extra flowers did Lana pick?
MultiArith$_B$	600	Q : Lana picked 36 tulips and _____ roses to make flower bouquets. If she only used 70 of the flowers though, how many extra flowers did Lana pick? A : 3

Table 7: Sample questions from the datasets we consider

The reason we choose to blank out only numeric tokens rather than an entire phrase or sentence is to make the task of validation easier. An alternative that was explored was phrase masking. However, phrase masking would lead to generations that would not be verifiable with perfect accuracy, and multiple possible generations for each question. The benefit of number masking is that quantities can be compared to each other without loss of accuracy, and every question-answer pair has a unique blank.

A.1.2 Generation Results

Using the above method, we were able to convert 1272 of the 1319 question and answer pairs in GSM8k to backward reasoning problems, and all 1000 and 600 pairs in SVAMP and MultiArith respectively.

A.1.3 Dataset Examples

Some examples of the datasets under consideration are shown in Table 7. Note that for GSM8k, the original dataset contains 1319 sample problems but our dataset generation method for the backward task filters out 47 of them. For comparability with the backward task, we have used the 1272 common examples of this dataset. We use all the problems in SVAMP and MultiArith.

A.2 Prompts

We construct prompts by changing the original examples of the papers we consider. We show one to two in-context examples of each prompt. The remaining examples may be seen in our code.

```
Rephrase the given blanked question and answer pairs and then find the
solution to the rephrased question. Give your answer as either a number
or a decimal (no fractions). Follow the format specified in the examples
below:

Q: There are 15 trees in the grove. Grove workers will plant _____ trees
in the grove today. After they are done, how many trees would be there?
A: 21
Rephrased: There are 15 trees in the grove. Grove workers will plant x
trees in the grove today. After they are done, there would be 21 trees.
Find the value of x.
Answer: There are 15 trees originally, Then there were 21 trees after
some more were planted. So there must have been x = 21 - 15 = 6 trees.
The answer is 6.

Q: If there are 3 cars in the parking lot and _____ more cars arrive,
how many cars are in the parking lot?
A: 5
Rephrased: If there are 3 cars in the parking lot and x more cars
arrive, there are 5 cars in the parking lot. Find the value of x.
Answer: There are originally 3 cars. x more cars arrive. 3 + x = 5, so x
= 5 - 3 = 2. The answer is 2.

...

Q: {{question}}
A: {{answer}}
Rephrased:
```

Figure 5: Rephrasing with x

```
You are given a math question with a blank value and an answer. Solve it step b step to find the value of blank. Strictly follow the format given in the examples below.

Question: Ben has four boxes with ten basketball cards in each box. Ben received ______ cards from his mother. If he gives 58 cards to his classmates, how many cards does he has left?
Answer: 22

Peano solution:


Let a be number of boxes [[var a]]. We have [[eq a = 4]].
Let b be number of cards in each box [[var b]]. We have [[eq b = 10]].
Let c be number of cards Ben initially has [[var c]]. We have [[eq c = a * b]].
Let d be cards received from mother [[var d]].
Let e be cards given to classmates [[var e]]. We have [[eq e = 58]].
Let f be cards left [[var f]]. From given Answer, we have [[eq f = 22]].
We have [[eq d = f + e - c]]
The answer is the value of d [[answer d]].




Question: Natalia sold _____ clips to her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?
Answer: 72

Peano solution:


Let a be number of clips Natalia sold in April [[var a]].
So number of clips Natalia sold in May are half of a.
Let b be number of clips sold altogether [[var b]]. From given Answer, we have [[eq b = 72]].
We have [[eq a = b / (1 + 1/2)]]
The answer is the value of a [[answer a]].

...

Q: {{question}}
A: {{answer}}

Peano solution:
```

Figure 6: Tools

```
Rephrase the given blanked question and answer pairs and then solve it step b step to find the value of blank. Strictly follow the format given in the examples below.

Question: Ben has four boxes with ten basketball cards in each box. Ben received ______ cards from his mother. If he gives 58 cards to his classmates, how many cards does he has left?
Answer: 22

Rephrased: Ben has four boxes with ten basketball cards in each box. Ben received x cards from his mother. If he gives 58 cards to his classmates, he has 22 cards left. Find the value of x.
Peano solution:


Let a be number of boxes [[var a]]. We have [[eq a = 4]].
Let b be number of cards in each box [[var b]]. We have [[eq b = 10]].
Let c be number of cards Ben initially has [[var c]]. We have [[eq c = a * b]].
Let x be cards received from mother [[var x]].
Let d be total cards with Ben [[var d]]. We have [[eq d = c + x]]
Let e be cards given to classmates [[var e]]. We have [[eq e = 58]].
Let f be cards left [[var f]]. From given Answer, we have [[eq f = 22]].
As cards left are also total cards - cards given to classmates, we have [[eq f = d - e]]
The answer will be the value of x [[answer x]].





Question: Natalia sold _____ clips to her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?
Answer: 72

Rephrased: Natalia sold clips to x of her friends in April, and then she sold half as many clips in May. Find the value of x such that she sold a total of 72 clips altogether in April and May.
Peano solution:


Let x be number of clips Natalia sold in April [[var x]]
Let a be number of clips Natalia sold in May [[var a]]. We have [[eq a = x / 2]].
Let b be number of clips sold altogether [[var b]]. From given Answer, we have [[eq b = 72]].
As clips sold altogether are also the sum of clips sold in April and May, we have [[eq b = x + a ]]
The answer will be the value of x [[answer x]].

...

Q: {{question}}
A: {{answer}}

Rephrased:
```

Figure 7: Tools with Rephrasing

````
Rephrase the given blanked question and answer pairs and then find the solution to the rephrased question. Write a python function that finds the value of x by solving step by step. Make sure you name your method finding_x. A few examples are given below.:

Question: Ben has four boxes with ten basketball cards in each box. Ben received ______ cards from his mother. If he gives 58 cards to his classmates, how many cards does he has left?
Answer: 22
Rephrased: Ben has four boxes with ten basketball cards in each box. Ben received x cards from his mother. He gives 58 cards to his classmates. He has 22 cards left.
Program:
```python
def finding_x():
    num_boxes = 4
    cards_per_box = 10
    # cards_received_from_mother = x - This line is commented because x is unknown
    # hence the variable cards_received_from_mother can't be used in R.H.S. of any calculation
    cards_given_to_classmates = 58
    cards_left = 22
    cards_in_boxes = num_boxes * cards_per_box
    total_cards_before_given_to_classmates = cards_given_to_classmates + cards_left

    cards_received_from_mother = total_cards_before_given_to_classmates - cards_in_boxes
    return cards_received_from_mother
```

Question: Olivia has $23. She bought _____ bagels for $3 each. How much money does she have left?
Answer: 8
Rephrased: Olivia has $23. She bought x bagels for $3 each. She has $8 left. Find the value of x.
Program:
```python
def finding_x():
    money_initial = 23
    # num_of_bagels = x - This line is commented because x is unknown
    # hence the variable num_of_bagels can't be used in R.H.S. of any calculation
    bagel_cost = 3
    money_left = 8
    money_spent = money_initial - money_left

    num_of_bagels = money_spent / bagel_cost
    return num_of_bagels
```

Question: {{question}}
Answer: {{answer}}
Rephrased:
````

Figure 8: PAL with Rephrasing

````
You are given a math question with a blank value and an answer. Write a python function called solution() using sympy that assumes the value of the blank is x and creates an equation in x that is solved by sympy.solve. Return the value of the blank. You may assume the neccessary libraries are imported. Strictly follow the format given in the examples below, as the method will be executed with the same name.

Q: Ben has four boxes with ten basketball cards in each box. Ben received _____ cards from his mother. If he gives 58 cards to his classmates, how many cards does he has left?
A: 22
Program:
```python
def solution():
    num_boxes = 4
    cards_per_box = 10
    total_cards_in_boxes = num_boxes * cards_per_box
    cards_from_mother = x
    cards_given_to_classmates = 58
    cards_left = 22

    equation = Eq(cards_from_mother + total_cards_in_boxes, cards_given_to_classmates + cards_left)
    blank = solve(equation)[0]

    return blank
```

Q: Natalia sold clips to _____ of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?
A: 72
Program:
```python
def solution():
    april_clips = x
    may_clips = april_clips / 2
    total_clips = 72

    equation = Eq(april_clips + may_clips, total_clips)
    blank = solve(equation)[0]

    return blank
```
...

Q: {{question}}
A: {{answer}}
Program:
````

Figure 9: PAL-Tools

````
Rephrase the given blanked question and answer pairs and then write a python function called solution() to find the value of x in the rephrased question. Return the value of x. You may assume the neccessary libraries are imported. Strictly follow the format given in the examples below, as the method will be executed with the same name.

Q: Ben has four boxes with ten basketball cards in each box. Ben received _____ cards from his mother. If he gives 58 cards to his classmates, how many cards does he has left?
A: 22
Rephrased: Ben has four boxes with ten basketball cards in each box. Ben received x cards from his mother. He gives 58 cards to his classmates. He has 22 cards left. Find the value of x.
Program:
```python
def solution():
    num_boxes = 4
    cards_per_box = 10
    total_cards_in_boxes = num_boxes * cards_per_box
    cards_from_mother = x
    cards_given_to_classmates = 58
    cards_left = 22

    equation = Eq(cards_from_mother + total_cards_in_boxes, cards_given_to_classmates + cards_left)
    blank = solve(equation)[0]

    return blank
```

Q: Natalia sold _____ clips to  her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?
A: 72
Rephrased: Natalia sold x clips to her friends in April, and then she sold half as many clips in May. Natalia sells 72 clips altogether in April and May. Find the value of x.
Program:
```python
def solution():
    april_clips = x
    may_clips = april_clips / 2
    total_clips = 72

    equation = Eq(april_clips + may_clips, total_clips)
    blank = solve(equation)[0]

    return blank
```

...

Q: {{question}}
A: {{answer}}
Rephrased:
````

Figure 10: PAL-Tools with Rephrasing

```
Fill in the blank given the question and answer examples below. Give your answer as either a number or a decimal (no fractions). Check your work by substituting your answer in the blank, solving the question and comparing to the original answer. Follow the format specified in the examples below:

Q: There are 15 trees in the grove. Grove workers will plant _____ trees in the grove today. After they are done, how many trees would be there?
A: 21
Answer: There are 15 trees originally, Then there were 21 trees after some more were planted. So there must have been 21 - 15 = 6 trees. The blank is 6.
Final question: There are 15 trees in the grove. Grove workers will plant 6 trees in the grove today. After they are done, how many trees would be there?
Check: There would be 15 + 6 = 21 trees in total. The original answer was 21. This matches the original answer.

Q: If there are 3 cars in the parking lot and _____ more cars arrive, how many cars are in the parking lot?
A: 5
Answer: There are originally 3 cars. There are finally 5 cars, so 5 - 3 = 2 cars arrived. The blank is 2.
Final question: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?
Check: There would be 3 + 2 = 5 cars in the parking lot. The original answer was 5. This matches the original answer.

...

Q: {{question}}
A: {{answer}}
Answer:
```

Figure 11: Check your work

```
Fill in the blank given the question and answer examples below. Give your answer as either a number or a decimal (no fractions). Check your work by substituting your answer in the blank, solving the question and comparing to the original answer. Follow the format specified in the examples below:

Q: There are 15 trees in the grove. Grove workers will plant _____ trees in the grove today. After they are done, how many trees would be there?
A: 21
Rephrased: There are 15 trees in the grove. Grove workers will plant x trees in the grove today. After they are done, there would be 21 trees. Find the value of x.
Answer: There are 15 trees originally, Then there were 21 trees after some more were planted. So there must have been x = 21 - 15 = 6 trees. The answer is 6.
Final question: There are 15 trees in the grove. Grove workers will plant 6 trees in the grove today. After they are done, how many trees would be there?
Check: There would be 15 + 6 = 21 trees in total. This matches the original answer.

Q: If there are 3 cars in the parking lot and _____ more cars arrive, how many cars are in the parking lot?
A: 5
Rephrased: If there are 3 cars in the parking lot and x more cars arrive, there are 5 cars in the parking lot. Find the value of x.
Answer: There are originally 3 cars. x more cars arrive. 3 + x = 5, so x = 5 - 3 = 2. The answer is 2.
Final question: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?
Check: There would be 3 + 2 = 5 cars in the parking lot. This matches the original answer.

...

Q: {{question}}
A: {{answer}}
Rephrased:
```

Figure 12: Check your work with Rephrasing

````
You are given a math question with a blank value and an answer. Rephrase the given blanked question and answer pairs and then write a python function called solution() to find the value of x in the rephrased question. Return the value of x. You may assume the neccessary libraries are imported. Strictly follow the format given in the examples below, as the method will be executed with the same name.

Q: Ben has four boxes with ten basketball cards in each box. Ben received _____ cards from his mother. If he gives 58 cards to his classmates, how many cards does he has left?
A: 22
Rephrased: Ben has four boxes with ten basketball cards in each box. Ben received x cards from his mother. He gives 58 cards to his classmates. He has 22 cards left. Find the value of x.
Program:
```python
def solution():
    num_boxes = 4
    cards_per_box = 10
    total_cards_in_boxes = num_boxes * cards_per_box
    cards_from_mother = x
    cards_given_to_classmates = 58
    cards_left = 22

    equation = Eq(cards_from_mother + total_cards_in_boxes, cards_given_to_classmates + cards_left)
    blank = solve(equation)[0]

    return blank
```


Q: Natalia sold _____ clips to  her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?
A: 72
Rephrased: Natalia sold x clips to her friends in April, and then she sold half as many clips in May. Natalia sells 72 clips altogether in April and May. Find the value of x.
Program:
```python
def solution():
    april_clips = x
    may_clips = april_clips / 2
    total_clips = 72

    equation = Eq(april_clips + may_clips, total_clips)
    blank = solve(equation)[0]

    return blank
```

Q: {{question}}
A: {{answer}}
Rephrased:
````

Figure 13: PAL-Tools with Rephrasing and Self-Refine: init prompt

```
You are given a question-answer pair with a blank, and a chain of thought (CoT) for filling in the blank. Go through the chain of thought step by step and point out mistakes, if any. Provide the final corrected answer as shown below.

Q: Kelly is grocery shopping at a supermarket and is making sure she has enough in her budget for the items in her cart. Her 5 packs of bacon cost $_____ in total and she has 6 packets of chicken which each cost twice as much as a pack of bacon. She also has 3 packs of strawberries, priced at $4 each, and 7 packs of apples, each priced at half the price of a pack of strawberries. If Kelly's budget is $65 then how much money, in dollars, does she have left in her budget?
A: 5
Rephrased: Kelly is grocery shopping at a supermarket and is making sure she has enough in her budget for the items in her cart. Her 5 packs of bacon cost a total of x dollars, and she has 6 packets of chicken, each costing twice as much as a pack of bacon. She also has 3 packs of strawberries priced at $4 each, and 7 packs of apples, each priced at half the price of a pack of strawberries. If Kelly's budget is $65, then how much money, in dollars, does she have left in her budget?

Program:

```python
def solution():
    bacon_cost = x
    chicken_cost = 2 * bacon_cost
    strawberry_cost = 4
    apple_cost = strawberry_cost / 2
    budget = 65
    remaining_budget = budget - (5 * bacon_cost + 6 * chicken_cost + 3 * strawberry_cost + 7 * apple_cost)

    equation = Eq(remaining_budget, 5)
    blank = solve(equation)[0]

    return blank
```

Correction:

The rephrased problem does not end with 'Find the value of x'. Instead, it asks us to find the money Kelly has left in her budget. This is incorrect.

Let's check the program:

    bacon_cost = x
    chicken_cost = 2 * bacon_cost

Looks good

    strawberry_cost = 4
    apple_cost = strawberry_cost / 2

Looks good

    budget = 65
    remaining_budget = budget - (5 * bacon_cost + 6 * chicken_cost + 3 * strawberry_cost + 7 * apple_cost)

The equation formed is correct, as are the number of packs of bacon, chicken, strawberry and apples

    equation = Eq(remaining_budget, 5)
    blank = solve(equation)[0]
    return blank

The equation has been equated to the correct answer, and the program returns the blank.
```

````
Final Rephrased Problem:
Kelly is grocery shopping at a supermarket and is making sure she has enough in her budget for the items in her cart. Her 5 packs of bacon cost a total of x dollars, and she has 6 packets of chicken, each costing twice as much as a pack of bacon. She also has 3 packs of strawberries priced at $4 each, and 7 packs of apples, each priced at half the price of a pack of strawberries. If Kelly's budget is $65, then she has $5 left in her budget after shopping. Find the value of x.

Final Program:
```python
def solution():
    bacon_cost = x
    chicken_cost = 2 * bacon_cost
    strawberry_cost = 4
    apple_cost = strawberry_cost / 2
    budget = 65
    remaining_budget = budget - (5 * bacon_cost + 6 * chicken_cost + 3 * strawberry_cost + 7 * apple_cost)

    equation = Eq(remaining_budget, 5)
    blank = solve(equation)[0]

    return blank
```
...

Q: {{question}}
A: {{answer}}
Rephrased: {{rephrased}}
Program:
```python
{{program}}
```

Correction:
````

Figure 15: PAL-Tools with Rephrasing and Self-Refine: feedback prompt continued