Interpretable Math Word Problem Solution Generation via Step-by-step Planning

Anonymous ACL submission

Abstract

A wide range of approaches exists for automatically solving math word problems (MWPs), with the majority focusing on obtaining the final correct final answer, which is not enough. Solutions with step-by-step explanations are valuable in many applications, especially in education, to help students better comprehend problem solving strategies. Recent approaches built on large-scale, pre-trained language models, offer a possibility not only to reach a single final answer but also to generate intermediate solution steps, leveraging the language understanding and generation capabilities of language models. However, language models lack mathematical reasoning ability and often cannot generate coherent steps with a clear solution strategy. Thus, we study the problem of fine-grained, step-by-step controllable solution generation for MWPs. We explore whether we can learn a solution strategy that informs future steps and apply controllable generation methods to generate step-by-step not word-by-word. We (i) train a math operation predictor to plan the mathematical operation to apply in the next step given history steps and (ii) use the predicted operation to prompt a language model to generate the next step token-by-token. We conduct numerical experiments on the GSM8K dataset and show that our method improves the overall MWP solving accuracy and solution interpretability with step-by-step plans.

1 Introduction

Arithmetic math word problems (MWPs) consist of natural language statements describing real-world scenarios that involve numerical quantities, followed by a question text asking for an unknown value. Solving MWPs require parsing the textual statements and carrying out the corresponding calculations (Kumar et al., 2022). MWPs are an important educational tool that helps assess and improve student knowledge in basic mathematical concepts and skills (Walkington, 2013; Verschaffel et al., 2020). They also represent a long-standing interest in artificial intelligence (AI) research since correctly solving them serves as a key benchmark task for testing and improving the mathematical reasoning skills of AI models (Feigenbaum and Feldman, 1995; Bommasani et al., 2021; Cobbe et al., 2021; Lewkowycz et al., 2022).

There is a large body of literature that focuses on automatically solving MWP. Earlier works took a modular approach that first analyzes unconstrained natural language and then maps intricate text patterns onto mathematical vocabulary (Sundaram et al., 2022). As a result, this approach relies heavily on hand-crafted rules to fill the gap between natural language and symbolic mathematical vocabulary (Sundaram et al., 2022). Recent works leverage advances in natural language processing and take a neural network-based, end-to-end approach, where a neural network encodes a numerical representation of MWP (and the underlying equation), from which a decoder generates the final answer (Zou and Lu, 2019; Wang et al., 2017; Wu et al., 2020; Chen et al., 2020). Unfortunately, the vast majority of these works focus on generating and predicting a single final answer, without providing any insights or explanations into how the models arrive at the answer. This is because final answer correctness has been, for the most part, the only metric for evaluating the effectiveness of different MWP solving approaches. As a result, it is often difficult, if not entirely impossible, to explain the model's behavior, especially when it produces a wrong answer. The lack of interpretability of these methods makes it challenging to analyze them and unsafe to use them in real-world applications.

This interpretability issue has attracted increasing interest in MWP solving. Recent works have shifted to designing models that not only generate the final answer for an MWP, *but also the intermediate steps*. The ability to generate intermediate steps not only enables researchers to investigate the behaviors of the model but also new applications. For example, in personalized education and intelligent tutoring systems, these models have the potential to generate detailed, personalized solution steps as feedback to improve student understanding of the mathematical concepts and resolve misconceptions (Walkington, 2013; Karpicke, 2012; Koedinger et al., 2015). The recent GSM8K (Cobbe et al., 2021) dataset contains MWPs that come with 2 to 8 intermediate steps described in natural language, which provides us a good resource to study step-by-step solution generation. Many works apply language models (LMs) on this dataset and achieve high accuracy in final answer generation, without studying the quality of intermediate steps (Wei et al., 2022; Wang et al., 2022; Chowdhery, Aakanksha and others, 2022; Lewkowycz et al., 2022). These works use verifiers, self-consistency decoding strategy (majority votes), chain of thought prompting, or calculators; we provide a detailed discussion in the Related Work section.

Despite LMs being capable of generating intermediate steps, these works still mainly focus on obtaining the final correct answer. They view intermediate steps only as a way to provide models additional context of MWP solving to improve the correctness of final answers. As a result, these works are prone to generate incorrect intermediate steps despite achieving the correct final answer, which indicates that these models are not really competent at numerical reasoning. A potential reason is that LMs restrict these approaches to generate intermediate steps in a word-by-word (or token-by-token) way. As a result, these approaches can only utilize shallow heuristics (Li et al., 2021) in word occurrence and lack an understanding of the overall structure of multi-step solution that solving an MWP requires.

1.1 Contributions

In this paper, we study the problem of generating accurate and high-quality intermediate solution steps with natural language explanation via step-by-step planning using a large LM. We formulate this problem as a controllable generation problem where the LM aims to generate the correct intermediate solution at each solution step, given the MWP and previous solution steps. This problem is particularly challenging since *the generated solution steps need to be accurate*, i.e., each intermediate step must be mathematically valid and on the path to the correct answer. Thus, we need an approach that is different from commonly-studied, attribute-controlled generation approaches for topic or sentiment control, where the attribute is more nuanced and cannot be matched exactly (Dathathri et al., 2020; Krause et al., 2020; Shirish Keskar et al., 2019).

To overcome these challenges, we introduce a *step-by-step planning* approach, where we plan the strategy for the next solution step and use the plan to guide LMs to generate the step. In particular, we propose to use specifically formulated *mathematical hints* in the form of mathematical operations to prompt the model to generate a particular intermediate step.

We summarize our contributions as follows.

- We explore the use of a planning approach for step-by-step solution generation in MWPs. To the best of our knowledge, this is the first work that focuses on generating high-quality intermediate solution steps via LMs.
- We first predict the mathematical operation applied in the next solution step using a small model and then apply a carefully-constructed prompt to control a large LM to generate the next solution step. Our approach can be extended to many downstream applications due to the interpretability and high controllability of the mathematical operation prompt.
- We verify the effectiveness of our planning-LM approach both quantitatively and qualitatively on the GSM8K dataset. We show that, with minimal additional parameters (0.02%) introduced, our planning-LM approach yields both higher-quality intermediate solution steps and more accurate final answers than existing approaches. Moreover, we show that by changing the operation prompt, we can control our framework to generate *different solution paths* that correctly solve the same MWP.

2 Notation

We first define all of terms and components in our approach. The MWP is defined as $Q = \{q_1, q_2, \ldots, q_n\}$ where q_i represents a token, which is either a numerical value, a mathematical operator, or a word/sub-word, and the corresponding step-by-step solution is defined as $S = \{S^1, S^2, \ldots\}$,

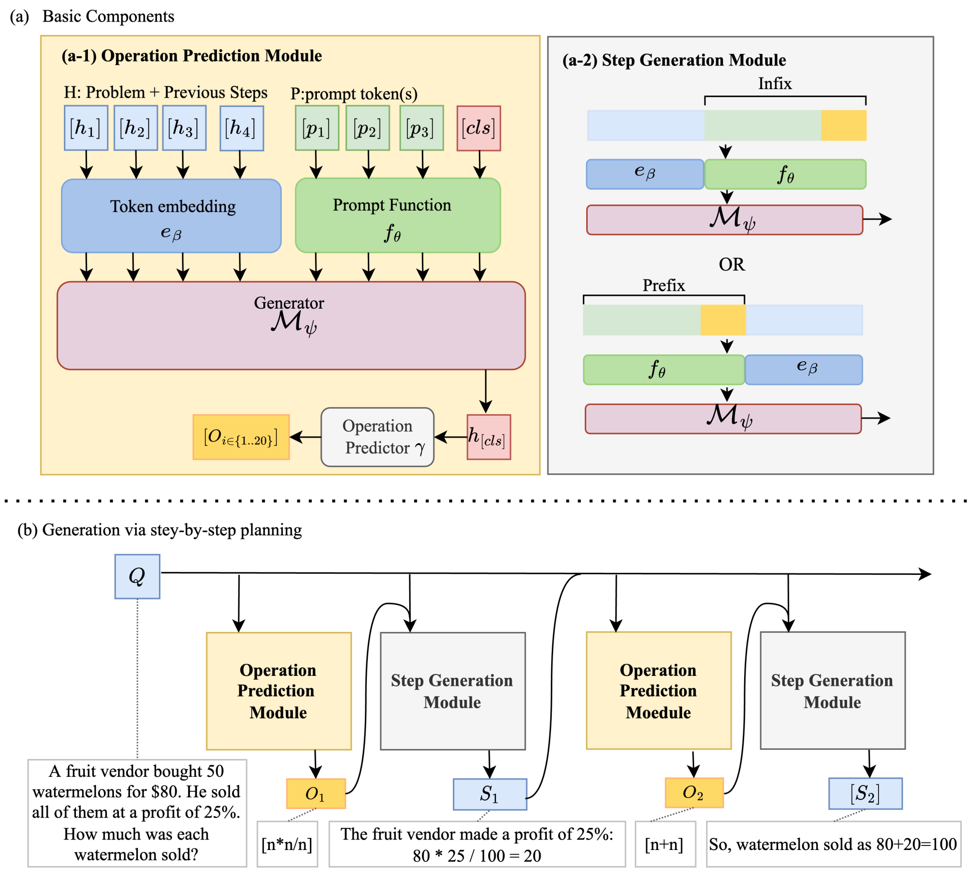

Figure 1: An overview of our step-by-step MWP solution generation approach. Planning-LM first predicts the next step operation hint **(a-1)** and controls the next step generate via the predicted operation hint **(a-2)**. **Figure (b)** shows the overview generation process by given the question Q.

where S^i denotes i^{th} step of the solution. For any step S^i, we have $S^i = \{s^i_1, s^i_2, \ldots\}$, consisting of a sequence of tokens. Next, we define our prompt with two parts. The first part is the textual instruction prompt, which contains words that LMs can understand and the second part is the mathematical operation prompt, which is a special token that contains next step operation information. We denote the instruction prompt as $P = \{p_1, p_2, \ldots\}$, where s_i represents a word/sub-word token and the operation prompt as $O = \{o\}$, where o is a categorical token as $o \in \{1 \ldots |O|\}$. We define H_i as the solution context, i.e., the history at step S^i, which consists of the problem Q and all previous steps, $\{S^1, \ldots, S^{i-1}\}$. $\mathcal{M}$ denotes the base pre-trained LM and e is its corresponding token embedding function. Finally, we define f as the prompt embedding function. Both e and f can map tokens into $\mathcal{R}^K$ where K is the hidden state dimension of the LM.

3 Methodology

We now define our MWP solution generation task and detail the specifics of our approach. Our task is that given a question Q, we need to generate a step-by-step solution $S = S^1, S^2, \ldots$, with each step consisting of a combination of textual and mathematical tokens, to reach the final answer. We formulate the problem as a step-wise controllable generation task using prompts-based LM fine-tuning. Figure 1 shows an overview of our approach, including its two main components: First, we utilize the MWP and the solution history to plan and predict next mathematical operation to apply in the next step. Second, we use the predicted operation prompt with instruction prompt to guide the next step generation process. Our key technical challenges are (i) how to learn a solution planning strategy to transition from step to step and (ii) once we have the next operation, how to apply and design prompts to guide the generative LM to generate the next step to follow the plan.

3.1 Operation Prediction

Our first step is to predict the mathematical operation to be applied in the next step. To achieve this, we concatenate the solution history H and a crafted instruction prompt P (e.g.,"What is next operation?") followed by the special token "$[cls]$" as input to an LM. We encode solution history tokens with a vocabulary embedding function e_β and instruction prompt tokens with a separate prompt embedding function f_θ; β and θ are the parameters of these parts, i.e., the embedding layer in an LM. Then, we obtain the representation of the solution history as the final layer hidden state of the LM, i.e., $\mathcal{M}$. To predict the operation action of the next step, we use a one-layer, fully-connected network as the classifier, with weight w_γ, to obtain an operation score vector for each valid math operation $s \in [0,1]^{|O|}$, where $|O|$ is the number of operation classes, as

$$s = w_\gamma \dot{h}_{[cls]},$$

where γ is the set of parameters for the classifier. Since we need to use an LM for step generation, introducing a separate LM for operation prediction leads to a large number of parameters. Therefore, we use the same LM for both operation planning and solution step generation. The objective function for operation planning is the cross-entropy loss on operators:

$$\mathcal{L}_{CE} = -\sum_{i}^{|O|} t_i \log(\frac{\exp s_i}{\sum_j^{|O|} \exp s_j}),$$

where s_i is the score of operation class i. t_i is an indicator such that $t_i = 1$ when i is the true label and $t_i = 0$ otherwise. We obtain true labels by extracting mathematical operations from each step of the solution in the training data, which we detail below.

3.2 Controllable Step Generation

Once we have the predicted operation O, we append the corresponding prompt to the instruction prompt P to form our final prompt for the step generation LM. Our task becomes a controllable generation task: given history H and the prompt $[P;O]$ that plans the next step, our goal is to generate the next step S token-by-token. We generate a step $S_i = \{s^i,..., s^i_T\} = \{s^i_j\}_{j=1}^{T}$ according to

$$p(S_i|[P_i;O_i],H_i) = \prod_{j=1}^{T} p(s^i_j|[P_i;O_i],H_i,\{s^i_j\}_{j=1}^{j-1}).$$

Then, the overall generation process for step-by-step solution S with N steps can be written as follows:

$$p(S) = \prod_{i=1}^{N} p(S_i|[P_i;O_i],H_i)p(O_i|H_i).$$

The step generation objective is given by the negative log-likelihood objective function

$$\mathcal{L}_{LM} = -\sum_{i=1}^{N} \log p_{\beta,\theta,\gamma,\psi}(S_i|[P_i;O_i],H_i), \quad (1)$$

where the set of parameters include previously defined β, θ, γ and the LM parameters ψ. β and ψ are fine-tuned while θ and γ are learned from scratch. We also investigate two ways to position the prompt in LM input: as prefix, where we place them at the beginning, i.e., the input is given by $[P;O;H]$ and as infix, where we append the prompt after the history, i.e., the input is given by $[H;P;O]$.

3.3 Prompt Design

Our prompt consists of two parts: the instruction prompt gives the LM general instructions on what to generate, while the operation prompt provides the specific guidelines for the mathematical calculation involved in the next step. For the instruction part, we apply prompt mining to find good instructions, i.e., word tokens that are the most informative for the LM to accomplish the desired task. For the operation prompt, we extract 20 common operations from the training data, such as one step addition $[n+n]$, subtraction $[n-n]$, multiplication $[n*n]$, etc. Some operations can also reflect multi-step operations, e.g., $[n+n+...]$, $[n+n-n]$, or $[(n+n)/n]$. We also use several other prompt types without mathematical operations, such as $[ans]$, which means "solution found, end the whole generation" and $[statement]$, which means that the next step involves no math calculation and only textual explanations.

3.3.1 Operation prompts

We initialize the embedding of each math operation token as the original pre-trained LM's embedding of the mathematical operator token instead of initializing them randomly (Liu et al., 2021c). For example, we initialize the operations action token $[n+n]$ with the same value as embedding of the "+" token in the pre-trained model. For operation classes that contain multiple operations, we initialize the embedding to the mean of all operation embeddings involved. We do this since initializing a new token with related embeddings has been proven to be effective on speeding up the training process of LM-based models (Li and Liang, 2021; Zhong et al., 2021; Lester et al., 2021; Hambardzumyan et al., 2021; Liu et al., 2021b).

3.3.2 Prompt mining through paraphrasing

For the instruction prompt, finding good prompts is an art that takes time and experience (Liu et al., 2021b). Thus, we apply prompt mining through paraphrasing by first starting with a seed prompt (e.g. "The next step operation is: ") and paraphrase it into a set of other candidate prompts with similar meaning (Yuan et al., 2021). Then, we tune the model with these candidates by treating them as hyper-parameters and select the one that performs best on the target task. We find that anchor tokens (e.g. "?") are helpful and leads to good performance, which is consistent with prior work (Liu et al., 2021c).

4 Experiments

We now detail a series of experiments that we conducted to validate the effectiveness of our proposed planning-LM approach on step-by-step MWP solution generation.

4.1 Data and Prepossessing

Since our focus is on MWP solution generation with explanation, GSM8K (Cobbe et al., 2021) is a good fit for our purpose. This dataset contains 8.5K high-quality and linguistically diverse MWPs, where each MWP has 2-8 solution steps. Each step contains arithmetic calculations involving (one or a combination of) basic math operations ($+$, $-$, $\times$, $\div$), together with textual explanations. We segment the data into 7.5K training problems and 1K test problems the same way as (Cobbe et al., 2021). For each MWP, we split the solution into steps according to the period symbol "." at the end of sentences. We restrict ourselves to the top-20 most frequent mathematical operations after merging some operations that have similar meaning, e.g., $[n+n+n]$ and $[n+n+n+n]$ are both labeled as "multi-step addition" to avoid highly infrequent operations. The supplementary contains a detailed list of all operations.

4.2 Metrics and baselines

We need a variety of different metrics to understand the effectiveness of our planning-LM approach. Since generating meaningful steps is key, we use the **BLEU** metric (Papineni et al., 2002) to evaluate language generation quality. For intermediate steps, we use the equation match accuracy (**ACC-eq**) metric to evaluate whether a generated step contains a math expression (including numbers) that matches the ground truth. Since LMs generate math equation as strings, we decompose the equation string into tokens and calculate the token level match rate instead of the overall string match. We also use the operation match accuracy (**ACC-op**) metric to evaluate whether a generated step's operation label matches the ground truth. For the final answer, we use the **solve rate** metric to evaluate whether the model generates the final correct answer to each MWP.

4.3 Experimental settings

We conduct two experiments to verify the effectiveness of our planning-LM framework. In the first single-step experiment, we input the question and ground-truth solution steps to the model and let it generate the next step and calculate the ACC-eq and ACC-op metrics for each generated step. Since some of the steps are too short, yielding a high variance in BLEU scores, we concatenate all generated steps and calculate the overall BLEU metric between the ground truth solution and this true history-informed solution. In the second all-step experiment, we only provide the model with the MWP and have it generate all solution steps. We then calculate the solve rate metric to evaluate whether the final generated answer is correct. We choose GPT-2 (117M parameters) and GPT-2-medium (345M) as our base models and compare the generation results between LM fine-tuning and planning-LM. Meanwhile, we perform another experiment using the ground truth operation prompt as input for planning-LM to generate the next step. The result, an upper bound on the performance of planning-LM, reflects the effectiveness of low-level token-by-token generation in each step, while ACC-eq and ACC-op reflect the effectiveness of high-level mathematical operation planning across steps.

We emphasize that we cannot compare the performance of planning-LM to recent works such as (Cobbe et al., 2021; Wang et al., 2022) since we do not have access to models such as GPT-3 (Brown et al., 2020), Palm (Chowdhery, Aakanksha and others, 2022), or LaMDA (Thoppilan et al., 2022)

Table 1: Planning-LM using mathematical operation prompts outperforms fine-tuning LMs for both small and medium-sized GPT-2. Moreover, Planning-LM with a small GPT-2 achieves performance comparable to fine-tuning medium GPT-2, implying that our method can make a smaller model rival larger ones fine-tuned in the traditional way.

Model	**BLEU**	**ACC-eq**	**ACC-op**	**Solve Rate**
Fine-tuning GPT-2 (117M)	34.3	49.4	55.1	8.1
Planning-GPT-2 with operation classifier (117M)	35.4	56.7	61.6	14.1
Planning-GPT-2 with ground truth prompt (117M)	42.1	71.2	93.1	27.6
Fine-tuning GPT-2-medium (345M)	38.1	58.1	61.1	16.1
Planning-GPT-2-medium with operation classifier (345M)	39.5	61.8	65.2	20.1
Planning-GPT-2-medium with ground truth prompt (345M)	45.1	75.3	91.0	35.2

Table 2: Ablation results for different components of our approach. Most components contribute significantly.

Method Component					**Metric**			
Infix	Prefix	Prompt function	Prompt mining	Opeartion Predictor	BLEU	ACC-eq	ACC-op	Solve Rate
✓		✓	✓	✓	**35.4**	**56.7**	61.6	**14.1**
	✓	✓	✓	✓	33.7	52.1	**63.2**	10.4
✓			✓	✓	33.1	51.9	58.4	10.2
✓		✓		✓	33.9	55.1	59.9	13.2
✓		✓	✓		34.1	54.2	60.1	13.5

used in these works. Such a comparison would be unfair since these models are much larger than models we have access to: GPT-3 has 175B parameters while PaLM has 540B, which are both more than 1000$\times$ larger than GPT-2. We also emphasize that the technical approaches proposed in these works, such as using verifiers or majority vote (Wang et al., 2022) during the decoding phase, or designing chain-of-thought prompts (Wei et al., 2022) follow different directions than our work: they all generate the entire solution in one shot, without breaking it down into steps and planning each step. One can combine our approach with these ones.

4.4 Quantitative Result

Table 1 lists the performance of all methods on all metrics across the two experiments. We see that planning-GPT-2 with our operation classifier outperforms simply fine-tuning GPT-2: GPT-2 with planning achieves 56.7 (+7.3) in ACC-eq, 61.6 (+6.5) in ACC-op, 14.1 (+5.0) in solve rate, and 35.4 (+1.1) in BLEU. However, there is a big gap between these numbers and those when the ground truth prompt is provided, which suggests planning is highly effective but our next-step mathematical operation classifier still has considerable room for improvement. We also observe that when the number of parameter increases, a similar trend holds for GPT-2-medium. We emphasize that with the planning component, which introduces only around 10K new parameters for the MWP solving task, a base GPT-2 model with 117M parameters performs similarly to a much larger base GPT-2-medium model with 345M parameters. This observation shows that our planning approach is highly parameter-efficient for MWP solving.

To validate the effectiveness of each component in our planning-ML approach, we conduct an ablation study on four different components: using prefix or infix prompts, applying fixed or fine-tuned mathematical operation prompts, instruction prompt mining, and the operation classifier. Among different settings, we find that using infix, fine-tuned mathematical prompts, and the operation predictor improve performance the most. We found that infix prompts are significantly better than prefix prompts, which is different from observation made in prior work (Li and Liang, 2021), which may be explained by the incompatibility between prefix prompting and step-by-step generation: prefix prompts put the most important instruction at the front of the LM input, making all generated tokens attend to it, which leads to improved operation classification accuracy but worse generation performance on other tokens.

4.5 Qualitative Analysis

Table 3 shows a two examples that compare the full step-by-step solutions generated by our planning-LM approach and fine-tuning LMs. In Examples 1 our approach successfully predicts the next operation step and outputs the correct equation, while fine-tuning GPT-2 cannot. Meanwhile, the quality of corresponding natural language explanation also improved. On the other hand, in Examples 2, both approaches do not produce equations that match the ground truth. For our approach, the generated solution step is consistent with the prediction, but since the predicted operation is incorrect, planning GPT-2 winds up generating worse results. It is worth noting that even with this potential drawback, the benefit of our approach still outweigh the potential negatives compared to fine-tuning GPT-2. We note that one substantial future improvement is to use the confidence of the operation classifier to decide whether to use the predicted operator as prompt.

Surprisingly, in Example 2, we observe that planning GPT-2 generates a valid alternative solution strategy, even though the predicted mathematical operation differs from the ground truth. Therefore, we conduct a follow-up experiment by giving the model a hand-crafted plan via operation prompts and checking whether it can generate an alternative correct solution strategy. Table 4 shows the results. Feeding plans I and II enables the model to generate the correct final answer among the four strategies we used; the generated solutions follow the operation steps given, which indicates that the model has some reasoning ability and is able to extract some meaningful patterns from the data. Plan III results in a flawed solution and Plan IV failed since we do not have an operation class that matches the step. For plan III, the first step, $[n+n+\ldots]$, is not seen often enough in the training data. For plan IV, $(n+n)\times n$ is not seen in the training data either. However, we note that in this case, using the closest operation, $[n+n\times n]$, results in a solution that gets very close to the correct final answer. These results suggest that a better representation of the operation prompt is crucial for future work since our current approach is limited to a finite number of predefined operations; a prompt operation *generator* rather than classifier could be a better choice for a wide variety of mathematical operations. We also note that this flexibility gives our planning-LM approach potential to be useful in real-world applications. For example, these solution plan controls may encourage students to experience different strategies and be more creative.

Table 3: Demonstrations of the step-by-step generated solutions comparing planning-LM and fine-tuning LM. Green and red highlight the correct and incorrect equation in the generated intermediate steps, respectively.

1.Question: Marie ordered one chicken meal that cost \$12, 5 packs of milk that cost \$3 each, 4 apples that cost \$1.50 each, and some pizza boxes. Marie paid a total of \$50. How many pizza boxes did Marie order if each box costs \$8.50? **Previous Steps:** Five packs of milk cost $\$3 \times 5 = 15$. Four apples $\$1.50 = \$6$.
Ground Truth: The total cost of the chicken meal, milk, and apples is \$12 + \$15 + \$6 = \$33. **Fine-tuning:** Some boxes of pizza cost \$8.50 - \$15 - \$6 = \$2 . **Planning with predicted step [n+n+...]:** So Marie spent a total of \$15 + \$12 + \$6 = \$33 .
2.Question: Eliza's rate per hour for the first 40 hours she works each week is \$10. She also receives an overtime pay of 1.2 times her regular hourly rate. If Eliza worked for 45 hours this week, how much are her earnings for this week? **Previous Steps:** None
Ground Truth: Eliza is entitled to 4540=5 hours overtime pay. **Fine-tuning:** She is paid $5 * \$10 = \$50$ for the first 40 hours. **Planning with predicted step [n*n]:** Eliza makes \$10/hour * 40 hours = \$400 per week .

5 Related work

MWP solver A large body of recently proposed MWP solvers parses an MWP into its underlying equation, which has been a very active research area with a plethora of related work. These works differ mainly in the technical approaches which broadly fall in three categories. First, some works explore MWP solving via reinforcement learning, which rewards the model with the correct answer generated (Huang et al., 2018; Wang et al., 2018b). RL methods generally requires a sizable dataset and can be unstable to train, which may not be suitable for most MWP datasets that are only of modest sizes. Second, some works exploit the combination of symbolic- and neural-network-based approaches, e.g., by combining a pre-defined symbolic patterns such as solution templates (Wang et al., 2019) and symbolic tree structures of equations (Xie and Sun, 2019; Li et al., 2020; Qin et al., 2020; Wang et al., 2018a; Wu et al., 2020; Zhang et al., 2021). These methods can be significantly constrained by these patterns and it may be challenging to generalize them to other MWPs whose solutions are not expressed by these patterns. Lastly, some works build on LLMs via special fine-tuning and inference techniques. Specifically, (Cobbe et al., 2021) fine-tunes an LLM on a collection of MWPs with intermediate steps. (Lewkowycz et al., 2022) simply fine-tunes an LLM by a large collection of math-specific dataset; the fine-tuned model generates the solution

Table 4: Qualitative examples of using our planning-LM to plan for four different but all valid solution strategies to achieve the same correct result for a given MWP.

Question: Two trains leave San Rafael at the same time. They begin traveling westward, both traveling for 80 miles. The next day, they travel northwards, covering 150 miles. What's the distance covered totally in the two days? Ans: 460
Plan I: $[n+n]$ -> $[n*n]$ -> $[Ans]$ **Result:** The total distance covered in the two days is 80 + 150 = 230 miles. The total distance covered in the two days is 230 * 2 = 460. Answer is 460.
Plan II: $[n*n]$ -> $[n*n]$ -> $[n+n]$ -> $[Ans]$ **Result:** The first train covers 80 * 2 = 160 miles. On the second day, the trains covered 2 trains * 150 miles/train = 300 miles. Total distance covered is 160 + 300 = 460 miles. Answer is 460.
Plan III: $[n+n+\ldots]$ -> $n+n$ -> $[Ans]$ **Result:** The total distance covered in the two days is 80 + 80 + 150 = 310 miles. The total distance covered in one day is 310 + 150 = 460 miles. Answer is 460.
Plan IV: $[n+n*n]$ -> $[Ans]$ **Result:** The total distance covered by trains in the two days is 150 + 80 * 2 = 310 miles. Answer is 310.

via chain-of-thought prompting (Wei et al., 2022). Our work is most similar to the those in the last category with key differences: first, our work focuses on the accuracy of each solution step whereas those above focus on only the accuracy of the final answer. Second, our work proposes a novel hierarchical planning method for fine-tuning whereas those above simply fine-tunes the model with the ordinary language modeling techniques that lack planning and reasoning.

Controllable text generation Given the rise of LLMs, controllable generation methods that guide these large models to generate desirable content and avoid potential pitfalls such as bias (Bender et al., 2021) has been a recent research trend. These controllable generation methods generally fall into two categories. Works in the first category modifies the token distribution at each time step to achieve controllable generation via gradient-based methods (Dathathri et al., 2020), external classifier-based methods (Krause et al., 2020; Liu et al., 2021a), or resampling (Clark et al., 2020; Bhattacharyya et al., 2021; Bakhtin et al., 2021). Works in the second category fine-tunes the LLM via either language modeling (Shirish Keskar et al., 2019) or reinforcement learning (Khalifa et al., 2021). These works focus on controllable generation for natural language and study nuanced control attributes such as topic and sentiment that can only be matched implicitly. In contrast, our work focuses differently on both natural and mathematical language, which involves control attributes, e.g., math operation hints in the form of equations that need to be matched exactly. Therefore, our work studies a different problem and proposes a novel methodology different from those above.

6 Conclusion and Future work

This paper addresses the problem of performing fine-grained, step-by-step controllable solution generation for math word problems. We proposed an approach that combines planning and language models that can generate interpretable solution steps. Our approach leverages pre-trained language models in two ways: at each step, plan the mathematical operation to be applied, followed by using these plans as prompts to control the token-by-token generation of each step. We demonstrated that with a minimal amount of additional parameters introduced, our approach significantly improves MWP solving performance over simply fine-tuning language models. We showed that due to the interpretability and high controllability of operation prompts, we can use our approach to generate solutions with alternative strategies by giving it different solution plans.

Although we have shown that our approach offers some significant improvements over existing methods, we believe that there are many possible ways to take this work even further. First, the verifier and self-consistency tools are proven to improve the model's generation results effectively. One can combine these tools with our approach to further enhance the quality of the generated solutions. Second, we can use a generator instead of a classifier to generate a more flexible set of operation prompts, making them more representative and meaningful. Third, to eliminate the drawback where inaccurately generated operation prompts would mislead the next step, we can apply a verifier to evaluate the reliability of the generated operation prompts. When the reliability is low, we ditch the operation prompt to prevent it from guiding the model into an incorrect path. Fourth, we could further explore generation via planning by supplying the model with not just operation prompts, but number and entity prompts as well, which are both key elements in math word problems.

References

Anton Bakhtin, Yuntian Deng, Sam Gross, Myle Ott, Marc'Aurelio Ranzato, and Arthur Szlam. 2021. Residual energy-based models for text. *J. Mach. Learn. Res.*, 22(40):1–41.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? In *Proc. ACM Conf. Fairness Accountability Transparency*, page 610–623.

Sumanta Bhattacharyya, Amirmohammad Rooshenas, Subhajit Naskar, Simeng Sun, Mohit Iyyer, and Andrew McCallum. 2021. Energy-based reranking: Improving neural machine translation using energy-based models. In *Proc. Annu. Meeting Assoc. Comput. Linguistics and Int. Joint Conf. Natural Lang. Process.*, pages 4528–4537.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2021. On the opportunities and risks of foundation models.

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, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey 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. 2020. Language models are few-shot learners. *CoRR*, abs/2005.14165.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V. Le. 2020. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *International Conference on Learning Representations*.

Chowdhery, Aakanksha and others. 2022. Palm: Scaling language modeling with pathways. arXiv preprint `https://arxiv.org/abs/2204.02311`.

Kevin Clark, Minh-Thang Luong, Quoc Le, and Christopher D. Manning. 2020. Pre-training transformers as energy-based cloze models. In *Proc. Conf. Empirical Methods Natural Lang. Process.*, pages 285–294.

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

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *Proc. Int. Conf. Learn. Representations*.

Edward A Feigenbaum and Julian Feldman, editors. 1995. *Computers and Thought*. MIT Press, London, England.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. WARP: word-level adversarial reprogramming. *CoRR*, abs/2101.00121.

Danqing Huang, Jing Liu, Chin-Yew Lin, and Jian Yin. 2018. Neural math word problem solver with reinforcement learning. In *Proc. ACL*, pages 213–223.

Jeffery D. Karpicke. 2012. Retrieval-based learning: Active retrieval promotes meaningful learning. *Current Directions Psychol. Sci.*, 21(3):157–163.

Muhammad Khalifa, Hady Elsahar, and Marc Dymetman. 2021. A distributional approach to controlled text generation. In *Proc. Int. Conf. Learn. Representations*.

Kenneth R. Koedinger, Jihee Kim, Julianna Zhuxin Jia, Elizabeth A. McLaughlin, and Norman L. Bier. 2015. Learning is not a spectator sport: Doing is better than watching for learning from a mooc. In *Proc. Conf. Learn. Scale*, pages 111–120.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. 2020. GeDi: Generative

Discriminator Guided Sequence Generation. *arXiv e-prints*.

Vivek Kumar, Rishabh Maheshwary, and Vikram Pudi. 2022. Practice makes a solver perfect: Data augmentation for math word problem solvers. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4194–4206, Seattle, United States. Association for Computational Linguistics.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *CoRR*, abs/2104.08691.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models.

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

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *CoRR*, abs/2101.00190.

Zhongli Li, Wenxuan Zhang, Chao Yan, Qingyu Zhou, Chao Li, Hongzhi Liu, and Yunbo Cao. 2021. Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems. *CoRR*, abs/2110.08464.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. 2021a. DExperts: Decoding-time controlled text generation with experts and anti-experts. In *Proc. Annu. Meeting Assoc. Comput. Linguistics and Int. Joint Conf. Natural Lang. Process.*, pages 6691–6706.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021b. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *CoRR*, abs/2107.13586.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021c. GPT understands, too. *CoRR*, abs/2103.10385.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Jinghui Qin, Lihui Lin, Xiaodan Liang, Rumin Zhang, and Liang Lin. 2020. Semantically-aligned universal tree-structured solver for math word problems. In *Proc. EMNLP*, pages 3780–3789.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. CTRL: A Conditional Transformer Language Model for Controllable Generation. *arXiv e-prints*.

Sowmya S. Sundaram, Sairam Gurajada, Marco Fisichella, Deepak P, and Savitha Sam Abraham. 2022. Why are NLP models fumbling at elementary math? A survey of deep learning based word problem solvers. *CoRR*, abs/2205.15683.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Kathleen S. Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed H. Chi, and Quoc Le. 2022. Lamda: Language models for dialog applications. *CoRR*, abs/2201.08239.

Lieven Verschaffel, Stanislaw Schukajlow, Jon Star, and Wim Van Dooren. 2020. Word problems in mathematics education: a survey. *ZDM*, 52(1):1–16.

Candace A. Walkington. 2013. Using adaptive learning technologies to personalize instruction to student interests: The impact of relevant contexts on performance and learning outcomes. *J. Educ. Psychol.*, 105(4):932–945.

Jianhong Wang, Yuan Zhang, Tae-Kyun Kim, and Yunjie Gu. 2020. Modelling hierarchical structure between dialogue policy and natural language generator with option framework for task-oriented dialogue system. *CoRR*, abs/2006.06814.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018a. Translating a math word problem to a expression tree. In *Proc. EMNLP*, pages 1064–1069.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018b. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Proc. AAAI*, pages 5545–5552.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In *Proc. AAAI*, volume 33, pages 7144–7151.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903.

Qinzhuo Wu, Qi Zhang, Jinlan Fu, and Xuanjing Huang. 2020. A knowledge-aware sequence-to-tree network for math word problem solving. In *Proc. EMNLP*, pages 7137–7146.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *Proc. IJCAI*.

Weizhe Yuan, Graham Neubig, and Pengfei Liu. 2021. Bartscore: Evaluating generated text as text generation. *CoRR*, abs/2106.11520.

Qiyuan Zhang, Lei Wang, Sicheng Yu, Shuohang Wang, Yang Wang, Jing Jiang, and Ee-Peng Lim. 2021. NOAHQA: Numerical reasoning with interpretable graph question answering dataset. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4147–4161, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual probing is [MASK]: learning vs. learning to recall. *CoRR*, abs/2104.05240.

Yanyan Zou and Wei Lu. 2019. Text2math: End-to-end parsing text into math expressions.

A Hyper-parameters

We use a learning rate of 5e-5, a batch size of 8, and 10 epochs for all training processes. We set "what is the next operation?" as our instruction prompt and apply *calculators* to avoid calculation errors and *greedy decoding* during token generation. Model training is carried out on an NVIDIA RTX 3090 GPU.

B Optimization

Although our entire approach can be trained together in an end-to-end way, we found that optimizing the operation prediction model and fine-tuning the LM/prompts for step generation asynchronously leads to better performance. Our intuition is that the operation predictor is a high-level decision-making policy for the entire solution while the LM generation process is a low-level (token-by-token) decision-making process for the current step. Optimizing these two modules simultaneously may cause inconsistency since the operation predictor may make a decision based on LM parameters that also need to be updated. Therefore, we first optimize the parameters of the generation LM and prompts with the step generation task loss, using ground truth operation labels, which we extract from the mathematical part of each step in the training data. Then, we iterate between freezing both the LM M and the prompt function f while tuning the operation predictor and switching the two. In this way, we can guarantee the whole model to converge in a stable process (Wang et al., 2020).

C List of all hand-crafted operations classes

Details in table 5

D Examples of control generation

Table 6 shows the generated step apply different operation prompts on same input. This table demonstrates the generated results from applying different operation prompts with the same input to the model. We observe that when the operation prompt is logical and aligned with solving the question, the generated result follows the guidance given by the operation prompt. In contrast, when the operation prompt does not make sense, the generated result will not obey its directions. Details in table 5

Table 5: Appendix C: list of all hand-crafted operations classes

no.	shortcut	description	no.	shortcut	decription
1	[n+n]	one-step addition	12	[n-n*n]	multiplication then subtraction
2	[n-n]	one-step subtraction	11	[n*(n/n)]	multiplication by a fraction
3	[n*n]	one-step multiplication	13	[(n/n)-(n/n)]	fraction subtraction
4	[n/n]	one step division	14	[(n/n)+(n/n)]	fraction addition
5	[n+n+. . .]	multi step addition	11	[(n/n)*(n/n)]	fraction multiplication
6	[n-n-. . .]	multi-step subtraction	16	[mixed]	other combination
7	[n*n*. . .]	multi-step multiplication	17	[ans]	solution found, end the whole generation
8	[n+n*n]	multiplication then addition	18	[statement]	involve no math calculation and only textual explanation
9	[n+n-n]	addition then subtraction	19	[assign]	assign a value to a paramter
10	[n+n/n]	division then addition	20	[define]	define a parameter

Table 6: Appendix D: Generated step apply different operation prompts on same input.

Input: Maggie spent a quarter of her money, while Riza spent one-third of her money.
They each had $60. How much money do the two of them have left?

no.	action	generated step
1	[n+n]	Riza and Maggie had $60 + $60 = $120 ✔
2	[n-n]	Riza had $60 - $60 = $0 left after spending some money on Maggie. ✔
3	[n*n]	Riza spent $60 x 0.33 = $20 ✔
4	[n/n]	Maggie had $60/4 = $15 left. ✔
5	[n+n+. . .]	Riza spent $60 + $60 + $60 = $180. ✔
6	[n-n-. . .]	Riza had $60 - $60 - $30 = -$30 ✔
7	[n*n*. . .]	Riza spent $60 x 1/3 = $60.
8	[n+n*n]	Maggie had $60 + $60*0.25 = $56 left after spending a quarter of her money. ✔
9	[n+n-n]	Maggie had $60 - $60 = $0 left after spending a quarter of her money.
10	[n+n/n]	Riza had $60 + $60/3 = $80 left. ✔
11	[n-n*n]	Riza had $60 - $30 = $30 left after spending a quarter of her money.
12	[n*(n/n)]	Riza spent $60 x 1/3 = $20. ✔
13	[(n/n)-(n/n)]	Riza and Maggie had $60/4 - $60/3 = $5 left. ✔
14	[(n/n)+(n/n)]	Riza and Maggie total had $60/4 + $60/3 = $35 left. ✔
11	[(n/n)*(n/n)]	Riza had $60 * 1/3 = $20 left.
16	[mixed]	Riza had $60/4 - $60/3 = $15 left.
17	[ans]	The answer is $60 ✔
18	[statement]	Riza had $60 in total. ✔
19	[assign]	Riza had x=$60. ✔
20	[define]	Let x be the money Riza had ✔