Leveraging Training Data in Few-Shot Prompting for Numerical Reasoning

Zhanming Jie
ByteDance Research
allan@bytedance.com

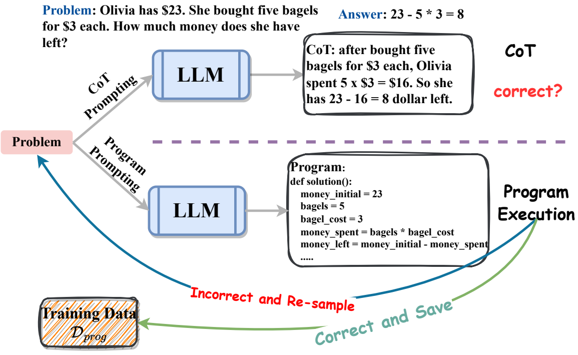
Wei Lu
StatNLP Research Group
Singapore University of Technology and Design
luwei@sutd.edu.sg

Abstract

Chain-of-thought (CoT) prompting with large language models has proven effective in numerous natural language processing tasks, but designing prompts that generalize well to diverse problem types can be challenging (Zhou et al., 2022), especially in the context of math word problem (MWP) solving. Additionally, it is common to have a large amount of training data that have a better diversity coverage but CoT annotations are not available, which limits the use of supervised learning techniques. To address these issues, we investigate two approaches to leverage the training data in a few-shot prompting scenario: *dynamic program prompting* and *program distillation*. Our approach is largely inspired by Gao et al. (2022), where they proposed to replace the CoT with the programs as the intermediate reasoning step. Such a prompting strategy allows us to accurately verify the answer correctness through program execution in MWP solving. Our dynamic program prompting involves annotating the training data by sampling correct programs from a large language model, while program distillation involves adapting a smaller model to the program-annotated training data. Our experiments on three standard MWP datasets demonstrate the effectiveness of these approaches, yielding significant improvements over previous baselines for prompting and fine-tuning. Our results suggest that leveraging a large amount of training data can improve the generalization ability of prompts and boost the performance of fine-tuned small models in MWP solving[1].

1 Introduction

Designing effective prompts is crucial for the success of few-shot prompting with large language models (LLMs) in tasks requiring complex reasoning skills (Wei et al., 2022; Zhou et al., 2022; Shrivastava et al., 2022; Fu et al., 2022). Especially for the task of arithmetic word problem, it poses a significant challenge to design a small number of chain-of-thought (CoT) prompts (Wei et al., 2022) to solve a wide range of problems.


Problem: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?

Answer: 23 - 5 * 3 = 8

CoT: after bought five bagels for $3 each, Olivia spent 5 x $3 = $16. So she has 23 - 16 = 8 dollar left.

```
Program:
def solution():
    money_initial = 23
    bagels = 5
    bagel_cost = 3
    money_spent = bagels * bagel_cost
    money_left = money_initial - money_spent
    .....
```

Figure 1: Program annotation with LLM.

Fortunately, a modest amount of training data is usually available though no chain-of-thought (CoT) annotation exists. Rubin et al. (2021) adopts a retrieval-based approach to select similar samples as prompts. While such a method does not work well for numerical reasoning compared to CoT prompting (Wei et al., 2022), recent work (Magister et al., 2022) also tried to distill the CoT knowledge from large language models to smaller language models. The distilled CoT annotations allow us to further fine-tune the small language models. However, there is no guarantee that the generated CoT prompts for the training data are correct, and it is challenging to perform an automatic evaluation to verify the CoT correctness. As an alternative, recent work (Drori et al., 2022; Gao et al., 2022; Mishra et al., 2022) has adopted programs as intermediate reasoning chains in tasks such as math word problems (MWPs), allowing for automatic verification of answers through program execution. Inspired by these approaches, we can perform prompting with code generation models, such as Codex (Chen et al., 2021), to annotate the training data with programs that can be executed. Figure

[1] Our code and data are available at https://github.com/allanj/dynamic-pal.

Findings of the Association for Computational Linguistics: ACL 2023, pages 10518–10526
July 9-14, 2023 ©2023 Association for Computational Linguistics

```python
def solution():
"""Natalia sold clips to 48 of her friends in
    April, and then she sold half as many clips
    in May. How many clips did Natalia sell
    altogether in April and May?"""
clips_april = 48
clips_may = clips_april / 2
clips_total = clips_april + clips_may
result = clips_total
return result
```

Figure 2: Example program from the GSM8K training set following the format in PAL.

Dataset	#Train	#Program	#Valid	#Test
GSM8K	7,473	6,363 (85.1%)	-	1,319
SVAMP	3,138	3,071 (97.9%)	-	1,000
MathQA†	16,191	7,676 (47.4%)	2,411	1,605

Table 1: Dataset statistics and the percentage of annotated programs. †: We follow Jie et al. (2022) to obtain the preprocessed split.

1 shows the process of automatic program annotation using large language models. As we can see in this example, though the final answer "*8 dollar*" by CoT is correctly generated, the intermediate reasoning path is wrong because of incorrect calculation for "5×3". Instead, the program sampling is relatively more rigorous in that we can execute to obtain the numeric answer rather than CoT in natural language. Apparently, we can keep sampling the program with different temperatures until the answer executed from the program matches the correct one. Once we obtain the annotated program, we can use the "*annotated*" training data with the pseudo-gold program to further improve the performance on the test set.

In this work, we primarily study two approaches for making use of the "*annotated*" programs: *dynamic program prompting* and *program distillation* (Magister et al., 2022). Dynamic program prompting employs the top-k similar training samples (with annotated pseudo-gold programs) as few-shot prompts. We use publicly available and state-of-the-art sentence encoders such as OpenAI embeddings (Neelakantan et al., 2022)[2] and Sentence-T5 (Ni et al., 2022) for computing the cosine similarity. On the other hand, we follow Magister et al. (2022) to fine-tune smaller language models on our pseudo-gold training data. Overall, our experiments on three standard math word problem datasets demonstrate the effectiveness of leveraging the training program in our few-shot prompting. We observe significant improvements for all datasets, especially for the MathQA (Amini et al., 2019) dataset, where diverse subjects (e.g., physics, probability, etc.) were involved in the problems where the fixed prompt accompanied by limited examples is insufficient to encapsulate the entire scope of requisite knowledge.

2 Approach

Training Data Annotation Following the approach in program-aided language model (PAL) (Gao et al., 2022), we can sample the program for each math word problem as an annotation. Specifically, we use the math prompts from PAL as seed prompts to perform few-shot prompting with large language models (i.e., Codex (Chen et al., 2021)). We follow the exact same format from PAL (Gao et al., 2022) without any changes, Figure 2 shows an example program from the GSM8K training set. We can verify the answer's correctness by comparing the result from the program execution with the ground-truth value.

For each math word problem $\boldsymbol{x}$ in training set $\mathcal{D}$, we first perform greedy decoding with temperature $T = 0$ to obtain the bet Python program. If the predicted answer $\hat{y}$ from the executed program $\boldsymbol{P}$ matches the ground-truth answer y, we add this tuple $(\boldsymbol{x}, \boldsymbol{P}, y)$ into a new training data set $\mathcal{D}_{prog}$. If the predicted answer is incorrect, we increase the temperature and continue sampling programs until we find one with the correct answer. In practice, we may not always obtain the correct answer and have a limited budget for Codex API usage. Thus, we sample at most K times for each instance. If we cannot find a program with the correct answer within K samples, we discard the instance $\boldsymbol{x}$. As a result, the size of the resulting training set $\mathcal{D}_{prog}$ is expected to be smaller than the original training set (refer to Table 1).

2.1 Dynamic Program Prompting

Prompt Retrieval Given all the instances $(\boldsymbol{x}, \boldsymbol{P}, y)$ in $\mathcal{D}_{prog}$, we retrieve the top M most relevant instances as prompts. We use state-of-the-art sentence embeddings such as sentence-T5 (Ni et al., 2022) and SimCSE (Gao et al., 2021) to obtain the representation for each math word problem $\boldsymbol{x}$. We then compute the cosine similarity between

[2] https://openai.com/blog/new-and-improved-embedding-model/

	Model	#Param	GSM8K	SVAMP	MathQA
Prompting	LaMDA (Thoppilan et al., 2022)	137B	17.1	-	-
	PaLM (Chowdhery et al., 2022)	540B	58.1	79.0	-
	GPT-3 CoT (`text-davinci-002`)	175B	48.1	-	-
	Codex CoT (`code-davinci-002`)	175B	65.6	74.8	29.9
	Complex CoT (Fu et al., 2022)	175B	55.4	-	36.0†
	PAL (Gao et al., 2022)	175B	72.0	79.4	-
	PAL (`reproduced`)	175B	71.6	77.4	30.0
	Our Dynamic Program Prompting	175B	**76.6**	**80.3**	**61.7**
Fine-tuning	GPT-3	175B	33.1	-	-
	CoT Fine-tune (Magister et al., 2022)	11B	38.2	-	-
	CoT Fine-tune (CodeGen)	6B	35.3	40.2	25.3
	Our Program Distillation	6B	**39.0**	**48.0**	**50.6**

Table 2: Performance comparison over previous approaches using prompting and fine-tuning. †: not directly comparable as they use less amount of test data.

each test sample and all training samples. Based on the similarities, we select the most similar M exemplars from the training instances in $\mathcal{D}_{prog}$.

Similarity To further verify the effectiveness of using similarity to select the prompting exemplars, we also experiment with alternative strategies such as random selection from $\mathcal{D}_{prog}$ and selecting the exemplar with the least similarity.

2.2 Program Distillation

Our purpose is to train a smaller model using the annotated data compared to LLMs such as Codex (Chen et al., 2021). In order to do this, we follow the approach of fine-tuning a pre-trained model on $\mathcal{D}_{prog}$, similar to Magister et al. (2022). Given a math word problem $\boldsymbol{x}$, our objective is to generate the corresponding Python program $\boldsymbol{P}$. We use the publicly available CodeGen (Nijkamp et al., 2022) for this task as it is trained specifically for code generation and pre-trained models are available[3]. CodeGen is a standard Transformer-based (Vaswani et al., 2017) autoregressive model.

3 Experiments

Dataset and Experiment Setting Similar to Fu et al. (2022), we mainly conduct experiments on GSM8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), and MathQA (Amini et al., 2019) datasets. Table 1 shows the statistics and the number of annotated programs. The programs are annotated via few-shot prompting with PAL (Gao et al., 2022) (§2). We perform prompting with Codex (`code-davinci-002`) where the API usage is free. Following Gao et al. (2022), we set the maximum token for the generation to 600. The training set in the SVAMP dataset is the easiest as we can obtain pseudo-gold programs about 98% We only managed to obtain program annotations for 47.4% of the instances for MathQA, as it is the most challenging and noisy-labeled (Fu et al., 2022) with diverse problem types (e.g., physics, probability, geometry, etc).

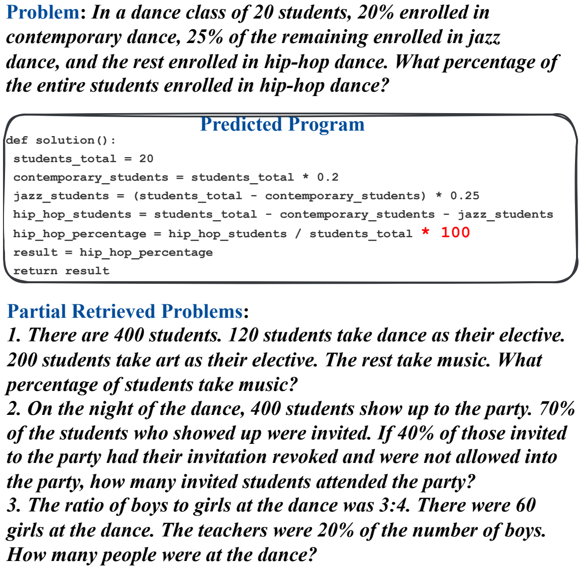
The maximum number of sampling K for each training instance is set to 5[4], and the temperature T is 0.5 following previous practice (Zelikman et al., 2022). We discard the training instance if we cannot find a proper program. The number of prompts M is set to 8 following previous work in math word problem solving (Gao et al., 2022; Fu et al., 2022; Wei et al., 2022). In fine-tuning experiments, we use the 6B CodeGen language model. The learning rate for fine-tuning experiments is 2e-5. We fine-tune the CodeGen model with a batch size of 48 and experiment with 40 epochs on all datasets. The fine-tuning Experiments are conducted with 8 A100 GPUs. We did not perform a hyper-parameter search for fine-tuning. All parameters are set to the above default values.

[3] `https://huggingface.co/Salesforce/codegen-16B-mono`

[4] We chose $K = 5$ to strike a balance between cost and efficiency. Increasing K may not lead to significant improvements.

		GSM8K	SVAMP	MathQA
Most Similar M Exemplars	OpenAI	76.6	80.3	61.7
	`SimCSE` (Gao et al., 2021)	76.4	80.1	61.0
	`ST5` (Ni et al., 2022)	76.6	79.9	61.6
Random	-	74.4	78.1	34.0
Least Similar M Exemplars	OpenAI	73.5	78.2	34.1
	`SimCSE` (Gao et al., 2021)	76.0	78.4	34.7
	`ST5` (Ni et al., 2022)	74.2	77.9	34.3

Table 3: Performance comparison among different sentence representations.



Problem: ***In a dance class of 20 students, 20% enrolled in contemporary dance, 25% of the remaining enrolled in jazz dance, and the rest enrolled in hip-hop dance. What percentage of the entire students enrolled in hip-hop dance?***

Predicted Program

```
def solution():
 students_total = 20
 contemporary_students = students_total * 0.2
 jazz_students = (students_total - contemporary_students) * 0.25
 hip_hop_students = students_total - contemporary_students - jazz_students
 hip_hop_percentage = hip_hop_students / students_total * 100
 result = hip_hop_percentage
 return result
```

Partial Retrieved Problems:

1. There are 400 students. 120 students take dance as their elective. 200 students take art as their elective. The rest take music. What percentage of students take music?

2. On the night of the dance, 400 students show up to the party. 70% of the students who showed up were invited. If 40% of those invited to the party had their invitation revoked and were not allowed into the party, how many invited students attended the party?

3. The ratio of boys to girls at the dance was 3:4. There were 60 girls at the dance. The teachers were 20% of the number of boys. How many people were at the dance?

Figure 3: Example prediction by our prompting approach and the corresponding retrieved problems.

Main Results We conduct both prompting and fine-tuning experiments on all datasets. Table 2 shows the performance comparison with previous prompting approaches using large language models. Similar to PAL (Gao et al., 2022), our results demonstrate that program-based approaches achieve the best performance across all the datasets. Our approach, based on the same underlying mechanism as PAL, achieves new state-of-the-art performance (at the time of submission) with a 175B model and obtains at most 5-point absolute improvements over PAL. On the easiest dataset, SVAMP, we still achieve a 0.9-point improvement over the best-performing baseline and 2.9 points better than the reproduced PAL. On the MathQA dataset, known for its noise, we see significant improvements of over 20 points compared to other prompting baselines. The observed substantial enhancement suggests that the utilization of in-context examples facilitates the model's comprehension of the questions and enables it to generate solutions based on analogous prompts. These results suggest that retrieving similar examples is crucial, especially for complex datasets.

In addition to our prompting approach, we also evaluate the effectiveness of fine-tuning a smaller language model on the annotated training data, as shown in Table 2 (bottom section). We fine-tune a 6B CodeGen model on the training data with annotated programs, and our approach achieves better performance with 0.8-point improvement than an 11B T5 model on the GSM8K dataset. We use the same method as Magister et al. (2022) to perform prompting on the training set and obtain the annotated CoT. Notably, for the SVAMP dataset, our fine-tuning approach with programs significantly outperforms fine-tuning with natural language CoT by 7.8 points. On the MathQA dataset, which is known to have noisy labels, our fine-tuning performance is significantly better than vanilla prompting performance. The dynamic program prompting achieves over 30-point improvements compared with PAL. Compared with CoT Fine-tuning approach using CodeGen, our program distillation approach is also 25.3 points better in accuracy. This observation further highlights the importance of leveraging the training data in complex datasets. In general, the fine-tuning performance with smaller models is worse than few-shot prompting with large language models on GSM8K and SVAMP, indicating that program distillation may not be sufficient to compensate for the generalization limitations of smaller language models.

Prompt Retrieval Strategy To further justify the effectiveness of using the most similar exemplars as prompts, we conduct experiments with different prompt retrieval strategies and the results are presented in Table 3. "*Random*" strategy is to randomly select M exemplars as the prompt. The table shows that using different sentence embeddings results in consistent performance when using the "*most similar M Exemplar*" strategy. However, using the "*least similar exemplars*" consistently leads to a drop in performance, especially on the MathQA dataset where the evaluation data is more similar to the training data (Fu et al., 2022). Moreover, the least similar exemplars are unlikely to encompass the full spectrum of information required in the MathQA dataset where a broader range of knowledge exists. The "*Random*" strategy also shows similar performance as using the "*least similar exemplars*", indicating that neither of them provides additional benefits compared to using the "*most similar exemplars*" strategy.

Qualitative Prompt Analysis To gain insights into how the prompts affect performance, we compare the results between PAL and our approach on the GSM8K dataset. The retrieved prompts by our approach have a higher level of word level overlapping with the question. Figure 3 shows an example of how our approach helps in making more accurate predictions. The code "`* 100`" marked in red is the information that PAL failed to generate. This suggests that PAL may not have been confident about the "*percentage*" for this question. Our prompts, on the other hand, contain many questions related to "*percentage*" which are more likely to help the model make correct predictions. However, we also note that the similarity-based method is not always better than fixed prompts by PAL. On GSM8K, PAL still performs better on 5.5% of the questions while our similarity-based approach performs better on 10.3% of all questions. Thus, similarity-based prompts can produce positive improvements in general.

4 Related Work

Our work is mostly related to recent literature that incorporates the training data to improve the language model performance on downstream tasks. Chung et al. (2022) shows that we can benefit from additional CoT data for both large and small language models. Li et al. (2022) samples CoT reasoning paths for the training data and uses them to diversify the prompts on the GSM8K dataset. Alternatively, we can use the sampled CoT to further fine-tune the language models (Huang et al., 2022; Magister et al., 2022; Meng et al., 2022). In practice, we cannot guarantee the correctness of the sampled CoT, especially for the task of math word problem solving, which requires rigorous reasoning paths. Recent approaches (Magister et al., 2022; Wang et al., 2022b) attempt to reduce the negative effect by matching the answer with generated CoT or assigning different weights for the samples. Simultaneously with this study, Uesato et al. (2022) proposes to use step-based reward to improve the performance specifically on GSM8K. In order to do so, the authors need to annotate a portion the data to train the underlying reward model. However, these methods cannot completely avoid the underlying limitation as it is challenging the evaluate the step-by-step natural language CoT (Golovneva et al., 2022; Prasad et al., 2023). Our approach is inspired by program generation via few-shot prompting (Gao et al., 2022), we perform prompting on the training data and easily verify the answer correctness by executing the program, which allows us to obtain more reliable pseudo-gold programs.

5 Conclusion and Future Work

Motivated by program-based prompting (Gao et al., 2022; Drori et al., 2022), we are able to obtain the pseudo-gold program as the intermediate reasoning step for training data. We then present two approaches to make use of such data with program annotations in both of the few-shot prompting and fine-tuning scenarios. In few-shot prompting with LLMs, we sample similar exemplars as prompts for experiments. In the fine-tuning approach, we directly fine-tune a pre-trained language model on program-annotated data. Our experiments demonstrate both few-shot prompting and fine-tuning can significantly benefit from the training data annotated with programs, especially for complex problems in the MathQA dataset.

For future research, our goal is to design a structured model that leverages the potential of data with program annotations, particularly in light of the substantial underperformance of smaller language models. Interestingly, even with their limitations, structured models (Jie et al., 2022; Shao et al., 2022) have exhibited the capacity to outshine large language model prompting on MathQA. Additionally, the recent emergence of instruction-following models (Ouyang et al., 2022; Wang et al., 2022a), exemplified by Alpaca (Taori et al., 2023), has prompted our interest in equipping large language models with mathematical reasoning capacities (Wang and Lu, 2023) while maintaining the integrity of their underlying language understanding capabilities.

Limitations

The methods we have employed for prompting and fine-tuning have yielded noticeable improvements, yet certain limitations persist within practical applications. To achieve optimal performance, we continue to rely on prompting using large language models, which prove to be costly for the research community. Furthermore, retrieval efficiency may present a challenge when dealing with extensive training sets, as identifying the top M

exemplars for each example becomes increasingly time-consuming. Consequently, devising a more efficient algorithm to expedite the retrieval process represents a potential area for future exploration.

Despite the potential for performance improvement by sampling 40 reasoning paths for each question as presented by Wang et al. (2022a); Fu et al. (2022), we were unable to incorporate this approach due to budget constraints. Additionally, although training data has proven beneficial, the gains for smaller models are insufficient to surpass the performance of large language models. This observation may indicate the necessity for a fundamentally different model design or a superior pre-trained model (e.g., Galactica (Taylor et al., 2022) or Code-T5 (Wang et al., 2023)) as a more effective basis for fine-tuning.

Acknowledgement

We would like to thank the anonymous reviewers, our meta-reviewer, and senior area chairs for their constructive comments and support on our work. This research/project is supported by the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Program (AISG Award No: AISG2-RP-2020-016), and the Ministry of Education, Singapore, under its Tier 3 Programme (The Award No.: MOET32020-0004).

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of NAACL*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

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

Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, et al. 2022. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32):e2123433119.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2022. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of EMNLP*.

Olga Golovneva, Moya Chen, Spencer Poff, Martin Corredor, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. 2022. Roscoe: A suite of metrics for scoring step-by-step reasoning. *arXiv preprint arXiv:2212.07919*.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2022. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*.

Zhanming Jie, Jierui Li, and Wei Lu. 2022. Learning to reason deductively: Math word problem solving as complex relation extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5944–5955.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2022. On the advance of making language models better reasoners. *arXiv preprint arXiv:2206.02336*.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. 2022. Teaching small language models to reason. *arXiv preprint arXiv:2212.08410*.

Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. 2022. Generating training data with language models: Towards zero-shot language understanding. In *Proceedings of NeuIPS*.

Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. 2022. Lila: A unified benchmark for mathematical reasoning. In *Proceedings of EMNLP*.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. 2022. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*.

Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. 2022. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. In *Findings of the Association for Computational Linguistics: ACL 2022*.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *ArXiv preprint, abs/2203.13474*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

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

Archiki Prasad, Swarnadeep Saha, Xiang Zhou, and Mohit Bansal. 2023. Receval: Evaluating reasoning chains via correctness and informativeness. *arXiv preprint arXiv:2304.10703*.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2021. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633*.

Zhihong Shao, Fei Huang, and Minlie Huang. 2022. Chaining simultaneous thoughts for numerical reasoning. In *Proceedings of EMNLP*.

Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. 2022. Repository-level prompt generation for large language models of code. *arXiv preprint arXiv:2206.12839*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of NeurIPS*.

Tianduo Wang and Wei Lu. 2023. Learning multi-step reasoning by solving arithmetic tasks. In *Proceedings of ACL*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022a. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Awadallah, and Jianfeng Gao. 2022b. List: Lite prompted self-training makes parameter-efficient few-shot learners. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 2262–2281.

Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. 2023. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*.

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

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. In *Proceedings of NeurIPS*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Limitation

□ A2. Did you discuss any potential risks of your work?
Not applicable. Left blank.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract and Section 1

☑ A4. Have you used AI writing assistants when working on this paper?
Just use grammarly to check.

B ☑ Did you use or create scientific artifacts?

Left blank.

☑ B1. Did you cite the creators of artifacts you used?
Section 3

□ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Not applicable. Left blank.

□ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Not applicable. Left blank.

☑ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Section 3

□ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Table 1

C ☒ Did you run computational experiments?

Left blank.

□ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
No response.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

□ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
No response.

□ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
No response.

□ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
No response.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

□ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

□ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

□ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

□ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

□ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.