

# Automatic Generation of In-Context Math Examples Using Multi-Modal Consistency

Anonymous ACL submission

## Abstract

Large Language Models (LLMs) have advanced Natural Language Processing (NLP) tasks but limited in mathematical reasoning. To address this, few-shot examples are used in prompts for in-context learning. However, existing methods require annotated datasets, resulting in higher computational costs and lower quality examples. To mitigate these limitations, we propose APMath, a framework that automatically generates high-quality in-context examples to enhance LLMs' mathematical reasoning. APMath ensures consistency across different modalities (e.g., Chain-of-Thought (CoT), code snippets, and equations) by generating and selecting mutations that improve response consistency. Evaluated on four math problem datasets, APMath outperforms six baselines, with LLM accuracy ranging from 87.0% to 99.3%. It surpasses the state-of-the-art in-context example retrieval method in three of the four datasets by 1.9% to 4.4%, without relying on an external annotated dataset.

## 1 Introduction

LLMs have achieved state-of-the-art performances in many NLP applications (Min et al., 2024). However, they exhibit limited proficiency in solving mathematical problems (Rae et al., 2021; Srivastava et al., 2022). This limitation arises due to the fact that math tasks require understanding complex multi-step reasoning to solve the problems. To overcome the deficiency in math-solving capability, in-context learning has been proposed (Wei et al., 2022; Zhang et al., 2023). These approaches leverage few-shot examples, each consisting of math problem and its explanation, embedding the examples into prompts to facilitate learning within the context towards improved performance.

However, existing in-context learning for math tasks has limitations. The generation of in-context examples requires extensive resources and often depends on large, externally annotated datasets (Wei

et al., 2022; Zhang et al., 2023). This process could be labor-intensive, involving manual curation of examples, and computationally expensive, relying on sophisticated retrieval models to find appropriate examples from the external datasets. Furthermore, the scale of the external dataset may be constrained, limiting the search space for identifying suitable math problems and their explanations for the target problem. These limitations hinder the automatic generation of appropriate in-context examples, limiting the practicality and scalability of in-context learning for math tasks.

To overcome these limitations, we identify two challenges. First, it is crucial to generate math problems relevant to the target problem for in-context learning. The relevant problems for in-context learning provides appropriate reasoning algorithms to solve the target problem, and these algorithms ensure accurate reasoning explanations. However, the creation of such examples requires substantial costs due to manual curation and extensive search within annotated datasets. Second, the retrieval of high-quality in-context examples is challenging. The existing method evaluates the relevancy of the in-context example with the target problem by measuring the semantic similarity between them (Zhang et al., 2023). However, this approach does not always guarantee that the retrieved examples contain comprehensive or high-quality explanations that can assist in solving the target problem. Consequently, the quality of the LLM's response to the target problem may depend on the quality of explanations provided by the in-context examples. In scenarios where explanations are evaluated solely by manual assessment, automating the evaluation of these explanations remains a challenging.

To address these challenges, we employ a multi-modal technique for the generation and retrieval of in-context examples. Multi-modal learning, which integrates information from diverse sources such as text, images, and videos, has demonstrated poten-

tial in improving model comprehension. Prior studies have shown that models trained on multi-modal data can attain a deeper understanding of the content, consistently across different modalities (Lin and Parikh, 2015; Su et al., 2020; Radford et al., 2021). Similarly, LLMs also possess the ability to produce diverse forms of responses to the same mathematical problem, known as modality, such as generating CoT, composing code snippets, or formulating complex mathematical equations (Kojima et al., 2022; Wang et al., 2023b; Imani et al., 2023). The consistency of LLM responses across these modalities can act as an indicator for evaluating the confidence in LLM predictions. Consistency, defined as the degree of agreement among model predictions, has been a focus of prior research as a method for measuring the reliability of responses, leading to accurate answers (Wang et al., 2023b; Imani et al., 2023). The convergence of the consistency across independent modalities suggests a lower likelihood of systematic bias or errors being present only in a single modality. Consequently, this aids in estimating the accuracy of LLM responses. Accordingly, the key insight of our work lies in leveraging consistency across modalities, combined with few-shot learning techniques, to improve model performance.

In this work, we present an automated in-context prompting approach for math problem, referred to as APMath, that addresses the above challenges with the aid of mutation and consistency over modalities. APMath operates by initially generating a collection of mutated math problems and their corresponding LLM responses across various prompt modalities. This procedure ensures that the mutation maintains the same reasoning algorithm utilized for solving the target problem, resulting in potentially the most relevant in-context examples, addressing the first challenge. Subsequently, APMath selects a subset of mutated examples that improves consistency of responses across modalities for the target math problem. This tackles the second challenge by evaluating LLM responses of mutations through consistency. By doing so, it elevates the confidence level of the LLM, thereby leading to a correct answer. Our experimental evaluation shows that APMath produces higher accuracies on four popular arithmetic reasoning datasets over OpenAI GPT large language models, including ASDiv (97.1%), SVAMP (87.0%), GSM8k (83.8%), and MultiArith (99.3%). These accuracies outperforms not only the single zero-shot prompt

baselines for all four datasets but also the state-of-the-art in-context example retrieval method in three out of the four datasets by 1.9% to 4.4%, without relying on an external annotated dataset.

## 2 Motivation

In this section, we present responses to an arithmetic math problem using in-context examples to motivate the development of APMath. Figure 1 illustrates an example from the SVAMP, a widely used dataset for arithmetic reasoning problems (Patel et al., 2021). This demonstrates the diverse responses of an LLM (GPT-3.5) to the same math problem under different settings. At the top, it presents the answers to the math problem across different modalities (CoT, code, and equation) in zero-shot setting, revealing that the LLM responses are inconsistent and the confidence level in the answers is low. For example, CoT response incorrectly concludes that “ $\$17 + \$7 = \$24$ ” (step 5), leading to the erroneous answer “*the answer is \$24*” (step 6). In contrast, the bottom of the figure shows in-context examples retrieved from the target problem by altering the numerical values (the yellow box). In this setting, the LLM’s answers across all modalities are consistent, resulting in a high confidence level and correct answers. Specifically, the accuracy of the CoT response is attributed to the underlying algorithm, represented by “ $x - 65 + 39 = 67$ ”, which is also applied to solve the target problem, “ $x - 17 + 10 = 7$ ” (step 4).

Overall, our study is motivated by the examples presented in Figure 1 with regard to two key facets. First, altering the numerical values in a math problem does not change the underlying reasoning algorithm used to solve it. Therefore, using the reasoning algorithm for in-context learning can provide relevant instructions that enable LLMs to accurately solve the problem. This insight facilitates the automated generation of math problems that are assumed to operate under the same reasoning algorithm. Second, the quality of LLM responses can vary across different modalities. Figure 1 demonstrates that answers to the math problem can differ across various modalities. This shows the effectiveness of using consistency across modalities to enhance the reliability of LLM responses. It motivates us to leverage the degree of the consistency as a metric for evaluating the confidence level of the responses for the automatic selection of high-quality in-context examples.

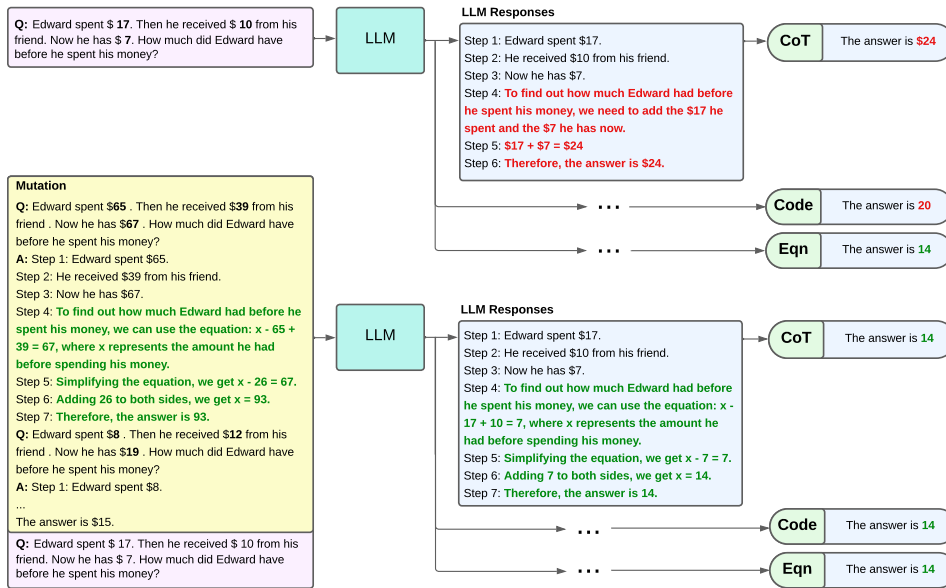


Figure 1: Responses of GPT-3.5 to an arithmetic reasoning problem derived from the SVAMP dataset (Patel et al., 2021). The top presents the responses across CoT, Python code, and Mathematical Equation in a zero-shot setting. The bottom shows the responses in the CoT modality utilizing in-context examples retrieved from the mutation.

### 3 Approach

We have developed APMath, an automated in-context example generation framework. APMath is structured with two main goals: 1. we employ a mutation technique on the target math problem for the generation of relevant in-context examples, considering these mutations as potential candidates for in-context examples. 2. we incorporate a genetic algorithm that selects mutations by maximizing the consistency of the responses to the target question to ensure the correct reasoning path in the prompt. Figure 2 presents an overview of APMath. The *Initial Consistency Computation Phase* first collects the preliminary responses of an LLM to the target problem across modalities. Next, the consistency of these responses is evaluated. If this consistency reaches its maximum value (i.e., all the answers are the same), then it returns the answer from the responses as the output. The inputs of this phase include the target problem in text and an LLM. In the scenarios where the consistency does not reach its maximum value, both the target problem and the LLM proceed to the *Target Problem Mutation Phase*. This phase mutates the target problem by altering the numerical values in the problem. Furthermore, we use the LLM to process these mutated problems and obtain their corresponding responses across modalities. A mutation is accepted if its responses are consistent across the modalities. This phase is crucial as it addresses the first

goal, which is to obtain the relevant problems for in-context learning. Additionally, we employ a *Mutation Selection by Consistency Optimization Phase* to achieve the second goal, which is the retrieval of high-quality in-context examples. In this phase, the LLM responses to the target problem are collected for each mutation, with the mutation prepended as an in-context example. The consistency of these responses is then evaluated. If this consistency reaches its maximum value, the response is used as the output. Otherwise, we further evaluate whether the new consistency score surpasses the previous score without the mutation or if the most consistent answer using the mutation differs from the previous one. If either condition is met, we update the prompts across modalities, the consistency score, and the most consistent answer with the new mutation as an in-context example. This process is repeated for all mutations, ultimately yielding the most consistent answer as the output.

#### 3.1 Initial Consistency Computation

Given a target problem and an LLM, we obtain top- $K$  responses for each modality. The answers are then extracted from these LLM responses, and their consistency is evaluated across the modalities. Specifically, given the top- $K$  answers for a specific modality  $mod$ ,  $ANS_{mod}$ , and its collection of LLM answers across modalities,  $ANS = \{ANS_{mod} | ANS_{mod} = LM(q_{tgt}, p_{mod}) \cap p_{mod} \in P_{MOD} \cap mod \in MOD\}$ , where  $LM$  is an LLM,

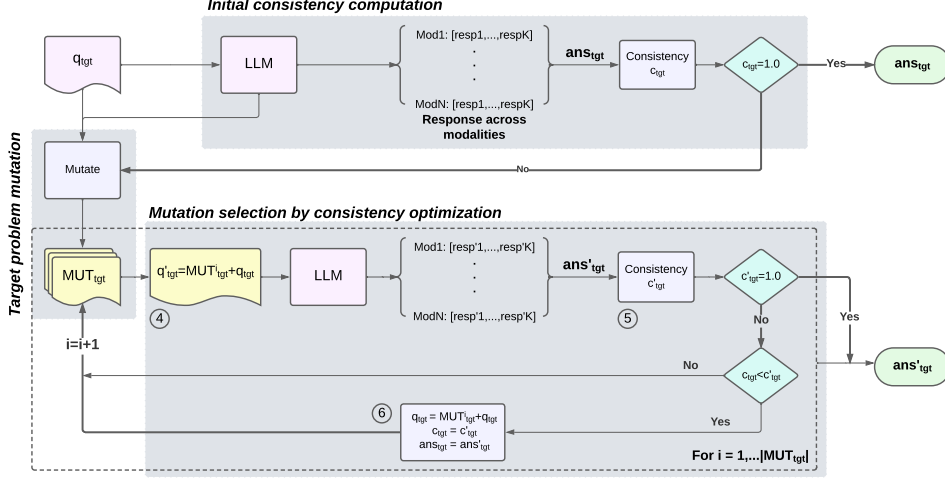


Figure 2: Overview of APMath.

$q_{tgt}$  is a target problem, and  $P_{MOD}$  is a set of prompt across modalities  $MOD$ . We define consistency across the modalities in Equation 3:

$$freq(a, ANS) = \sum_{ans \in ANS} \delta(a = ans) \quad (1)$$

$$SC(a, ANS) = \frac{freq(a, ANS)}{|ANS|} \quad (2)$$

$$C(a, ANS, w) = \sum_{mod \in MOD} w_{mod} \cdot SC(a, ANS_{mod}) \quad (3)$$

where  $freq(a, ANS)$  in Equation 1 represents the number of occurrences of a specific answer  $a$  within the answer set  $ANS$ . In Equation 2,  $SC$  denotes self-consistency score of the  $a$  for the  $ANS$  (Wang et al., 2023b). The consistency score across the  $MOD$ ,  $C(a, ANS, w)$  in Equation 3, is the weighted sum of  $SC$  for a unique answer  $a$  across the  $MOD$ . The modality belief weight  $w_{mod}$  represents the degree of empirical confidence of a specific modality and is set as a hyperparameter of APMath. An answer with a higher consistency score  $C$  for a specific answer indicates a higher level of confidence in the answer across modalities, while a lower score indicates lower confidence. If the consistency score of answer to the target problem fails to reach the maximum value, we proceed to the mutation phase, described in Section 3.2.

### 3.2 Target Problem Mutation

This phase generates a pool of problems that can potentially provide relevant knowledge for LLM to solve the target problem. This is achieved by mutating the target problem, resulting in a set of mutated problems. We operate under the assumption that a problem identical to the original one, but with different numerical values, follows the same

reasoning path in solving the original problem. The process first identifies the numerical values present in the target problem. These identified values are then randomly mutated; the original values are replaced with their mutated values, and responses are obtained from the LLM to these mutated problems across modalities. The validity of these mutations is verified by estimating the accuracy of the LLM’s responses to the mutated problems through response consistency across modalities. A mutation is considered acceptable if responses are consistent across modalities; otherwise, it is rejected. By repeatedly applying this mutation process to the target problem, this phase generates pairs of mutated problems and their corresponding LLM responses across modalities.

### 3.3 Mutation Selection by Consistency Optimization

Although the mutation phase is capable of producing numerous mutations, it is important to evaluate the quality of these mutations for their utility as in-context examples to accurately solve the target problem. To address this challenge, we have developed an optimization strategy aimed at improving consistency as defined in the Equation 3.

Algorithm 1 shows how the optimization strategy identifies the in-context examples. It takes the LLM, the target problem, prompts across modalities, initial answer and consistency score to the target problem measured in the phase described in Section 3.1, a pool of mutations and the maximum number of in-context examples to be used as inputs. If the algorithm attains a maximum consistency value, which means that all answers are the same across modalities, it returns this answer as

---

**Algorithm 1** Consistency optimization algorithm.

---

```
1: Input: a large language model  $LM$ , target question  $q_{tgt}$ , prompts over modalities  $P_{MOD}$ , initial the most consistent answer of the target problem  $ans_0$ , consistency score of the  $ans_0$   $mc_0$ , mutations  $M_{tgt}$ , maximum number of in-context examples  $N_{example}$ , belief weights across modalities  $W$ 
2: Output: final answer  $ans_{final}$ 
3:  $n = 1$ 
4:  $ans_{final} = ans_0$ 
5: while  $n < N_{example}$  do
6:   if  $mc_0$  is maximum then
7:     return  $ans_{final}$ 
8:   else
9:      $m = M_{tgt}.pop()$ 
10:    if  $m = \emptyset$  then
11:      break
12:     $P_{MOD}^+, R^+ = [], []$ 
13:    for each  $p_{mod}$  from  $P_{MOD}$  do
14:       $p_{mod}^+ = m.mod + p_{mod}$ 
15:       $r_{mod}^+ = LM(q_{tgt}, p_{mod}^+)$ 
16:       $P_{MOD}^+.append(p_{mod}^+)$ 
17:       $R^+.append(r_{mod}^+)$ 
18:       $ans^+, mc^+ = get\_answer(R^+, W)$ 
19:      if  $(mc^+ > mc_0) \vee (ans^+ \neq ans_{final})$  then
20:         $P_{MOD} = P_{MOD}^+$ 
21:         $R = R^+$ 
22:         $mc_0 = mc^+$ 
23:         $ans_{final} = ans^+$ 
24:         $n = n + 1$ 
25: return  $ans_{final}$ 
```

---

---

**Algorithm 2**  $get\_answer$  algorithm.

---

```
1: Input: responses of the target question over modalities  $R$ , belief weights across modalities  $W$ 
2: Output: final answer  $final\_ans$ , consistency score of the final answer  $final\_cs$ 
3:  $unique\_answers = set()$ 
4:  $cs, sc = dict(), dict()$ 
5: for each  $mod$  from  $R.modalities$  do
6:    $Ans_{topk} = extract\_answers(R[mod])$ 
7:   for each  $ans$  from  $unq\_ans$  do
8:      $sc[mod, ans] = self\_consistency(ans, Ans_{topk})$ 
9:    $unique\_answers.add(unq\_ans)$ 
10: for each  $ans$  from  $unique\_answers$  do
11:    $cs[ans] = 0.0$ 
12:   for each  $mod$  from  $R.modalities$  do
13:      $cs[ans] += W_{mod} * SC[mod, ans]$ 
14:  $final\_ans, final\_cs = get\_highest\_score\_answer(cs)$ 
15: return  $final\_ans, final\_cs$ 
```

---

the final answer (lines 6-7). Otherwise, the algorithm incorporates each mutation as an in-context example in prompts (lines 9-18). Subsequently, LLM responses using the mutation across modalities are obtained, and the consistency is calculated based on these responses using the  $get\_answer()$  algorithm (line 18). Algorithm 2 elucidates the  $get\_answer()$  function. This function first gathers the self-consistency score for each unique answer across modalities (lines 7-8). Then it computes the consistency score, as defined in Equation 3 (lines 11-13). Finally, it returns the highest consistency score and its corresponding answer. Returning to Algorithm 1, the algorithm then evaluates the difference in the consistency scores before and after the addition of the mutation to the modality prompts. Mutations are selected as in-

Table 1: Prompts for large language model over different modalities.

Modality	Prompt
CoT	“ <i>{QUESTION}</i> Let’s think step by step and end your response with ‘the answer is {answer}’”
Code	“ <i>I want you to act like a mathematician. I will type mathematical question and you will respond with a function named with ‘func’ in python code that returns the answer of the question. the function should have no arguments. I want you to answer only with the final python code and nothing else. Do not write explanations: {QUESTION}</i> ”
Equation	“ <i>{QUESTION}</i> Write a wolframalpha mathematical equation with no explanations and no units to the numbers in the equation. Generate the answer format starting with ‘Answer =’”

context examples in prompts if there is an increase in modal-consistency following their addition, or if the LLM generates a different answer than the previous one (line 19). Upon selection of a mutation, prompts across modalities, answer for the target problem, and their respective consistency values are updated to facilitate the search for additional mutations (lines 20-24). This process iterates until either the number of selected mutations or the consistency reaches its maximum, or until no mutations remain (lines 5, 6 and 10, respectively). The final answer for the target problem, obtained through the iterative optimization process, is returned.

## 4 Experiments

### 4.1 Experimental setup

**Dataset.** We assess the performance of APMath on the following widely used public arithmetic reasoning benchmarks: the Math Word Problem Repository MultiArith (Roy and Roth, 2016), AS-Div (Miao et al., 2020), SVAMP (Patel et al., 2021), and GSM8k (Cobbe et al., 2021), a recently published benchmark of grade-school-math problems.

**Large Language Models.** We evaluate the APMath using GPT-3.5 (OpenAI, 2023). It is a transformer-based architecture with 175 billion parameters. Specifically, we utilize the public engine *gpt-3.5-turbo* from the OpenAI models.

**Prompts over modalities.** Inspired by the prompts presented in (Akin, 2022), we manually crafted the prompts detailed in Table 1 over three modalities. The first and second columns of Table 1 represent the modality type and the corresponding prompt text, respectively, with the placeholder “*{QUESTION}*” used to represent the input question. The goal of the prompt design is to segregate the explanation from the corresponding final answer, thereby facilitating the automatic parsing of

the answer from the LLM responses. The prompt for the CoT modality generates a reasoning path. The phrase within the prompt, “*Let’s think step by step*”, facilitates step-by-step thinking before providing an answer. The instruction “*end your response with ‘the answer is {answer}’*” prompts the LLM to conclude its response with the phrase “*the answer is {answer}*”, where *{answer}* represents the ultimate answer to the question. For the code and equation modalities, we obtain the generated executable Python code and WolframAlpha mathematical equation from the LLM with no additional explanation provided. We then execute the code and equation using the Python command and WolframAlpha API (WolframAlpha, 2023), respectively. Finally, we consider the returned value as the answer for the respective modality.

**Evaluation Metric.** We compare accuracy of LLM responses, defined as the ratio of the number of correctly predicted answers to the number of arithmetic math questions in the test datasets.

**Baselines.** We evaluate APMath by assessing its accuracy on the datasets compared to baseline methods. It aims to demonstrate APMath’s ability to generate relevant examples in zero-shot contexts. We also show the effectiveness of consistency across modalities by comparing it to APMath’s performance without this feature. Furthermore, we compare an existing state-of-the-art method for retrieving in-context examples with APMath to highlight the effectiveness of mutations for in-context examples over those from external datasets.

- **Zero-shot with a specific Modality:** This prompt solely uses a specific modality without any in-context examples. In this experiment, we utilize the CoT, Code, and Equation modalities, denoted as CoTPrompt, CodePrompt and EqnPrompt, respectively. For each modality, the final answer is determined by selecting the most frequently occurring answers from the top three responses (Wang et al., 2023b). The prompt used is identical to the corresponding modality prompt in Table 1.
- **Majority voting of answers across modalities (MajVotModals):** This method determines the final answer by majority voting of answers across the three modalities.
- **APMath w/o consistency over modalities (APMath w/o modalities):** For a specific modality, we employ a subset of mutations that improve the self-consistency, as defined in

Equation 2, of the LLM’s top- $K$  responses to a target problem, utilizing these as in-context examples. Specifically, we extract the top-3 responses of the LLM for the CoT modality.

- **In-context example retrieval method (Auto-CoT):** AutoCoT (Zhang et al., 2023) is implemented. It clusters the embedding vectors of retrieval examples using SentenceBERT (Reimers and Gurevych, 2019) into  $K$  clusters. Next, for each clustered examples, the embedding vector of the target question is compared with them, and the closest example is selected. These  $K$  examples are then utilized as in-context examples. In this experiment, we construct 8 clusters, providing 8 in-context examples for each target question.

**Implementation Details and Hardware Environment.** We utilized the OpenAI API to run GPT-3.5. We applied temperature with  $T = 0.7$  and truncated at the top-3 responses. Due to the limited resources, we generated 20 mutated questions for each original question and obtained their LLM responses to identify relevant in-context examples. In addition, The modality belief weight for the aggregation across the modalities is set to 0.4 for CoT modality and 0.3 for both code and equation modalities, reflecting the more important role of CoT modality in reasoning and logical flow (Wei et al., 2022; Chowdhery et al., 2023). All experiments were conducted on a Ubuntu 14.04 server with three Intel Xeon E5-2660 v3 CPUs @2.60GHz, eight Nvidia 1080Ti GPUs, and 500GB of RAM.

## 4.2 Results

**Comparison of APMath with Zero-Shot Baselines.** We first report the experimental results of APMath to demonstrate the effectiveness of APMath in the zero-shot setting. Table 2 shows the accuracies of the LLM on different math problem datasets. The first and second columns denotes the names of the datasets and the number of math problems for each dataset used in the experiment, respectively. Columns 3-6 show the accuracies of LLMs achieved using baseline methods. The last column shows the LLM accuracy attained with APMath. The results show that *APMath outperforms all the baselines on all four datasets*. APMath achieved better accuracy than the baselines by 1% to 7.3% on ASDiv, 3.8% to 15.4% on SVAMP, 4% to 22.4% on GSM8k, and 1% to 62.1% on Multi-Arith. This result indicates that APMath exhibits

Table 2: Accuracies of GPT-3.5 across different math problem datasets in the zero-shot setting.

Dataset	#Data	CoTPrompt [%]	CodePrompt [%]	EqnPrompt [%]	MajVotModals [%]	APMath [%]
ASDiv	1218	95.2	95.0	89.8	96.1	<b>97.1</b>
SVAMP	1000	79.5	79.7	71.6	83.2	<b>87.0</b>
GSM8k	1319	77.5	69.4	61.4	79.8	<b>83.8</b>
MultiArith	600	96.0	98.3	37.2	97.0	<b>99.3</b>

Table 3: Accuracies on GPT-3.5 with in-context examples across different math problem datasets.

Dataset	AutoCoT [%]	APMath w/o modalities [%]	APMath [%]
ASDiv	<b>97.4</b>	96.1	97.1
SVAMP	82.6	85.4	<b>87.0</b>
GSM8k	81.4	<b>86.9</b>	83.8
MultiArith	97.2	96.8	<b>99.3</b>

Table 4: Results of manual study to evaluate the correctness of GPT-3.5 responses to mutated math problems. The number of mutations and correct mutations used as in-context examples are denoted as MutUsed and CorrectMutUsed, respectively.

Dataset	#MutUsed	#CorrectMutUsed
ASDiv	98	89
SVAMP	180	132
GSM8k	180	149
MultiArith	24	24

significant improvements in enhancing accuracy across various mathematical problem datasets in zero-shot setting. It suggests that the additional contextual processing enabled by APMath’s mutation selection phase is crucial for handling more mathematical queries that may not be as effectively addressed through standard zero-shot methodologies. In Appendix A.1, we provide samples of APMath generated in-context examples for each of the four datasets.

**Effectiveness of Mutation as In-context Examples.** The comparison of LLM accuracies using APMath with the other baselines are shown in Table 3. The first column lists the names of the mathematical problem datasets. The second and third columns report the accuracies achieved by APMath without consistency over modalities and by the in-context example retrieval method, AutoCoT, respectively. Comparing to AutoCoT, APMath improves accuracy by 4.4%, 2.4%, and 2.1% for the SVAMP, GSM8k, and MultiArith, respectively. For the ASDiv dataset, APMath exhibits a slight decrease in accuracy by 0.3%. Overall, *APMath’s in-context examples is more effective than those produced by current retrieval-based methods in the absence of external retrieval datasets and models.*

**Effectiveness of Combination Across Modalities.** Comparing to APMath w/o modalities (third col-

umn in Table 3), APMath exhibits a decreased accuracy by 3.1% for the GSM8k dataset. Out of the 79 problems correctly answered by APMath but incorrectly by APMath w/o modalities, APMath fails to find in-context examples for 53 problems, while APMath w/o modalities succeeds. For the remaining 26 problems, the discrepancies are attributed to the randomness induced by the temperature setting of the LLM. However, APMath improves accuracy by 1%, 1.6%, and 2.5% for the ASDiv, SVAMP, and MultiArith datasets, respectively. This result suggests that the idea of increasing consistency over various modalities is effective. Additionally, we conduct a manual study to assess the consistency across modalities as a metric of evaluating the correctness of LLM responses to mutated problems used as in-context examples. The correctness is determined by manually verifying the LLM responses with the expected outcomes derived from the ground truth data, ensuring that the model accurately interprets and solves the mutated problems. The results of this manual study are presented in Table 4. The second column shows the number of mutations used as in-context examples for target problems in each dataset. The third column shows the number of correct mutations used as in-context examples for the target math problems. Table 4 shows that APMath provides accurate LLM responses to the mutated math problems, with accuracy ranging from 73.3% (132 out of 180) for SVAMP to 100% (24 out of 24) for MultiArith. The variance in accuracy across datasets is attributed to the different complexities of the problems in each dataset. Complexity arises from the structure of the problems, the steps required to solve them, and the mathematical concepts involved. Consequently, more complex problems present greater challenges for LLMs. Despite this variance, these results suggest that consistency across modalities plays a crucial role in ensuring the correctness of the responses, thereby enhancing the effectiveness of evaluating LLM responses across different modalities.

## 5 Related Work

**In-context Learning.** There has been recent advancement in in-context learning. Saunshi *et al.* (Saunshi *et al.*, 2021) suggests that downstream tasks can be solved linearly by conditioning on a prompting words following an input text. Xie *et al.* (Xie *et al.*, 2022) suggests that the language model can infer in-context shared latent concept among examples in a prompt. Levine *et al.* (Levine *et al.*, 2022) establishes that the information within in-context examples gives more improvements. In addition, Wei *et al.* (Wei *et al.*, 2022) has implemented manually hand-crafted the few-shot examples for improving quality of CoT explanation that LLM generates. However, to tackle the need for manually hand-crafted few-shot examples, recent studies have developed a retriever to select analogy examples for demonstration (Zhang *et al.*, 2023; Rubin *et al.*, 2022; Su *et al.*, 2023; Wang *et al.*, 2023a; Luo *et al.*, 2023). These studies differ from ours in that they require a substantial amount of fully annotated data to train models and retrieve in-context examples, whereas APMath generates in-context examples automatically through mutation and consistency optimization.

**Consistency in LLM.** Prior research has suggested that language models may experience inconsistency in natural language conversation (Adiwardana *et al.*, 2020), and factual knowledge extraction (Elazar *et al.*, 2021). Wang *et al.* (Wang *et al.*, 2023b) utilize answer consistency across various reasoning paths within top- $K$  responses to enhance accuracy. Camburu *et al.* (Camburu *et al.*, 2020) introduced an adversarial framework aimed at verifying language models’ coherence in generating natural language explanations. Moreover, recent studies have tackled the issue of inconsistency in the long-form creative writing generated by LLMs through techniques like prompt chaining (Mirowski *et al.*, 2022) and editing to rectify long-range factual inconsistencies within story passages (Yang *et al.*, 2022). In this paper, we concentrate on quantifying the consistency of answers across various modalities and leveraging this metric to estimate the accuracy of LLM responses by incorporating mutations as in-context examples.

**Prompt Optimization.** Our research also intersects with prompt optimization. Research work improves hard prompts via an iterative local edit and gradient-free search (Prasad *et al.*, 2023) or gradient-based optimization (Sun *et al.*, 2023).

Yang *et al.* (Yang *et al.*, 2023) describes the optimization task in natural language and feeds it to the large language model as a prompt and then generates new prompt. Compared with them, APMath automatically optimizes in-context examples across modalities, rather than relying on a single modality to improve the robustness of evaluation of LLM behavior. In addition, prior research work have optimized a small continuous vector for downstream tasks, leaving LLM parameters frozen (Li and Liang, 2021; Zhong *et al.*, 2021; Sun *et al.*, 2022b,a; Chen *et al.*, 2023). Diao *et al.* (Diao *et al.*, 2023) applies a policy gradient to estimate the gradients of the parameters of the categorical distribution of each discrete prompt. However, they are limited to the white-box setting, requiring accessing the parameters of a pre-trained model while APMath is in black-box optimization by the consistency of LLM responses across modalities. In addition, Mishra *et al.* (Khashabi *et al.*, 2022) studies advantages of prompt tuning, but it requires manual efforts. Zhou *et al.* (Zhou *et al.*, 2023) automate the generation of instructions and select the most suitable instruction based on computed evaluation scores. However, their focus lies on instruction induction tasks rather than math problem-solving tasks.

## 6 Conclusions

This paper introduces APMath, a novel tool that automates the generation of relevant in-context examples to enhance the arithmetic problem-solving capabilities of LLMs. APMath automates the mutation of target math problems, generating variants that use the same solving algorithm. It also employs a consistency check across various LLM response modalities to evaluate answer confidence and estimate accuracy for both original and altered problems. Additionally, it identifies mutations for in-context examples that improve response consistency. Evaluations show APMath significantly enhances answer accuracy for mutated math problems, demonstrating efficient generation of relevant in-context examples without manual annotations or external datasets. The proposed consistency check method increases LLM response confidence and correctness.

## 7 Limitation

This work has several limitations. First, APMath relies on the random mutation of the identified numerical values. The effectiveness of such random muta-



tions is limited in generating realistic mutated math problems. We leave it as our future work to explore context-aware mutation mutation techniques for the target math problems. Second, APMath demonstrates evident advantages only for solving arithmetic math problems through mutation while maintaining the same reasoning algorithm. Future work will aim to extend generating mutations for a broader range of reasoning problems.

## References

Daniel Adiwardana, Minh-Thang Luong, David R. So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, and Quoc V. Le. 2020. [Towards a human-like open-domain chatbot](#). *CoRR*, abs/2001.09977.

Fatih Kadir Akin. 2022. [Awesome chatgpt prompts](#). <https://github.com/f/awesome-chatgpt-prompts>.

Oana-Maria Camburu, Brendan Shillingford, Pasquale Minervini, Thomas Lukasiewicz, and Phil Blunsom. 2020. [Make up your mind! adversarial generation of inconsistent natural language explanations](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4157–4165, Online. Association for Computational Linguistics.

Lichang Chen, Jiu-hai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. 2023. [Instructzero: Efficient instruction optimization for black-box large language models](#). *CoRR*, abs/2306.03082.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayanan Pilla, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. [Palm: Scaling language modeling with pathways](#). *J. Mach. Learn. Res.*, 24:240:1–240:113.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias

Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. [Training verifiers to solve math word problems](#). *CoRR*, abs/2110.14168.

Shizhe Diao, Zhichao Huang, Ruijia Xu, Xuechun Li, Yong Lin, Xiao Zhou, and Tong Zhang. 2023. [Black-box prompt learning for pre-trained language models](#). *Trans. Mach. Learn. Res.*, 2023.

Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard H. Hovy, Hinrich Schütze, and Yoav Goldberg. 2021. [Measuring and improving consistency in pretrained language models](#). *Trans. Assoc. Comput. Linguistics*, 9:1012–1031.

Shima Imani, Liang Du, and Harsh Shrivastava. 2023. [Mathprompter: Mathematical reasoning using large language models](#). In *Proceedings of the The 61st Annual Meeting of the Association for Computational Linguistics: Industry Track, ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 37–42. Association for Computational Linguistics.

Daniel Khashabi, Chitta Baral, Yejin Choi, and Hananeh Hajishirzi. 2022. [Reframing instructional prompts to gptk’s language](#). In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 589–612. Association for Computational Linguistics.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. [Large language models are zero-shot reasoners](#). In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Yoav Levine, Noam Wies, Daniel Jannai, Dan Navon, Yedid Hoshen, and Amnon Shashua. 2022. [The inductive bias of in-context learning: Rethinking pre-training example design](#). In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Xiang Lisa Li and Percy Liang. 2021. [Prefix-tuning: Optimizing continuous prompts for generation](#). In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4582–4597. Association for Computational Linguistics.

Xiao Lin and Devi Parikh. 2015. [Don’t just listen, use your imagination: Leveraging visual common sense for non-visual tasks](#). In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 2984–2993. IEEE Computer Society.

Man Luo, Xin Xu, Zhuyun Dai, Panupong Pasupat, Seyed Mehran Kazemi, Chitta Baral, Vaiva Imbrasaitė, and Vincent Y. Zhao. 2023. [Dr.icl: Demonstration-retrieved in-context learning](#). *CoRR*, abs/2305.14128.

755	Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su.	Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena	813
756	2020. <a href="#">A diverse corpus for evaluating and developing</a>	Gribovskaya, Domenic Donato, Angeliki Lazaridou,	814
757	<a href="#">English math word problem solvers</a> . In <i>Proceedings</i>	Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsim-	815
758	<i>of the 58th Annual Meeting of the Association for</i>	poukelli, Nikolai Grigorev, Doug Fritz, Thibault Sot-	816
759	<i>Computational Linguistics</i> , pages 975–984, Online.	tiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong,	817
760	Association for Computational Linguistics.	Daniel Toyama, Cyprien de Masson d’Autume, Yujia	818
761	Bonan Min, Hayley Ross, Elior Sulem, Amir	Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin,	819
762	Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz,	Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris	820
763	Eneko Agirre, Ilana Heintz, and Dan Roth. 2024.	Jones, James Bradbury, Matthew J. Johnson, Blake A.	821
764	<a href="#">Recent advances in natural language processing via</a>	Hechtman, Laura Weidinger, Jason Gabriel, William	822
765	<a href="#">large pre-trained language models: A survey</a> . <i>ACM</i>	Isaac, Edward Lockhart, Simon Osindero, Laura	823
766	<i>Comput. Surv.</i> , 56(2):30:1–30:40.	Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub,	824
767	Piotr Mirowski, Kory W. Mathewson, Jaylen Pittman,	Jeff Stanway, Lorraine Bennett, Demis Hassabis, Ko-	825
768	and Richard Evans. 2022. <a href="#">Co-writing screen-</a>	ray Kavukcuoglu, and Geoffrey Irving. 2021. <a href="#">Scaling</a>	826
769	<a href="#">plays and theatre scripts with language models:</a>	<a href="#">language models: Methods, analysis &amp; insights from</a>	827
770	<a href="#">An evaluation by industry professionals</a> . <i>CoRR</i> ,	<a href="#">training gopher</a> . <i>CoRR</i> , abs/2112.11446.	828
771	abs/2209.14958.	Nils Reimers and Iryna Gurevych. 2019. <a href="#">Sentence-bert:</a>	829
772	OpenAI. 2023. <a href="#">Introducing chatgpt</a> . <a href="https://openai.com/blog/chatgpt">https://openai.com/blog/chatgpt</a> .	<a href="#">Sentence embeddings using siamese bert-networks</a> .	830
773		<i>CoRR</i> , abs/1908.10084.	831
774	Arkil Patel, Satwik Bhattamishra, and Navin Goyal.	Subhro Roy and Dan Roth. 2016. <a href="#">Solving general arith-</a>	832
775	2021. <a href="#">Are NLP models really able to solve simple</a>	<a href="#">metic word problems</a> . <i>CoRR</i> , abs/1608.01413.	833
776	<a href="#">math word problems?</a> In <i>Proceedings of the 2021</i>	Ohad Rubin, Jonathan Herzig, and Jonathan Berant.	834
777	<i>Conference of the North American Chapter of the</i>	2022. <a href="#">Learning to retrieve prompts for in-context</a>	835
778	<i>Association for Computational Linguistics: Human</i>	<a href="#">learning</a> . In <i>Proceedings of the 2022 Conference of</i>	836
779	<i>Language Technologies, NAACL-HLT 2021, Online,</i>	<i>the North American Chapter of the Association for</i>	837
780	<i>June 6-11, 2021</i> , pages 2080–2094. Association for	<i>Computational Linguistics: Human Language Tech-</i>	838
781	Computational Linguistics.	<i>nologies, NAACL 2022, Seattle, WA, United States,</i>	839
782	Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit	<i>July 10-15, 2022</i> , pages 2655–2671. Association for	840
783	Bansal. 2023. <a href="#">Grips: Gradient-free, edit-based in-</a>	Computational Linguistics.	841
784	<a href="#">struction search for prompting large language models</a> .	Nikunj Saunshi, Sadhika Malladi, and Sanjeev Arora.	842
785	In <i>Proceedings of the 17th Conference of the Euro-</i>	2021. <a href="#">A mathematical exploration of why language</a>	843
786	<i>pean Chapter of the Association for Computational</i>	<a href="#">models help solve downstream tasks</a> . In <i>9th Inter-</i>	844
787	<i>Linguistics, EACL 2023, Dubrovnik, Croatia, May</i>	<i>national Conference on Learning Representations,</i>	845
788	<i>2-6, 2023</i> , pages 3827–3846. Association for Com-	<i>ICLR 2021, Virtual Event, Austria, May 3-7, 2021</i> .	846
789	putational Linguistics.	OpenReview.net.	847
790	Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya	Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao,	848
791	Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sas-	Abu Awal Md Shoeb, Abubakar Abid, Adam	849
792	try, Amanda Askell, Pamela Mishkin, Jack Clark,	Fisch, Adam R. Brown, Adam Santoro, Aditya	850
793	Gretchen Krueger, and Ilya Sutskever. 2021. <a href="#">Learn-</a>	Gupta, Adrià Garriga-Alonso, Agnieszka Kluska,	851
794	<a href="#">ing transferable visual models from natural language</a>	Aitor Lewkowycz, Akshat Agarwal, Alethea Power,	852
795	<a href="#">supervision</a> . In <i>Proceedings of the 38th International</i>	Alex Ray, Alex Warstadt, Alexander W. Kocurek,	853
796	<i>Conference on Machine Learning, ICML 2021, 18-24</i>	Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Par-	854
797	<i>July 2021, Virtual Event</i> , volume 139 of <i>Proceedings</i>	ish, Allen Nie, Aman Hussain, Amanda Askell,	855
798	<i>of Machine Learning Research</i> , pages 8748–8763.	Amanda Dsouza, Ameet Rahane, Anantharaman S.	856
799	PMLR.	Iyer, Anders Andreassen, Andrea Santilli, Andreas	857
800	Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie	Stuhlmüller, Andrew M. Dai, Andrew La, Andrew K.	858
801	Millican, Jordan Hoffmann, H. Francis Song, John	Lampinen, Andy Zou, Angela Jiang, Angelica Chen,	859
802	Aslanides, Sarah Henderson, Roman Ring, Susan-	Anh Vuong, Animesh Gupta, Anna Gottardi, Anto-	860
803	nah Young, Eliza Rutherford, Tom Hennigan, Jac-	nio Norelli, Anu Venkatesh, Arash Gholamidavoodi,	861
804	cob Menick, Albin Cassirer, Richard Powell, George	Arfa Tabassum, Arul Menezes, Arun Kirubarajan,	862
805	van den Driessche, Lisa Anne Hendricks, Mari-	Asher Mullokandov, Ashish Sabharwal, Austin Her-	863
806	beth Rauh, Po-Sen Huang, Amelia Glaese, Jo-	rick, Avia Efrat, Aykut Erdem, Ayla Karakas, and	864
807	hannes Welbl, Sumanth Dathathri, Saffron Huang,	et al. 2022. <a href="#">Beyond the imitation game: Quantifying</a>	865
808	Jonathan Uesato, John Mellor, Irina Higgins, Antonia	<a href="#">and extrapolating the capabilities of language models</a> .	866
809	Creswell, Nat McAleese, Amy Wu, Erich Elsen, Sid-	<i>CoRR</i> , abs/2206.04615.	867
810	dhand M. Jayakumar, Elena Buchatskaya, David Bud-	Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi,	868
811	den, Esme Sutherland, Karen Simonyan, Michela Pa-	Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf,	869
812	ganini, Laurent Sifre, Lena Martens, Xiang Lorraine	Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023.	870
		<a href="#">Selective annotation makes language models better</a>	871

872	few-shot learners. In <i>The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023</i> . OpenReview.net.	2023. Large language models as optimizers. <i>CoRR</i> , abs/2309.03409.	928
873			929
874			
875	Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. 2020. VL-BERT: pre-training of generic visual-linguistic representations. In <i>8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020</i> . OpenReview.net.	Kevin Yang, Yuandong Tian, Nanyun Peng, and Dan Klein. 2022. Re3: Generating longer stories with recursive reprompting and revision. In <i>Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022</i> , pages 4393–4479. Association for Computational Linguistics.	930
876			931
877			932
878			933
879			934
880			935
881	Hong Sun, Xue Li, Yinchuan Xu, Youkow Homma, Qi Cao, Min Wu, Jian Jiao, and Denis Charles. 2023. Autohint: Automatic prompt optimization with hint generation. <i>CoRR</i> , abs/2307.07415.	Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2023. Automatic chain of thought prompting in large language models. In <i>The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023</i> . OpenReview.net.	936
882			937
883			938
884			939
885	Tianxiang Sun, Zhengfu He, Hong Qian, Yunhua Zhou, Xuanjing Huang, and Xipeng Qiu. 2022a. Bbtv2: Towards a gradient-free future with large language models. In <i>Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022</i> , pages 3916–3930. Association for Computational Linguistics.	Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual probing is [MASK]: learning vs. learning to recall. In <i>Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021</i> , pages 5017–5033. Association for Computational Linguistics.	940
886			941
887			942
888			943
889			944
890			945
891			946
892			947
893	Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022b. Black-box tuning for language-model-as-a-service. In <i>International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA</i> , volume 162 of <i>Proceedings of Machine Learning Research</i> , pages 20841–20855. PMLR.	Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In <i>The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023</i> . OpenReview.net.	948
894			949
895			950
896			951
897			952
898			953
899			954
900	Liang Wang, Nan Yang, and Furu Wei. 2023a. Learning to retrieve in-context examples for large language models. <i>CoRR</i> , abs/2307.07164.		955
901			956
902			957
903	Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In <i>The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023</i> . OpenReview.net.	<b>A Appendix</b>	958
904			959
905			960
906			961
907			962
908			963
909			964
910	Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In <i>Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022</i> .	<b>A.1 Qualitative Analysis of APMath</b>	965
911			966
912			967
913			968
914			969
915			970
916			971
917			972
918	WolframAlpha. 2023. Wolframalpha apis. <a href="https://products.wolframalpha.com/api">https://products.wolframalpha.com/api</a> .	Table 5 to 8 presents four illustrative examples from the datasets employed in our experiments to evaluate APMath. In each table, the first row displays in-context examples generated by APMath. The second row shows the target mathematical problem to be solved. Furthermore, the first and second columns in the third row present GPT-3.5’s responses to the target problem across modalities, using the zero-shot setting and APMath, respectively.	963
919			964
920	Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2022. An explanation of in-context learning as implicit bayesian inference. In <i>The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022</i> . OpenReview.net.	These examples demonstrate that in the zero-shot setting, GPT-3.5’s reasoning is incorrect in at least one modality across all datasets. Conversely, the in-context examples generated by APMath correctly instruct GPT-3.5 to solve the target problem, resulting in accurate answers by following the same reasoning implemented in the in-context examples.	965
921			966
922			967
923			968
924			969
925			970
926	Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen.		971
927			972
			973
			974
			975
			976

Table 5: APMath generated in-context examples for ASDiv.

**(In-context 1)**

**Q:** Olivia had 69 math problems to complete . She spent about 15 minutes on each problem and then she spent 42 minutes checking her answers . How many minutes did she spend total?

**[CoT] A:** To find the total time Olivia spent, we need to calculate the time she spent on solving the math problems and the time she spent checking her answers separately. First, let's calculate the time Olivia spent on solving the math problems. Since she spent about 15 minutes on each problem, we can multiply the number of problems by 15:  $69 \text{ problems} * 15 \text{ minutes/problem} = 1035 \text{ minutes}$  Next, let's calculate the time Olivia spent checking her answers, which is given as 42 minutes. Now, let's add the time spent on solving the problems and checking the answers together:  $1035 \text{ minutes} + 42 \text{ minutes} = 1077 \text{ minutes}$  Therefore, Olivia spent a total of 1077 minutes. The answer is 1077 minutes.

**[Code] A:** `def func(): return 69 * 15 + 42`

**[Eqn] A:** `Answer = 69*15 + 42`

**(In-context 2)**

**Q:** Olivia had 52 math problems to complete . She spent about 10 minutes on each problem and then she spent 50 minutes checking her answers . How many minutes did she spend total?

**[CoT] A:** To find the total number of minutes Olivia spent, we need to add the time she spent solving the math problems to the time she spent checking her answers. First, we need to calculate the time Olivia spent solving the math problems. Since she spent about 10 minutes on each problem, we can multiply the number of problems by 10:  $52 \text{ problems} * 10 \text{ minutes/problem} = 520 \text{ minutes}$ . Next, we need to calculate the time Olivia spent checking her answers, which is given as 50 minutes. Finally, we can find the total time by adding the time spent solving the problems to the time spent checking the answers:  $520 \text{ minutes} + 50 \text{ minutes} = 570 \text{ minutes}$ . Therefore, the answer is 570 minutes.

**[Code] A:** `def func(): return 52 * 10 + 50`

**[Eqn] A:** `Answer = 52*10 + 50`

---

**Target Q:** Olivia had 7 math problems to complete. She spent about 4 minutes on each problem and then she spent 3 minutes checking her answers. How many minutes did she spend total?

---

**Zero-shot**

**Target [CoT]A:** To find the total time Olivia spent, we need to calculate the time she spent on solving the math problems and the time she spent checking her answers separately. First, let's calculate the time Olivia spent on solving the math problems. Since she spent about 4 minutes on each problem, we can multiply 4 by the number of problems she had:  $4 \text{ minutes/problem} * 7 \text{ problems} = 28 \text{ minutes}$ . Next, let's calculate the time Olivia spent checking her answers. She spent 3 minutes checking her answers after solving each problem, so we can multiply 3 by the number of problems she had:  $3 \text{ minutes/problem} * 7 \text{ problems} = 21 \text{ minutes}$ . Finally, to find the total time Olivia spent, we add the time spent on solving the problems to the time spent checking her answers:  $28 \text{ minutes} + 21 \text{ minutes} = 49 \text{ minutes}$ . Therefore, the answer is 49 minutes.

**Target [Code]A:** `def func(): return 7 * (4 + 3)`

**Target [Eqn]A:** `Answer = 7 * 4 + 7 * 3`

**After APMath**

**Target [CoT]A:** To find the total time Olivia spent, we need to calculate the time she spent on solving the math problems and the time she spent checking her answers separately. First, let's calculate the time Olivia spent on solving the math problems. Since she spent about 4 minutes on each problem, we can multiply the number of problems by 4:  $7 \text{ problems} * 4 \text{ minutes/problem} = 28 \text{ minutes}$  Next, let's calculate the time Olivia spent checking her answers, which is given as 3 minutes. Now, let's add the time spent on solving the problems and checking the answers together:  $28 \text{ minutes} + 3 \text{ minutes} = 31 \text{ minutes}$  Therefore, Olivia spent a total of 31 minutes. The answer is 31 minutes.

**Target [Code]A:** `def func(): return 7 * 4 + 3`

**Target [Eqn]A:** `Answer = 7*4 + 3`

---

Table 6: APMath generated in-context examples for SVAMP.

**(In-context 1)**

**Q:** Edward spent \$8 . Then he received \$12 from his friend . Now he has \$19 . How much did Edward have before he spent his money?

**[CoT]A:** Step 1: Edward spent \$8. Step 2: He received \$12 from his friend. Step 3: Now he has \$19. Step 4: To find out how much Edward had before he spent his money, we can use the equation:  $x - 8 + 12 = 19$ , where  $x$  represents the amount he had before spending his money. Step 5: Simplifying the equation, we get  $x + 4 = 19$ . Step 6: Subtracting 4 from both sides, we get  $x = 15$ . Step 7: Therefore, Edward had \$15 before he spent his money. The answer is \$15.

**[Code]A:** def func(): return (19 - 12 + 8)

**[Eqn]A:** Answer =  $x - 8 + 12 = 19$

**(In-context 2)**

**Q:** Olivia had 52 math problems to complete . She spent about 10 minutes on each problem and then she spent 50 minutes checking her answers . How many minutes did she spend total?

**[CoT]A:** Step 1: Edward spent \$65. Step 2: He received \$39 from his friend. Step 3: Now he has \$67. Step 4: To find out how much Edward had before he spent his money, we can use the equation:  $x - 65 + 39 = 67$ , where  $x$  represents the amount he had before spending his money. Step 5: Simplifying the equation, we get  $x - 26 = 67$ . Step 6: Adding 26 to both sides, we get  $x = 93$ . Step 7: Therefore, the answer is 93.

**[Code]A:** def func(): return (67 - 39) + 65

**[Eqn]A:** Answer =  $x - 65 + 39 = 67$

---

**Target Q:** Edward spent \$17. Then he received \$10 from his friend. Now he has \$7. How much did Edward have before he spent his money?

---

**Zero-shot**

**Target [CoT]A:** Step 1: Edward spent \$17. Step 2: He received \$10 from his friend. Step 3: Now he has \$7. Step 4: To find out how much Edward had before he spent his money, we need to add the \$17 he spent and the \$7 he has now. Step 5:  $\$17 + \$7 = \$24$  Step 6: Therefore, the answer is \$24.

**Target [Code]A:** def func(): return  $17 + 10 - 7$

**Target [Eqn]A:** Answer =  $x - 17 + 10 = 7$

**After APMath**

**Target [CoT]A:** Step 1: Edward spent \$17. Step 2: He received \$10 from his friend. Step 3: Now he has \$7. Step 4: To find out how much Edward had before he spent his money, we can use the equation:  $x - 17 + 10 = 7$ , where  $x$  represents the amount he had before spending his money. Step 5: Simplifying the equation, we get  $x - 7 = 7$ . Step 6: Adding 7 to both sides, we get  $x = 14$ . Step 7: Therefore, the answer is 14.

**Target [Code]A:** def func(): return  $(7 - 10) + 17$

**Target [Eqn]A:** Answer =  $x - 17 + 10 = 7$

---

Table 7: APMath generated in-context examples for MultiArith.

<b>(In-context 1)</b>	
<p><b>Q:</b> Gwen and her friends were recycling paper for their class . For every 1 pounds they recycled they earned 38 point . If Gwen recycled 15 pounds and her friends recycled 36 pounds, how many points did they earn?</p> <p><b>[CoT]A:</b> First, we need to calculate the number of points Gwen earned for recycling 15 pounds. Since they earned 38 points for every 1 pound, Gwen earned <math>15 * 38 = 570</math> points. Next, we need to calculate the number of points Gwen’s friends earned for recycling 36 pounds. Following the same logic, her friends earned <math>36 * 38 = 1368</math> points. Finally, we need to find the total number of points earned by Gwen and her friends. Adding Gwen’s points to her friends’ points, we get <math>570 + 1368 = 1938</math> points. Therefore, the answer is 1938.</p> <p><b>[Code]A:</b> <code>def func(): gwen_pounds = 15 friends_pounds = 36 points_per_pound = 38 total_points = (gwen_pounds + friends_pounds) * points_per_pound return total_points</code></p> <p><b>[Eqn]A:</b> <math>Answer = 15*38 + 36*38</math></p>	
<p><b>Target Q:</b> Gwen and her friends were recycling paper for their class. For every 3 pounds they recycled they earned one point. If Gwen recycled 5 pounds and her friends recycled 13 pounds, how many points did they earn?</p>	
Zero-shot	After APMath
<p><b>Target [CoT]A:</b> First, we need to calculate how many points Gwen earned. Since she recycled 5 pounds, we divide 5 by 3 to find out how many sets of 3 pounds she recycled. 5 divided by 3 equals 1 with a remainder of 2. So, Gwen earned 1 point. Next, we need to calculate how many points Gwen’s friends earned. Since they recycled 13 pounds, we divide 13 by 3 to find out how many sets of 3 pounds they recycled. 13 divided by 3 equals 4 with a remainder of 1. So, Gwen’s friends earned 4 points. Finally, we add the points Gwen earned (1 point) to the points her friends earned (4 points) to find the total number of points they earned. <math>1 + 4</math> equals 5. Therefore, the answer is 5 points.</p> <p><b>Target [Code]A:</b> <code>def func(): gwen_pounds = 5 friends_pounds = 13 total_pounds = gwen_pounds + friends_pounds points = total_pounds // 3 return points</code></p> <p><b>Target [Eqn]A:</b> <math>Answer = (5 + 13) / 3</math></p>	<p><b>Target [CoT]A:</b> First, we need to find the total pounds of paper recycled by Gwen and her friends. Gwen recycled 5 pounds and her friends recycled 13 pounds, so the total pounds recycled is <math>5 + 13 = 18</math> pounds. Next, we need to determine how many points they earn for recycling 18 pounds. For every 3 pounds recycled, they earn 1 point. So, 18 pounds divided by 3 pounds per point equals 6 points. Therefore, Gwen and her friends earned a total of 6 points for recycling paper. the answer is 6</p> <p><b>Target [Code]A:</b> <code>def func(): gwen_pounds = 5 friends_pounds = 13 total_points = (gwen_pounds + friends_pounds) // 3 return total_points</code></p> <p><b>Target [Eqn]A:</b> <math>Answer = 6</math></p>

Table 8: APMath generated in-context examples for GSM8k.

Zero-shot	After APMath
<p><b>(In-context 1)</b></p> <p><b>Q:</b> George has 46% more pears than bananas . If George has 90 bananas, how many fruits does George have?</p> <p><b>[CoT]A:</b> If George has 90 bananas, then he has <math>46/100 * 90 = \langle\langle 46/100 * 90 = 41.4 \rangle\rangle 41.4</math> more pears than bananas. This means George has <math>90 + 41.4 = \langle\langle 90 + 41.4 = 131.4 \rangle\rangle 131.4</math> pears. In total, George has <math>90 + 131.4 = \langle\langle 90 + 131.4 = 221.4 \rangle\rangle 221.4</math> fruits. The answer is 221.4.</p> <p><b>[Code]A:</b> <code>def func(): bananas = 90 pears = bananas * 1.46 total_fruits = bananas + pears return total_fruits</code></p> <p><b>[Eqn]A:</b> <math>\text{Answer} = 90 + 1.46 * 90</math></p>	
<p><b>(In-context 2)</b></p> <p><b>Q:</b> George has 56% more pears than bananas . If George has 74 bananas, how many fruits does George have?</p> <p><b>[CoT]A:</b> If George has 74 bananas, then he has <math>56/100 * 74 = \langle\langle 56/100 * 74 = 41.44 \rangle\rangle 41.44</math> more pears than bananas. This means George has <math>74 + 41.44 = \langle\langle 74 + 41.44 = 115.44 \rangle\rangle 115.44</math> pears. Therefore, George has a total of <math>74 + 115.44 = \langle\langle 74 + 115.44 = 189.44 \rangle\rangle 189.44</math> fruits. The answer is 189.44.</p> <p><b>[Code]A:</b> <code>def func(): bananas = 74 pears = bananas + (bananas * 56 / 100) return bananas + pears</code></p> <p><b>[Eqn]A:</b> <math>\text{Answer} = 74 + 1.56 * 74</math></p>	
<p><b>Target Q:</b> George has 45% more pears than bananas. If George has 200 bananas, how many fruits does George have?</p>	
<p><b>Target [CoT]A:</b> Step 1: Calculate 45% of 200 bananas. <math>45\% \text{ of } 200 = (45/100) * 200 = 0.45 * 200 = 90</math> Step 2: Add the result from step 1 to the number of bananas. <math>200 + 90 = 290</math> Therefore, George has 290 fruits. The answer is <b>290</b>.</p> <p><b>Target [Code]A:</b> <code>def func(): bananas = 200 pears = bananas + (0.45 * bananas) return bananas + pears</code></p> <p><b>Target [Eqn]A:</b> <math>\text{Answer} = 200 + 0.45 * 200</math></p>	<p><b>Target [CoT]A:</b> If George has 200 bananas, then he has <math>45/100 * 200 = \langle\langle 45/100 * 200 = 90 \rangle\rangle 90</math> more pears than bananas. This means George has <math>200 + 90 = \langle\langle 200 + 90 = 290 \rangle\rangle 290</math> pears. Therefore, George has a total of <math>200 + 290 = \langle\langle 200 + 290 = 490 \rangle\rangle 490</math> fruits. The answer is <b>490</b>.</p> <p><b>Target [Code]A:</b> <code>def func(): bananas = 200 pears = bananas + (bananas * 45 / 100) return bananas + pears</code></p> <p><b>Target [Eqn]A:</b> <math>\text{Answer} = 200 + 1.45 * 200</math></p>