

Automatic Mathematic In-Context Example Generation for LLM Using Multi-Modal Consistency

Jaeseong Lee Wei Yang Gopal Gupta Shiyi Wei

Department of Computer Science
The University of Texas at Dallas
Richardson, TX

Abstract

Large Language Models (LLMs) have advanced Natural Language Processing (NLP) tasks but are 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 AutoMathIC, a framework that automatically generates high-quality in-context examples to enhance LLMs' mathematical reasoning. AutoMathIC 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, AutoMathIC outperforms six baselines, with LLM accuracy ranging from 87.0% to 99.3% for GPT-3.5 and 93.1% to 98.7% for GPT-4o-mini. It surpasses the state-of-the-art in-context example retrieval method in three of the four datasets by 0.3% to 11.8%, without relying on an annotated dataset.

1 Introduction

LLMs have achieved state-of-the-art performance in many NLP applications (Min et al., 2024). However, they exhibit limited proficiency in solving mathematical questions (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 questions. 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 question and its explanation, embedding the examples into prompts to facilitate learning within the context towards improved performance.

However, in-context learning for math tasks has limitations, requiring extensive resources and often relying on large, externally annotated datasets (Wei

et al., 2022; Zhang et al., 2023). This process can be labor-intensive, requiring manual curation of examples, and computationally expensive, relying on complex retrieval models. Additionally, the external dataset's limited scale restricts the search space for finding suitable math questions and explanations. These limitations restrict the automatic generation of in-context examples, reducing the practicality and scalability of math tasks.

To overcome these limitations, we identify two challenges. First, searching for a suitable math question to use as in-context learning for solving the target question is a challenging. The second challenge lies in automatically generating accurate explanations for in-context learning examples. To address this, it is crucial to ensure that the selected in-context examples not only align semantically with the target question but also contain accurate explanations that can aid in problem-solving. The existing method assesses the in-context example's relevance to the target question by measuring semantic similarity between questions and relies on manually annotated explanations, assuming this similarity reflects explanation quality (Zhang et al., 2023). In situations where external annotated explanations are unavailable, automatically ensuring the accuracy of these explanations remains a challenge.

To address these challenges, we employ a multi-modal technique for the retrieval of in-context examples. Multi-modal learning, which integrates information from diverse sources such as text, images, and videos, has demonstrated potential 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 question, known as modality, such as generating CoT, composing code snippets, or for-

mulating complex mathematical equations (Kojima et al., 2022; Wang et al., 2023b; Imani et al., 2023).

When addressing mathematical questions, the diversity in LLM responses across these modalities can introduce inconsistencies that can undermine the reliability and accuracy of the responses. Modality consistency, the degree of agreement among model predictions across modalities, is crucial in ensuring that the LLM produces stable and uniform responses when solving the same question (Wang et al., 2023b; Imani et al., 2023). The convergence across the modalities not only reduces the likelihood of systematic biases or errors being present only in a single modality but also facilitates cross-validation between modalities, thereby enhancing prediction reliability. This aids in estimating the accuracy of LLM responses. Therefore, the modality consistency of LLM responses can act as an indicator for evaluating the confidence in LLM predictions. 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 questions, AutoMathIC, that addresses the above challenges through the dynamic generation of mutations, thereby eliminating reliance on pre-existing examples. Additionally, AutoMathIC employs consistency checks across modalities in a zero-shot setting to improve both performance and reliability. AutoMathIC operates by initially generating a collection of mutated math questions and their corresponding LLM responses across various modalities. This procedure ensures that the mutation maintains the same reasoning algorithm utilized for solving the target question, resulting in potentially the most relevant in-context examples, addressing the first challenge. Subsequently, AutoMathIC selects a subset of mutated examples that improves consistency of responses across modalities for the target math question. This tackles the second challenge by evaluating LLM responses of mutations through consistency. Doing so elevates the confidence level of the LLM, thereby leading to a correct answer.

Our experiments show that AutoMathIC produces higher accuracies than the baselines, from 83.8% to 99.3% for GPT-3.5 and from 93.0% to 98.7% for GPT-4o-mini over four arithmetic reasoning datasets. In addition, AutoMathIC outperforms the state-of-the-art in-context retrieval method in three out of the four datasets by 0.3% to

11.8%, without an external annotated dataset.

2 Motivation

In this section, we present responses to an arithmetic math question using in-context examples to motivate the development of AutoMathIC. Figure 1 illustrates an example from the SVAMP, a widely used dataset for arithmetic reasoning questions (Patel et al., 2021). This demonstrates the diverse responses of an LLM (GPT-3.5) to the same math question under different settings. At the top, it presents the answers to the math question 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. 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 question 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 reasoning algorithm, represented by “ $x - 65 + 39 = 67$ ”, which is also applied to solve the target question, “ $x - 17 + 10 = 7$ ” (step 4).

Overall, our study is motivated by the examples with regard to two key facets. First, altering the numerical values in a math question 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 question. This insight facilitates the automated generation of math questions that are assumed to operate under the same reasoning algorithm. Second, there is a correlation between the quality of LLM responses and the consistency of their answers across different modalities. As shown in Figure 1, when an LLM provides consistent solutions across modalities, it indicates a higher confidence level and results in more accurate answers. This consistency implies that the LLM has effectively grasped the underlying reasoning algorithm, leading to improved performance in solving arithmetic reasoning questions. It motivates us to utilize the degree of consistency as a metric for assessing the confidence level of responses, thereby enabling the automatic selection of high-quality in-context examples.

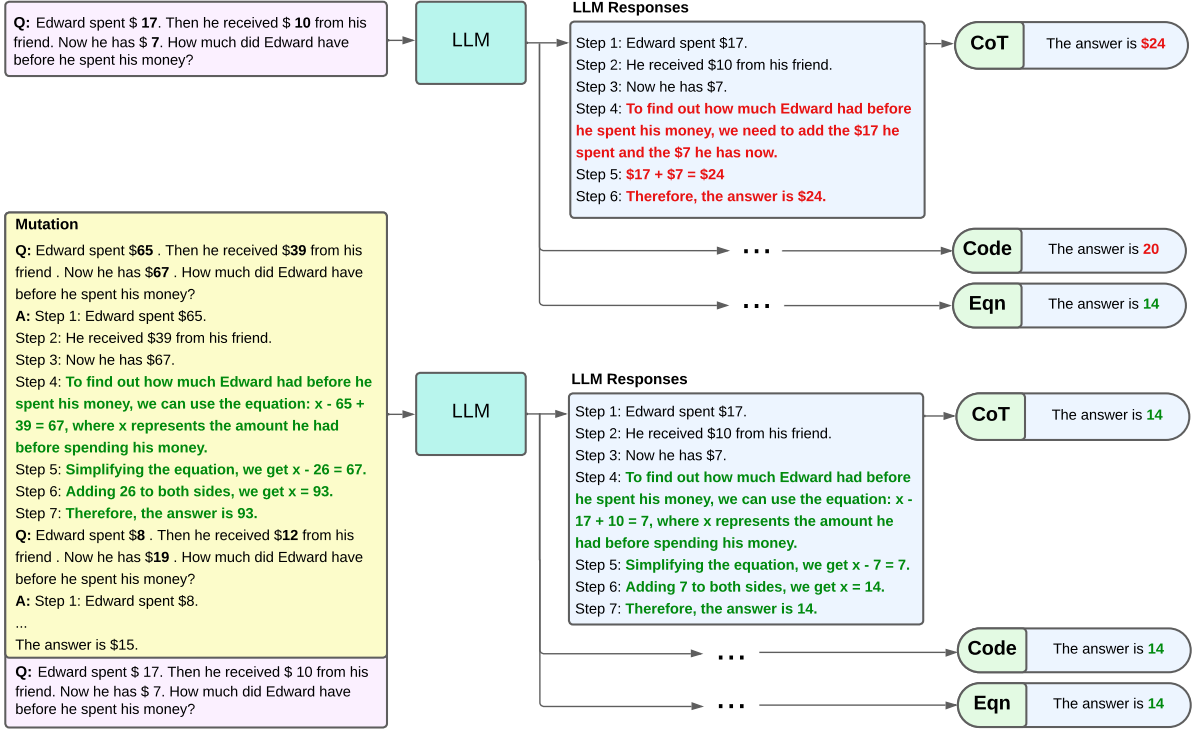


Figure 1: Responses of GPT-3.5 to an arithmetic reasoning problem derived from the SVAMP dataset (Patel et al., 2021). The top presents the responses across CoT, Python code, and Mathematical Equation in a zero-shot setting. The bottom shows the responses in the CoT modality utilizing in-context examples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

3 Approach

We have developed AutoMathIC, an automated in-context example generation framework. AutoMathIC is structured with two main steps: 1. We employ a mutation technique on the target math question for the generation of relevant in-context examples, considering these mutations as potential candidates for in-context examples. 2. We employ a genetic algorithm to select a subset of mutations, aiming to maximize response consistency with the target question. These steps ensure the generated examples are relevant and enhance response quality. Figure 2 presents an overview of AutoMathIC. The *Initial Consistency Computation Phase* begins by inputting the target question along with an LLM. It collects the LLM’s initial responses to the target question across modalities. The consistency of these responses is then assessed. If maximum consistency is achieved (i.e., all responses are identical), the phase concludes by returning the consistent answer as the output. If not, the process advances to the *Target Question Mutation Phase*. This phase mutates the target question by altering the numerical values in the question. Furthermore, we use the LLM to process these mutated questions and obtain their corresponding re-

sponses 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 questions 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 question are collected for each mutation, prepended as an in-context example. If the consistency of these responses 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 question and an LLM, we generate the top- K responses for each modality. The answers are then extracted from these LLM re-

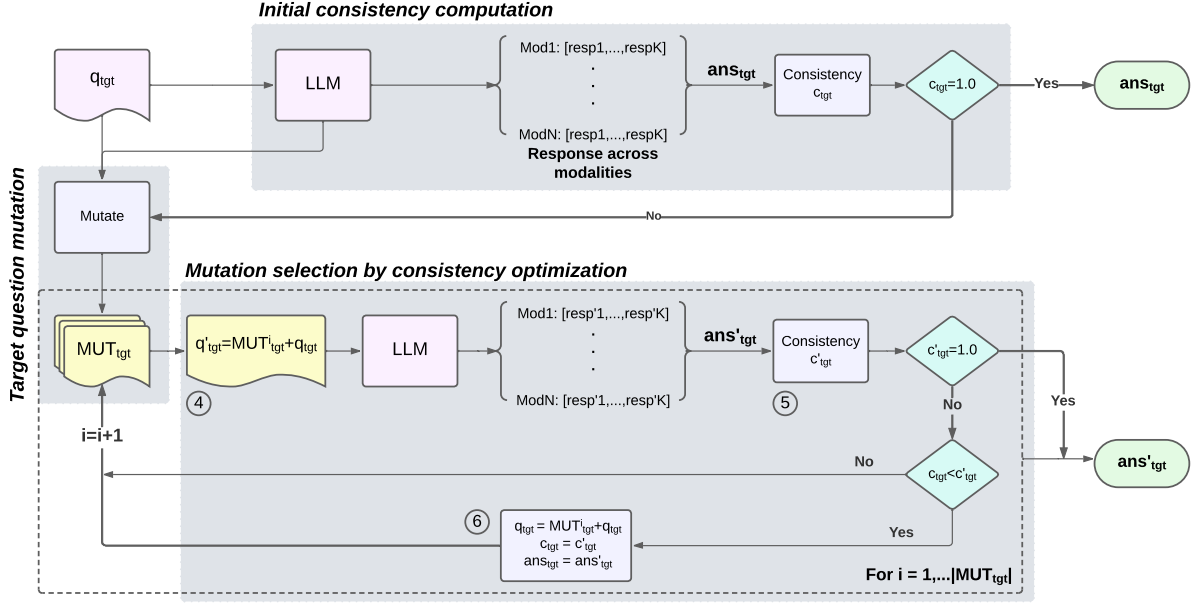


Figure 2: Overview of AutoMathIC.

sponses, and their consistency is evaluated across the different modalities. Formally, for a specific modality mod in MOD , the set of all modalities, we define the top- K answers as ANS_{mod} , and the overall collection of LLM answers across all modalities as $ANS = \{ANS_{mod} \mid ANS_{mod} = LM(q_{tgt}, p_{mod}) \cap p_{mod} \in P_{MOD} \cap mod \in MOD\}$, where LM represents the LLM, q_{tgt} is the target question, P_{MOD} is the set of prompts across modalities. 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 in 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 the 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 AutoMathIC. 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 question fails to reach the maximum value, we proceed to the mutation phase.

3.2 Target Question Mutation

This phase generates a pool of questions, potentially offering relevant knowledge for the LLM to address the target question effectively. This is achieved by mutating the target question, resulting in a set of mutated questions. We operate under the assumption that a problem identical to the original one, but with different numerical values that preserve the characteristics of the original values—such as being positive or negative, and being floating-point or natural numbers—follows the same reasoning path in solving the original problem. The phase first identifies the numerical values present in the target question. These identified values are then randomly mutated; the original values are replaced with their mutated values. To maintain the realism and validity of the generated questions, the mutations are constrained to ensure that the resulting numerical values remain positive and consistent with the data type of the original values. Subsequently, LLM responses are then obtained to these mutated questions across modalities. The validity of these mutations is verified by estimating the accuracy of the LLM responses to the mutated questions 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 question, this

Algorithm 1 Consistency optimization algorithm.

```
1: Input: a large language model  $LM$ , target question  $qtgt$ , 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  $M_{qtgt}$ , 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:      $m = M_{qtgt}.pop()$ 
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:       $r_{mod}^+ = LM(qtgt, p_{mod}^+)$ 
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  $ans_{final}$ 
```

phase generates pairs of mutated questions 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 question. To address this challenge, we have developed an optimization strategy aimed at improving consistency as defined in the Equation 3.

Algorithm 1 summarizes the optimization strategy used to identify in-context examples. This algorithm operates by taking as input the following: the LLM, the target question, prompts across modalities, the initial answer generated for the target question, and its corresponding consistency score, which was determined as described in Section 3.1. Additionally, the algorithm uses a pool of potential mutations and the predefined maximum number of in-context examples that can be utilized. If a maximum consistency value is achieved, indicating that all answers are the same across modalities, it returns this answer as the final answer (lines

Algorithm 2 get_answer algorithm.

```
1: Input: responses of the target question over modalities  $R$ , belief weights across modalities  $W$ 
2: Output: final answer  $ans^*$ , consistency score of the final answer  $cs^*$ 
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  $uniq\_ans$  do
8:      $sc[mod, ans] = self\_consistency(ans, Ans_{topk})$ 
9:    $unique\_answers.add(uniq\_ans)$ 
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:  $ans^*, cs^* = get\_highest\_score\_answer(cs)$ 
15: return  $ans^*, cs^*$ 
```

6-7). Otherwise, the algorithm integrates each mutation presented to the LLM within the prompts (lines 9-18). Following the incorporation of mutations, the LLM generates responses across the modalities. The consistency of these responses is then evaluated using the $get_answer()$ algorithm, as described in Algorithm 2 (line 18).

Algorithm 2 operates by taking LLM responses over modalities and belief weights across modalities as input. For each modality, it calculates the self-consistency score for each unique answer generated by the LLM across the modalities (lines 5-9). Next, using the self-consistency scores, the algorithm then computes the overall consistency score for each unique answer, as defined mathematically in Equation 3 (lines 10-13). The algorithm then identifies and returns the highest consistency score, along with the corresponding answer that achieves this score, which is considered the most reliable answer (lines 14-15).

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-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 question, 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 question is then provided

Modality	Prompt
CoT	“{QUESTION} Let’s think step by step and end your response with ‘the answer is {answer}’”
Code	“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}”
Equation	“{QUESTION} Write a wolframalpha mathematical equation with no explanations and no units to the numbers in the equation. Generate the answer format starting with ‘Answer =’”

Table 1: LLM prompts over different modalities.

through this iterative optimization.

4 Experiments

4.1 Experimental setup

Dataset. We assess the performance of AutoMathIC on the 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 questions.

Large Language Models. We evaluate AutoMathIC using GPT-3.5 and GPT-4o-mini (OpenAI, 2023, 2024), which are transformer-based architectures with 175 billion and 8 billion parameters, respectively. Specifically, we utilize the public engine *gpt-3.5-turbo* from the OpenAI models. These models differ significantly in scale and architectural parameters.

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

- **Zero-shot with a specific Modality:** It solely uses a specific modality without any in-context examples. In this experiment, we utilize the CoT, Code, and Equation modalities, denoted as CoT-Prompt, 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):** It determines the final answer by majority voting of answers across modalities.
- **AutoMathIC w/o modalities:** For a specific modality, we employ a subset of mutations that improve the self-consistency, as defined in Equation 2, of top- K LLM responses, utilizing these as in-context examples. Specifically, we extract the top-3 CoT responses.
- **In-context retrieval method (AutoCoT):** AutoCoT (Zhang et al., 2023) is implemented. It clusters the embedding vectors of retrieval examples using Sentence-BERT (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 LLMs. 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 weights used for aggregation across the modalities are set to 0.4 for the CoT modality and 0.3 for both the code and equation modalities, respectively. These weights are determined based on empirical performance, highlighting the CoT modality’s pivotal role 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

This section presents the experimental results and these results are available at the AutoMathIC repository (Lee et al., 2024)

Comparison of AutoMathIC with Zero-Shot Baselines. Table 2 shows LLM accuracies on different math problem datasets. The first two columns denote the dataset names and the number of problems used. Columns 3-6 show accuracies achieved using baseline methods, and the last column shows the accuracy with AutoMathIC. *AutoMathIC outperforms all baselines on all four datasets.* Over the two LLMs, AutoMathIC achieved better accuracy than the baselines by 1% to 7.3% on ASDiv, 1.4% to 15.4% on SVAMP, 4% to 22.4% on GSM8k, and 0.2% to 62.1% on MultiArith. This signifies significant improvements in accuracy across various mathematical problem datasets in the zero-shot setting. The additional contextual processing enabled by AutoMathIC’s mutation selection phase is crucial for handling mathematical queries that may not be effectively addressed through standard zero-shot methodologies. In Appendix A.1, we provide samples of AutoMathIC-generated in-context examples for each dataset.

Effectiveness of Mutation as In-context Examples. Table 3 compares LLM accuracies using AutoMathIC and AutoCoT. The datasets are listed in the first column, with the accuracies of AutoCoT

and AutoMathIC in the second and third columns, respectively. AutoMathIC improves GPT-3.5’s accuracy by up to 4.4%, except for a 0.3% decrease on ASDiv. For GPT-4o-mini, AutoMathIC boosts accuracy by 0.6% to 11.8% across all datasets. Overall, *AutoMathIC’s in-context examples are more effective than current retrieval-based methods without external datasets and models.* We also manually assess consistency across modalities to evaluate LLM responses to mutated questions, determining correctness by comparing responses to ground truth data. Table 4 shows the number of mutations used and the number of correct mutations for each dataset. AutoMathIC achieves 73.3% (132/180) to 100% (24/24) accuracy across datasets. Accuracy variations are due to differing problem complexities. These results suggest that consistency across modalities is crucial for response correctness, enhancing LLM evaluation effectiveness. **Effectiveness of Modalities.** Comparing to AutoMathIC w/o modalities (fourth column in Table 3), AutoMathIC shows a 3.1% and 0.6% decrease in accuracy for the GSM8k and ASDiv datasets using GPT-3.5 and GPT-4o-mini, respectively. However, it improves accuracy by 1% to 2.5% with GPT-3.5 and 0% to 1.1% with GPT-4o-mini. For example, of the 79 problems AutoMathIC correctly solved but AutoMathIC w/o modalities failed using GPT-3.5, AutoMathIC could not find in-context examples for 53 problems, while AutoMathIC w/o modalities could. The remaining 26 discrepancies are due to randomness from the LLM’s temperature setting. This indicates that *enhancing consistency across modalities is effective.*

Effectiveness of Consistency Optimization. We also conduct an ablation study of the optimization phase in AutoMathIC, as described in Section 3.3. While the optimization methods improve accuracies in certain cases—such as the ASDiv with the GPT-3.5 and SVAMP and GSM8k with the GPT-4o-mini model—they had minimal impact or slight negative effects in others. Specifically, using the ASDiv, SVAMP, MultiArith, and GSM8k datasets, while the GPT-4o-mini improved consistency scores, it produced the same incorrect results of AutoMathIC both with and without the optimization phase for 23, 41, 8 and 82 examples, respectively. Among these examples, AutoMathIC applied 6, 16, 4, and 59 mutations but failed to correct the responses. Additionally, 1, 1, 0 and 10 mutations selected via the optimization phase converted 2, 1, 0, and 17 correct results into incorrect

Dataset	#Data	CoTPrompt [%]	CodePrompt [%]	EqnPrompt [%]	MajVotModals [%]	AutoMathIC [%]
ASDiv	1218	95.2 98.2	95.0 97.6	89.8 94.7	96.1 97.9	97.1 98.0
SVAMP	1000	79.5 93.6	79.7 91.2	71.6 88.1	83.2 94.4	87.0 95.8
GSM8k	1319	77.5 90.9	69.4 87.1	61.4 74.2	79.8 90.8	83.8 93.0
MultiArith	600	96.0 98.0	98.3 97.7	37.2 93.3	97.0 98.5	99.3 98.7

Table 2: Accuracies of LLMs on various math problem datasets in the zero-shot setting. For each dataset, the left and right numbers represent results from GPT-3.5 and GPT-4o-mini, respectively.

Dataset	AutoCoT [%]	AutoMathIC w/o Opt [%]	AutoMathIC w/o modalities [%]	AutoMathIC [%]
ASDiv	97.4 97.4	96.9 98.1	96.1 98.6	97.1 98.0
SVAMP	82.6 84.0	87.0 95.3	85.4 94.7	87.0 95.8
GSM8k	81.4 97.4	83.9 92.2	86.9 93.0	83.8 93.0
MultiArith	97.2 97.7	99.5 98.7	96.8 98.7	99.3 98.7

Table 3: Comparison of LLM accuracies with in-context example retrieval methods across math problem datasets. For each dataset, the left and right numbers represent results from GPT-3.5 and GPT-4o-mini, respectively.

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

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.

ones, respectively. Searching within 20 random mutated questions for our experiment hinders the achievement of high-quality in-context examples.

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 AutoMathIC 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 quantify answer consistency across modalities and use it to estimate LLM response accuracy with 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, AutoMathIC 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 AutoMathIC 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 AutoMathIC, a novel tool that automates the generation of relevant in-context examples to enhance the arithmetic problem-solving capabilities of LLMs. AutoMathIC 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 AutoMathIC 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, AutoMathIC relies on random mutations of numerical values, limiting the realism of generated math problems. Future work will explore context-aware mutation techniques. Second, AutoMathIC eval-

uates LLM responses to multiple mutated questions across modalities, which can increase time consumption. To address this computational overhead, we implement an adaptive mutation mechanism that dynamically adjusts the number of mutations and in-context examples. Enhancing evaluation efficiency is identified as a direction for future work. Third, AutoMathIC shows advantages mainly for solving arithmetic math problems; we focus on arithmetic reasoning as a key benchmark for LLMs’ mathematical understanding and problem-solving. Though currently centered on arithmetic, AutoMathIC’s principles are generalizable, with future plans to extend mutation generation to a broader range of reasoning problems.

Acknowledgments

This work was partly supported by NSF grants CCF-2008905, CCF-2047682, CCF-2146443 and ISI-1910131, and multiple industry grants through the UT Dallas Center for Applied AI and Machine Learning (CAIML).

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 Pillai, 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*.
- Jaeseong Lee, Wei Yang, Gopal Gupta, and Shiyi Wei. 2024. [AutoMathIC](#).
- 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.
- Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. [A diverse corpus for evaluating and developing English math word problem solvers](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.
- Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2024. [Recent advances in natural language processing via large pre-trained language models: A survey](#). *ACM Comput. Surv.*, 56(2):30:1–30:40.
- Piotr Mirowski, Kory W. Mathewson, Jaylen Pittman, and Richard Evans. 2022. [Co-writing screenplays and theatre scripts with language models: An evaluation by industry professionals](#). *CoRR*, abs/2209.14958.
- OpenAI. 2023. [Introducing chatgpt](#). <https://openai.com/blog/chatgpt>.
- OpenAI. 2024. [Gpt-4o mini: advancing cost-efficient intelligence](#). <https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/>.
- Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. [Are NLP models really able to solve simple math word problems?](#) In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2080–2094. Association for Computational Linguistics.
- Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. [Grips: Gradient-free, edit-based instruction search for prompting large language models](#). In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pages 3827–3846. Association for Computational Linguistics.

- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. [Learning transferable visual models from natural language supervision](#). In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR.
- Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, H. Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimppoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sotiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d’Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew J. Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorraine Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. [Scaling language models: Methods, analysis & insights from training gopher](#). *CoRR*, abs/2112.11446.
- Nils Reimers and Iryna Gurevych. 2019. [Sentence-bert: Sentence embeddings using siamese bert-networks](#). *CoRR*, abs/1908.10084.
- Subhro Roy and Dan Roth. 2016. [Solving general arithmetic word problems](#). *CoRR*, abs/1608.01413.
- Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. [Learning to retrieve prompts for in-context learning](#). In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 2655–2671. Association for Computational Linguistics.
- Nikunj Saunshi, Sadhika Malladi, and Sanjeev Arora. 2021. [A mathematical exploration of why language models help solve downstream tasks](#). In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.
- Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ameet Rahane, Anantharaman S. Iyer, Anders Andreassen, Andrea Santilli, Andreas Stuhlmüller, Andrew M. Dai, Andrew La, Andrew K. Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubakaran, Asher Mullokandov, Ashish Sabharwal, Austin Herick, Avia Efrat, Aykut Erdem, Ayla Karakas, and et al. 2022. [Beyond the imitation game: Quantifying and extrapolating the capabilities of language models](#). *CoRR*, abs/2206.04615.
- Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. [Selective annotation makes language models better few-shot learners](#). In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.
- 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 *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.
- 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](#). *CoRR*, abs/2307.07415.
- 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 *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 3916–3930. Association for Computational Linguistics.
- Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022b. [Black-box tuning for language-model-as-a-service](#). In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 20841–20855. PMLR.
- Liang Wang, Nan Yang, and Furu Wei. 2023a. [Learning to retrieve in-context examples for large language models](#). *CoRR*, abs/2307.07164.
- 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 *The Eleventh International Conference*

on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net.

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 *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*.

WolframAlpha. 2023. [Wolframalpha apis](#). <https://products.wolframalpha.com/api>.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2022. [An explanation of in-context learning as implicit bayesian inference](#). In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2023. [Large language models as optimizers](#). *CoRR*, abs/2309.03409.

Kevin Yang, Yuandong Tian, Nanyun Peng, and Dan Klein. 2022. [Re3: Generating longer stories with recursive reprompting and revision](#). In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 4393–4479. Association for Computational Linguistics.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2023. [Automatic chain of thought prompting in large language models](#). In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. [Factual probing is \[MASK\]: learning vs. learning to recall](#). In *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*, pages 5017–5033. Association for Computational Linguistics.

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 *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

A Appendix

A.1 Qualitative Analysis of AutoMathIC

Tables ?? to ?? present four illustrative examples from the datasets employed in our experiments to evaluate AutoMathIC. In each table, the first row displays in-context examples generated by AutoMathIC. 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 AutoMathIC, respectively.

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 AutoMathIC 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.

(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: $\text{Answer} = 69 * 15 + 42$

(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: $\text{Answer} = 52 * 10 + 50$

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: $\text{Answer} = 7 * 4 + 7 * 3$

After AutoMathIC

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: $\text{Answer} = 7 * 4 + 3$

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

(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 AutoMathIC

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 6: AutoMathIC generated in-context examples for SVAMP.

(In-context 1)

Q: 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?

[CoT]A: 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 $15 * 38 = 570$ 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 $36 * 38 = 1368$ 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 $570 + 1368 = 1938$ points. Therefore, the answer is 1938.

[Code]A: `def func(): gwen_pounds = 15 friends_pounds = 36 points_per_pound = 38 total_points = (gwen_pounds + friends_pounds) * points_per_pound return total_points`

[Eqn]A: $Answer = 15*38 + 36*38$

Target Q: 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?

Zero-shot

Target [CoT]A: 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. $1 + 4$ equals 5. Therefore, the answer is 5 points.

Target [Code]A: `def func(): gwen_pounds = 5 friends_pounds = 13 total_pounds = gwen_pounds + friends_pounds points = total_pounds // 3 return points`

Target [Eqn]A: $Answer = (5 + 13) / 3$

After AutoMathIC

Target [CoT]A: 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 $5 + 13 = 18$ 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

Target [Code]A: `def func(): gwen_pounds = 5 friends_pounds = 13 total_points = (gwen_pounds + friends_pounds) // 3 return total_points`

Target [Eqn]A: $Answer = 6$

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

(In-context 1)

Q: George has 46% more pears than bananas . If George has 90 bananas, how many fruits does George have?

[CoT]A: If George has 90 bananas, then he has $46/100 * 90 = \langle 46/100*90=41.4 \rangle 41.4$ more pears than bananas. This means George has $90 + 41.4 = \langle 90+41.4=131.4 \rangle 131.4$ pears. In total, George has $90 + 131.4 = \langle 90+131.4=221.4 \rangle 221.4$ fruits. The answer is 221.4.

[Code]A: def func(): bananas = 90 pears = bananas * 1.46 total_fruits = bananas + pears return total_fruits

[Eqn]A: Answer = $90 + 1.46 * 90$

(In-context 2)

Q: George has 56% more pears than bananas . If George has 74 bananas, how many fruits does George have?

[CoT]A: If George has 74 bananas, then he has $56/100 * 74 = \langle 56/100*74=41.44 \rangle 41.44$ more pears than bananas. This means George has $74 + 41.44 = \langle 74+41.44=115.44 \rangle 115.44$ pears. Therefore, George has a total of $74 + 115.44 = \langle 74+115.44=189.44 \rangle 189.44$ fruits. The answer is 189.44.

[Code]A: def func(): bananas = 74 pears = bananas + (bananas * 56 / 100) return bananas + pears

[Eqn]A: Answer = $74 + 1.56 * 74$

Target Q: George has 45% more pears than bananas. If George has 200 bananas, how many fruits does George have?

Zero-shot

Target [CoT]A: Step 1: Calculate 45% of 200 bananas. 45% of 200 = $(45/100) * 200 = 0.45 * 200 = 90$ Step 2: Add the result from step 1 to the number of bananas. $200 + 90 = 290$ Therefore, George has 290 fruits. The answer is **290**.

Target [Code]A: def func(): bananas = 200 pears = **bananas + (0.45 * bananas)** return bananas + pears

Target [Eqn]A: Answer = **$200 + 0.45 * 200$**

After AutoMathIC

Target [CoT]A: If George has 200 bananas, then he has $45/100 * 200 = \langle 45/100*200=90 \rangle 90$ more pears than bananas. This means George has $200 + 90 = \langle 200+90=290 \rangle 290$ pears. Therefore, George has a total of $200 + 290 = \langle 200+290=490 \rangle 490$ fruits. The answer is **490**.

Target [Code]A: def func(): bananas = 200 **pears = bananas + (bananas * 45 / 100)** return bananas + pears

Target [Eqn]A: Answer = **$200 + 1.45 * 200$**

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