Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks

Anonymous authors
Paper under double-blind review

Abstract

Recently, there has been significant progress in teaching language models to perform step-by-step reasoning to solve complex numerical reasoning tasks. Chain-of-thoughts prompting (CoT) is the state-of-art method for many of these tasks. CoT uses language models to produce text describing reasoning, and computation, and finally the answer to a question. Here we propose 'Program of Thoughts' (PoT), which uses language models (mainly Codex) to generate text and programming language statements, and finally an answer. In PoT, the computation can be delegated to a program interpreter, which is used to execute the generated program, thus decoupling complex computation from reasoning and language understanding. We evaluate PoT on five math word problem datasets and three financial-QA datasets in both few-shot and zero-shot settings. We find that PoT has an average performance gain over CoT of around 12% across all datasets. By combining PoT with self-consistency decoding, we can achieve extremely strong performance on all the math datasets and financial datasets. All of our data and code will be released.

1 Introduction

Numerical reasoning is a long-standing task in artificial intelligence. A surge of datasets has been proposed recently to benchmark deep-learning models' capabilities to perform numerical/arithmetic reasoning. Some widely used benchmarks are based on Math word problems (MWP) Cobbe et al. (2021); Patel et al. (2021); Lu et al. (2022); Ling et al. (2017), where systems are supposed to answer math questions expressed with natural text. Besides MWP, some datasets also consider financial problems Chen et al. (2021b; 2022); Zhu et al. (2021), where systems need to answer math-driven financial questions.

Prior work Ling et al. (2017); Cobbe et al. (2021) has studied how to train models from scratch or fine-tune models to generate intermediate steps to derive the final answer. Such methods are data-intensive, requiring a significant number of training examples with expert-annotated steps. Recently, Wei et al. (2022) have discovered that the large language models (LLMs) Brown et al. (2020); Chen et al. (2021a); Chowdhery et al. (2022) can be prompted with a few input-output exemplars to solve these tasks without any training or fine-tuning. In particular, when prompted with a few examples containing inputs, natural language 'rationales', and outputs, LLMs can imitate the demonstrations to both generate rationales and answer these questions. Such a prompting method is dubbed 'Chain of Thoughts (CoT)', and it is able to achieve state-of-the-art performance on a wide spectrum of textual and numerical reasoning datasets.

CoT uses LLMs for both reasoning and computation, i.e. the language model not only needs to generate the mathematical expressions but also needs to perform the computation in each step. We argue that language models are not ideal for actually solving these mathematical expressions, because: 1) LLMs are very prone to arithmetic calculation errors, especially when dealing with large numbers; 2) LLMs cannot solve complex mathematical expressions like polynomial equations or even differential equations; 3) LLMs are highly inefficient at expressing iteration, especially when the number of iteration steps is large.

In order to solve these issues, we propose program-of-thoughts (PoT) prompting, which will delegate computation steps to an external language interpreter. In PoT, LMs can express reasoning steps as Python programs, and the computation can be accomplished by a Python interpreter. We depict the difference

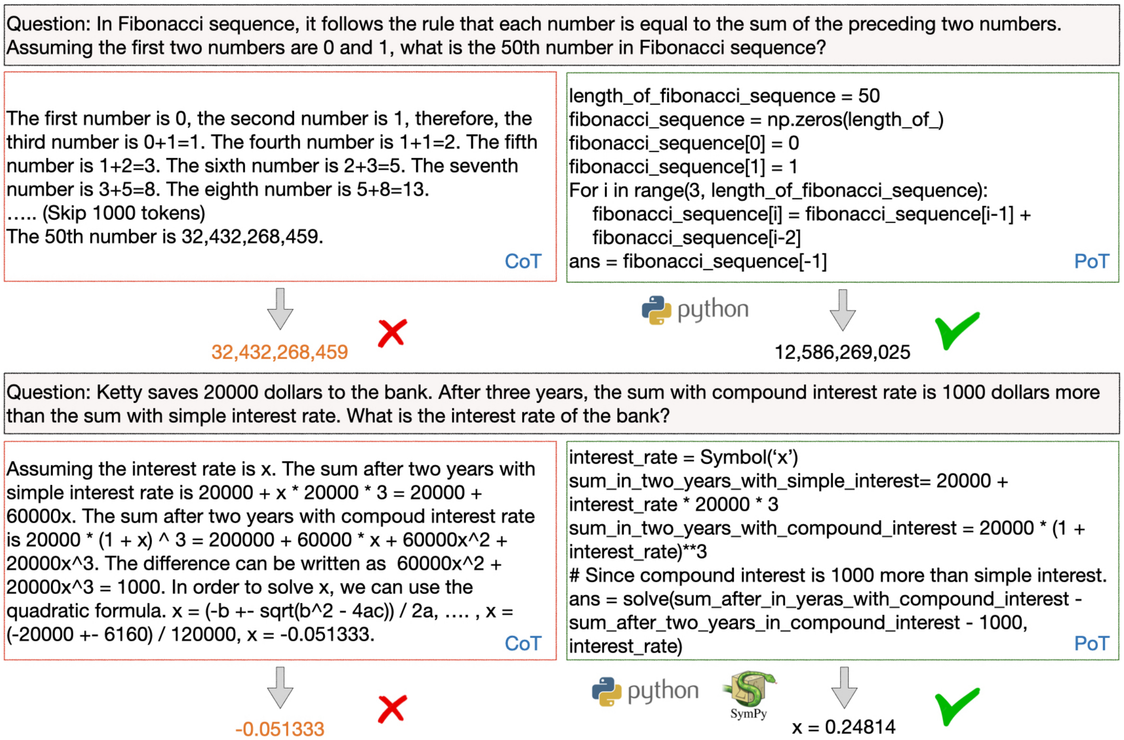

Figure 1: Comparison between Chain of Thoughts and Program of Thoughts.

between CoT and PoT in Figure 1. In the upper example, for CoT the iteration runs for 50 times, which leads to extremely low accuracy;[1] in the lower example, CoT cannot solve the cubic equation with language models and outputs a wrong answer. In contrast, in the upper example, PoT can express the iteration process with a few lines of code, which can be executed on a Python interpreter to derive an accurate answer; and in the lower example, PoT can convert the problem into a program that relies on 'SymPy' library in Python to solve the complex equation.

We evaluate PoT prompting across five MWP datasets, GSM8K, AQuA, SVAMP, TabMWP, MultiArith; and three financial datasets, FinQA, ConvFinQA, and TATQA. These datasets cover various input formats including text, tables, and conversation. We give an overview of the results in Figure 2. Under both few-shot and zero-shot settings, PoT outperforms CoT significantly across all the evaluated datasets. Under the few-shot setting, the average gain over CoT is around 8% for the MWP datasets and 15% for the financial datasets. Under the zero-shot setting, the average gain over CoT is around 12% for the MWP datasets. PoT combined with self-consistency (SC) also outperforms CoT+SC Wang et al. (2022b) by an average of 10% across all datasets. Our PoT+SC achieves the best-known results on all the evaluated MWP datasets and near best-known results on the financial datasets (excluding GPT-4 OpenAI (2023)). Finally, we conduct comprehensive ablation studies to understand the different components of PoT.

2 Program of Thoughts

2.1 Preliminaries

In-context learning has been described in Brown et al. (2020); Chen et al. (2021a); Chowdhery et al. (2022); Rae et al. (2021). Compared with fine-tuning, in-context learning (1) only takes a few annotations/demonstrations as a prompt, and (2) performs inference without training the model parameters. With in-context learning, LLMs receive the input-output exemplars as the prefix, followed by an input problem, and generate outputs imitating the exemplars. More recently, 'chain of thoughts prompting' Wei et al.

[1]Assuming each addition is correct with 90% chance, after 50 additions, the likelihood of a correct output is less than 1%.

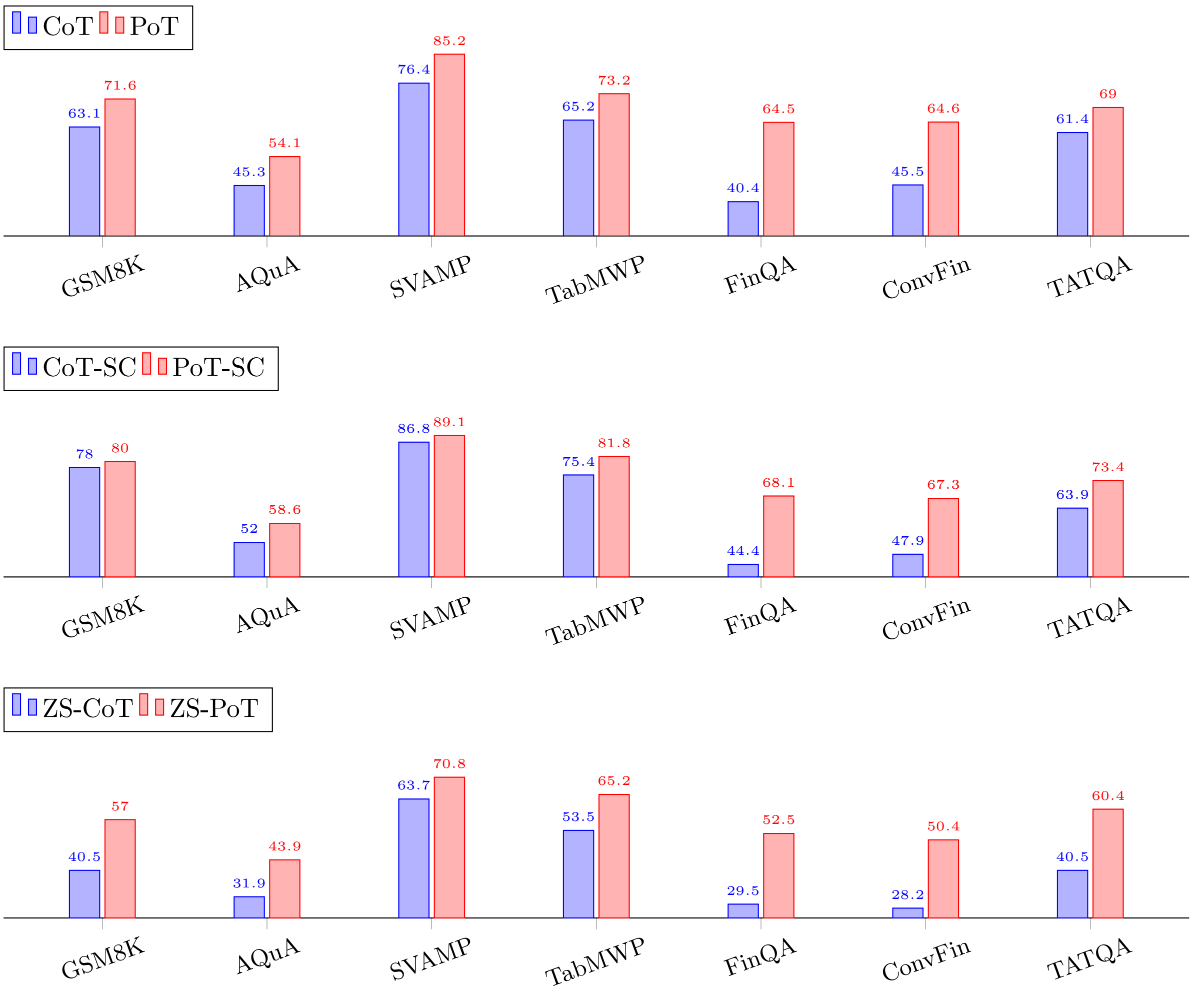

Figure 2: Few-shot (upper), Few-shot + SC (middle) and Zero-Shot (lower) Performance overview of Codex PoT and Codex CoT across different datasets.

(2022) has been proposed as a specific type of in-context learning where the exemplar's output contains the 'thought process' or rationale instead of just an output. This approach has been shown to elicit LLMs' strong reasoning capabilities on various kinds of tasks.

2.2 Program of Thoughts

Besides natural language, programs can also be used to express our thought processes. By using semantically meaningful variable names, a program can also be a natural representation to convey human thoughts. For example, in the lower example in Figure 1, we first create an unknown variable named `interest_rate`. Then we bind 'summation in two years with ... interest rate' to the variable `sum_in_two_years_with_XXX_interest` and write down the equation expressing their mathematical relations with `interest_rate`. These equations are packaged into the 'solve' function provided by 'SymPy'. The program is executed with Python to solve the equations to derive the answer variable `interest_rate`.

Unlike CoT, PoT relegates some computation to an external process (a Python interpreter). The LLMs are only responsible for expressing the 'reasoning process' in the programming language. In contrast, CoT aims to use LLMs to perform both reasoning and computation. We argue that such an approach is more expressive and accurate.

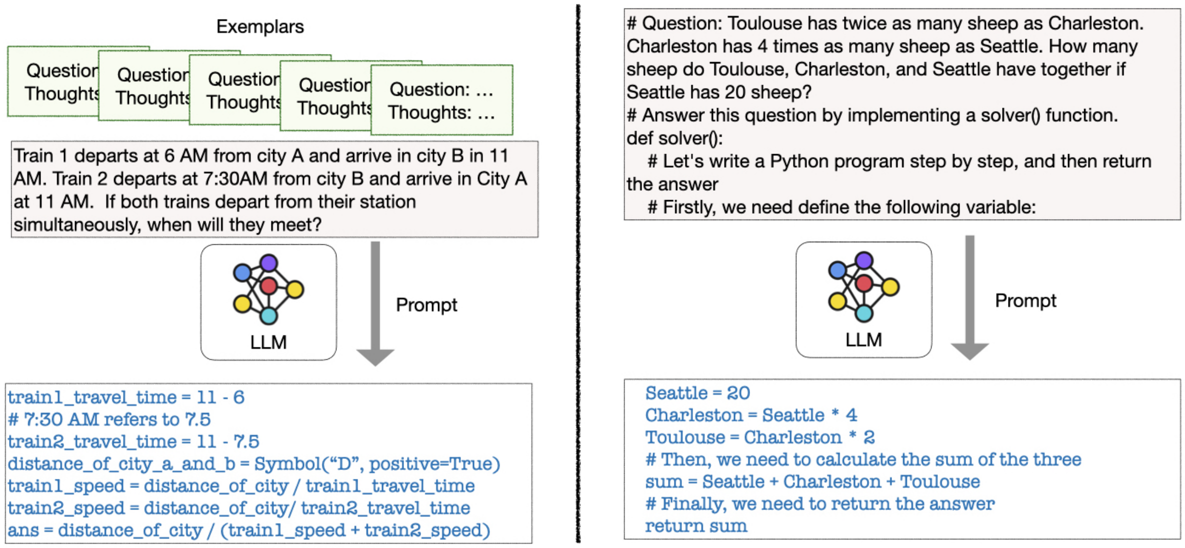

Figure 3: Left: Few-shot PoT prompting, Right: Zero-shot PoT prompting.

The 'program of thoughts' is different from generating equations directly, where the generation target would be $\texttt{solve}(20000*(1+x)^3-2000-x*20000*3-1000,x)$. As observed by Wei et al. (2022) for CoT, directly generating such equations is challenging for LLMs. PoT differs from equation generation in two aspects: (1) PoT breaks down the equation into a multi-step 'thought' process, and (2) PoT binds semantic meanings to variables to help ground the model in language. We found that this sort of 'thoughtful' process can elicit language models' reasoning capabilities and generate more accurate programs. We provide a detailed comparison in the experimental section.

We show the proposed PoT prompting method in Figure 3 under the few-shot and zero-shot settings. Under the few-shot setting, a few exemplars of (question, 'program of thoughts') pairs will be prefixed as demonstrations to teach the LLM how to generate 'thoughtful' programs. Under the zero-shot setting, the prompt only contains an instruction without any exemplar demonstration. Unlike zero-shot CoT Kojima et al. (2022), which requires an extra step to extract the answer from the 'chain of thoughts', zero-shot PoT can return the answer straightforwardly without extra steps.

2.3 PoT as an Intermediate Step

For certain problems requiring additional commonsense reasoning, we need to first utilize PoT to generate a program to compute an intermediate result, which is combined with the question to continue prompting LLM to derive the final answer. For instance, in the left example in Figure 3, the program will generate a float number 2.05. However, adding 2.05 to 11 AM cannot be easily handled by a Python program. Therefore, we continue to prompt the LLM to perform an additional step of textual reasoning to derive the final answer. We adopt this approach for the AQuA dataset.

3 Experiments

3.1 Experimental Setup

Datasets We summarize our evaluated datasets in Table 1. We use the test set for all the evaluated datasets except TATQA. These datasets are highly heterogeneous in terms of their input formats. We conduct comprehensive experiments on this broad spectrum of datasets to show the generalizability and applicability of PoT prompting.

To incorporate the diverse inputs, we propose to linearize these inputs in the prompt. For table inputs, we adopt the same strategy as Chen (2022) to linearize a table into a text string. The columns of the table are

Dataset	Split	Example	Domain	Input	Output
GSM8K Cobbe et al. (2021)	Test	1318	MWP	Question	Number
AQuA Ling et al. (2017)	Test	253	MWP	Question	Option
SVAMP Patel et al. (2021)	Test	1000	MWP	Question	Number
MultiArith Roy & Roth (2015)	Test	600	MWP	Question	Number
TabMWP Lu et al. (2022)	Test	7861	MWP	Table + Question	Number + Text
FinQA Chen et al. (2021b)	Test	1147	Finance	Table + Text + Question	Number + Binary
ConvFinQA Chen et al. (2022)	Test	421	Finance	Table + Text + Conversation	Number + Binary
TATQA Zhu et al. (2021)	Dev	1668	Finance	Table + Text + Question	Number + Text

Table 1: Summarization of all the datasets being evaluated.

separated by '|' and the rows are separated by '\n'. If a table cell is empty, it is filled by '-'. For text+table hybrid inputs, we separate tables and text with '\n'. For conversational history, we also separate conversation turns by '\n'. The prompt is constructed by the concatenation of task instruction, text, linearized table, and question. For conversational question answering, we simply concatenate all the dialog history in the prompt.

Implementation Details We use the OpenAI Codex (code-davinci-002) API[2] and GPT-3 (text-davinci-002) API[3] model in our experiments. We use Python 3.8 with the SymPy library[4] to execute the generated program. For the few-shot setting, we use 4-8 shots for all the datasets, based on their difficulty. For simple datasets like FinQA Chen et al. (2021b), we tend to use less shots, while for more challenging datasets like AQuA Ling et al. (2017) and TATQA Zhu et al. (2021), we use 8 shots to cover more diverse problems. The examples are taken from the training set. We generally write prompts for 10-20 examples, and then tune the exemplar selection on a small validation set to choose the best 4-8 shots for the full set evaluation.

To elicit the LLM's capability to perform multi-step reasoning, we use the text "Let's write a Python program step by step" as our prompt. However, a caveat is that LLM can fall back to generating a reasoning chain in comments rather than in program. Therefore, we suppress the '#' token logits by -2 to decrease its probability to avoid such cases. We found that this simple strategy can greatly improve performance.

Metrics We adopt exact match scores as our evaluation metrics for GSM8K, SVAMP, and MultiArith datasets. We will round the predicted number to a specific precision and then compare it with the reference number. For the AQuA dataset, we use PoT to compute the intermediate answer and then prompt the LLM again to output the closest option to measure the accuracy. For TabMWP, ConvFinQA, and TATQA datasets, we use the official evaluation scripts provided on Github. For FinQA, we relax the evaluation for CoT because LLMs cannot perform the computation precisely (especially with high-precision floats and large numbers), so we adopt 'math.isclose' with relative tolerance of 0.001 to compare answers.

Baselines We report results for three different models including Codex Chen et al. (2021a), GPT-3 Brown et al. (2020), PaLM Chowdhery et al. (2022) and LaMDA Thoppilan et al. (2022). We consider two types of prediction strategies including direct answer output and chain of thought to derive the answer. Since PaLM API is not public, we only list PaLM results reported from previous work Wei et al. (2022); Wang et al. (2022b). We also leverage an external calculator as suggested in Wei et al. (2022) for all the equations generated by CoT, which is denoted as CoT + calc. Besides greedy decoding, we use self-consistency Wang et al. (2022b) with CoT, taking majority vote over 40 different completions as the prediction.

3.2 Main Results

Few-shot Results We give our few-shot results in Table 2. On MWP datasets, PoT with greedy decoding improves on GSM8K/AQuA/TabMWP by more than 8%. On SVAMP, the improvement is 4% mainly due to its simplicity. For financial QA datasets, PoT improves over CoT by roughly 20% on FinQA/ConvFinQA

[2]`https://openai.com/blog/openai-codex/`
[3]`https://beta.openai.com/`
[4]`https://www.sympy.org/en/index.html`

Model	#Params	GSM8K	AQuA	SVAMP	TabWMP	FinQA	ConvFin	TATQA	Avg
Fine-tuned or few-shot prompt									
Published SoTA	-	78.0	52.0	86.8	68.2	68.0	**68.9**	**73.6**	70.7
Few-shot prompt (Greedy Decoding)									
Codex Direct	175B	19.7	29.5	69.9	59.4	25.6	40.0	55.0	42.7
Codex CoT	175B	63.1	45.3	76.4	65.2	40.4	45.6	61.4	56.7
GPT-3 Direct	175B	15.6	24.8	65.7	57.1	14.4	29.1	37.9	34.9
GPT-3 CoT	175B	46.9	35.8	68.9	62.9	26.1	37.4	42.5	45.7
PaLM Direct	540B	17.9	25.2	69.4	-	-	-	-	-
PaLM CoT	540B	56.9	35.8	79.0	-	-	-	-	-
Codex CoT$_{calc}$	175B	65.4	45.3	77.0	65.8	-	-	-	-
GPT-3 CoT$_{calc}$	175B	49.6	35.8	70.3	63.4	-	-	-	-
PaLM CoT$_{calc}$	540B	58.6	35.8	79.8	-	-	-	-	-
PoT (Ours)	175B	71.6	54.1	85.2	73.2	64.5	64.6	69.0	68.9
Few-shot prompt (Self-Consistency Decoding)									
LaMDA CoT-SC	137B	27.7	26.8	53.5	-	-	-	-	-
Codex CoT-SC	175B	78.0	52.0	86.8	75.4	44.4	47.9	63.2	63.9
PaLM CoT-SC	540B	74.4	48.3	86.6	-	-	-	-	-
PoT-SC (Ours)	175B	**80.0**	**58.6**	**89.1**	**81.8**	**68.1**	67.3	70.2	**73.6**

Table 2: The few-shot results for different datasets. Published SoTA includes the best-known results. On GSM8K, AQuA and SVAMP, the prior SoTA results are CoT + self-consistency decoding Wang et al. (2022b). On FinQA, the prior best result is from Wang et al. (2022a). On ConvFinQA, the prior best result is achieved by FinQANet Chen et al. (2022). On TabWMP Lu et al. (2022), the prior best result is achieved by Dynamic Prompt Learning Lu et al. (2022). On TATQA, the SoTA result is achieved by RegHNT Lei et al. (2022).

and 8% on TATQA. The larger improvements in FinQA and ConvFinQA are mainly due to miscalculations on LLMs for large numbers (e.g. in the millions). CoT adopts LLMs to perform the computation, which is highly prone to miscalculation errors, while PoT adopts a highly precise external computer to solve the problem. As an ablation, we also compare with CoT+calc, which leverages an external calculator to correct the calculation results in the generated 'chain of thoughts'. The experiments show that adding an external calculator only shows mild improvement over CoT on MWP datasets, much behind PoT.

Few-shot + Self-Consistency Results We leverage self-consistency (SC) decoding to understand the upper bound of our method. This sampling-based decoding algorithm can greatly reduce randomness in the generation procedure and boosts performance. Specifically, we set a temperature of 0.4 and K=40 throughout our experiments. According to to Table 2, we found that PoT + SC still outperforms CoT + SC on MWP datasets with notable margins. On financial datasets, we observe that self-consistency decoding is less impactful for both PoT and CoT. Similarly, PoT + SC outperforms CoT + SC by roughly 20% on FinQA/ConvFinQA, and 7% on TATQA.

Zero-shot Results We also evaluate the zero-shot performance of PoT and compare with Kojima et al. (2022) in Table 3. As can be seen, zero-shot PoT significantly outperforms zero-shot CoT across all the MWP datasets evaluated. Compared to few-shot prompting, zero-shot PoT outperforms zero-shot CoT Kojima et al. (2022) by an even larger margin. On the evaluated datasets, PoT's outperforms CoT by an average of 12%. On TabMWP, zero-shot PoT is even higher than few-shot CoT. These results show the great potential to directly generalize to many unseen numerical tasks even without any dataset-specific exemplars.

Model	#Params	GSM8K	AQuA	SVAMP	TabMWP	MultiArith	Avg
Zero-shot Direct (GPT-3)	150B	12.6	22.4	58.7	38.9	22.7	31.0
Zero-shot CoT (GPT-3)	150B	40.5	31.9	63.7	53.5	79.3	53.7
Zero-shot CoT (PaLM)	540B	43.0	-	-	-	66.1	-
Zero-shot PoT (Ours)	150B	**57.0**	**43.9**	**70.8**	**66.5**	**92.2**	**66.1**

Table 3: The zero-shot results for different datasets. The baseline results are taken from Kojima et al. (2022).

Model	GSM8K	SVAMP	FinQA
Codex CoT	63.1	76.4	40.4
GPT3 CoT	46.9	58.9	26.1
Codex - GPT3 (CoT)	+16.2	+7.5	+14.3
Codex PoT	71.6	85.2	64.5
GPT3 PoT	60.4	80.1	56.7
Codex - GPT3 (PoT)	+11.2	+5.1	+7.8

Table 4: GPT-3 and Codex performance difference under CoT and PoT prompting.

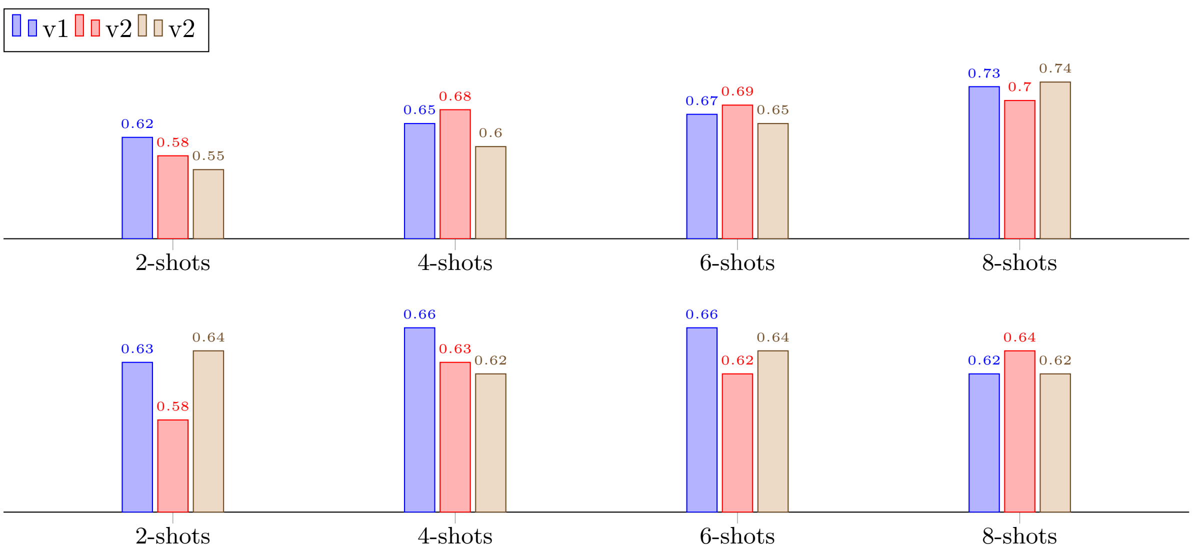

Figure 4: Exemplar sensitivity analysis for GSM8K and FinQA, where v1, v2 and v3 are three versions of k-shot demonstration sampled from the pool.

3.3 Ablation Studies

We performed multiple ablation studies under the few-shot setting to understand the importance of different factors in PoT.

Backend GPT-3 vs. Codex We evaluated GPT-3 (text-davinci-002) with PoT prompting. Unlike Codex, GPT-3 is not optimized for generating programs, and one would expect degraded performance with GPT-3 as the LLM. We choose three datasets, GSM8K, SVAMP, and FinQA, to analyze the performance difference of PoT and compare that relative to CoT. We show our experimental results in Table 4. We can see that the gap between Codex and GPT-3 with PoT is consistently smaller than their gap with CoT. We conclude that our prompting approach is still effective for models that are not specifically optimized for program generation. However, we do observe that the gap increases as the task becomes more challenging.

Sensitivity to Exemplars To better understand how sensitive PoT is w.r.t different exemplars, we conduct a sensitivity analysis. Specifically, we wrote 20 total exemplars. For k-shot learning, we randomly

Method	GSM8K	SVAMP	FinQA
PoT	71.6	85.2	64.5
PoT - Binding	60.2	83.8	61.6
PoT - MultiStep	45.8	81.9	58.9

Table 5: Comparison between PoT and equation generation on three different datasets.

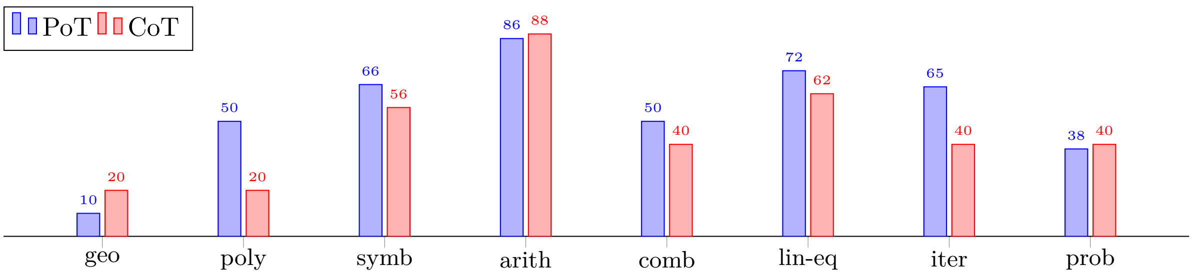

Figure 5: PoT and CoT's breakdown accuracy across different types of questions.

sample k = (2, 4, 6, 8) out of the 20 exemplars three times as v1, v2, and v3. We will use these randomly sampled exemplars as demonstrations for PoT. We summarize our sensitivity analysis in Figure 4. First of all, we found that increasing the number of shots helps more for GSM8K than FinQA. This is mainly due to the diversity of questions in GSM8K. By adding more exemplars, the language models can better generalize to diverse questions. Another observation is that when given fewer exemplars, PoT's performance variance is larger. When K=2, the performance variance can be as large as 7% for both datasets. With more exemplars, the performance becomes more stable.

Semantic Binding and Multi-Step Reasoning The two core properties of 'program of thoughts' are: (1) multiple steps: breaking down the thought process into the step-by-step program, (2) semantic binding: associating semantic meaning to the variable names. To better understand how these two properties contribute, we compared with two variants. One variant is to remove the semantic binding and simply use a, b, c as the variable names. The other variant is to directly predict the final mathematical equation to compute the results. We show our findings in Table 5. As can be seen, removing the binding will in general hurt the model's performance. On more complex questions involving more variables like GSM8K, the performance drop is larger. Similarly, prompting LLMs to directly generate the target equations is also very challenging. Breaking down the target equation into multiple reasoning steps helps boost performance.

Breakdown Analysis We perform further analysis to determine which kinds of problems CoT and PoT differ most in performance. We use AQuA Ling et al. (2017) as our testbed for this. Specifically, we manually classify the questions in AQuA into several categories including geometry, polynomial, symbolic, arithmetic, combinatorics, linear equation, iterative and probability. We show the accuracy for each subcategory in Figure 5. The major categories are (1) linear equations, (2) arithmetic, (3) combinatorics, (4) probability, and (5) iterative. The largest improvements of PoT are in the categories 'linear/polynomial equation', 'iterative', 'symbolic', and 'combinatorics'. These questions require more complex arithmetic or symbolic skills to solve. In contrast, on 'arithmetic', 'probability', and 'geometric' questions, PoT and CoT are performing quite similarly. Such observation reflects our assumption that 'program' is more effective on challenging problems.

Error Analysis We considered two types of errors: (1) value grounding error, and (2) logic generation error. The first type indicates that the model fails to assign correct values to the variables relevant to the question. The second type indicates that the model fails to generate the correct computation process to answer the question based on the defined variables. Figure 6 shows an example of each type of error. In the upper example, the model fetches the value of the variables incorrectly while the computation logic is correct. In the lower example, the model grounded relevant variables correctly but fails to generate proper computation logic to answer the question. We manually examined the errors made in the TAT-QA results. Among the 198 failure cases of numerical reasoning question with the PoT (greedy) method, 47% have value

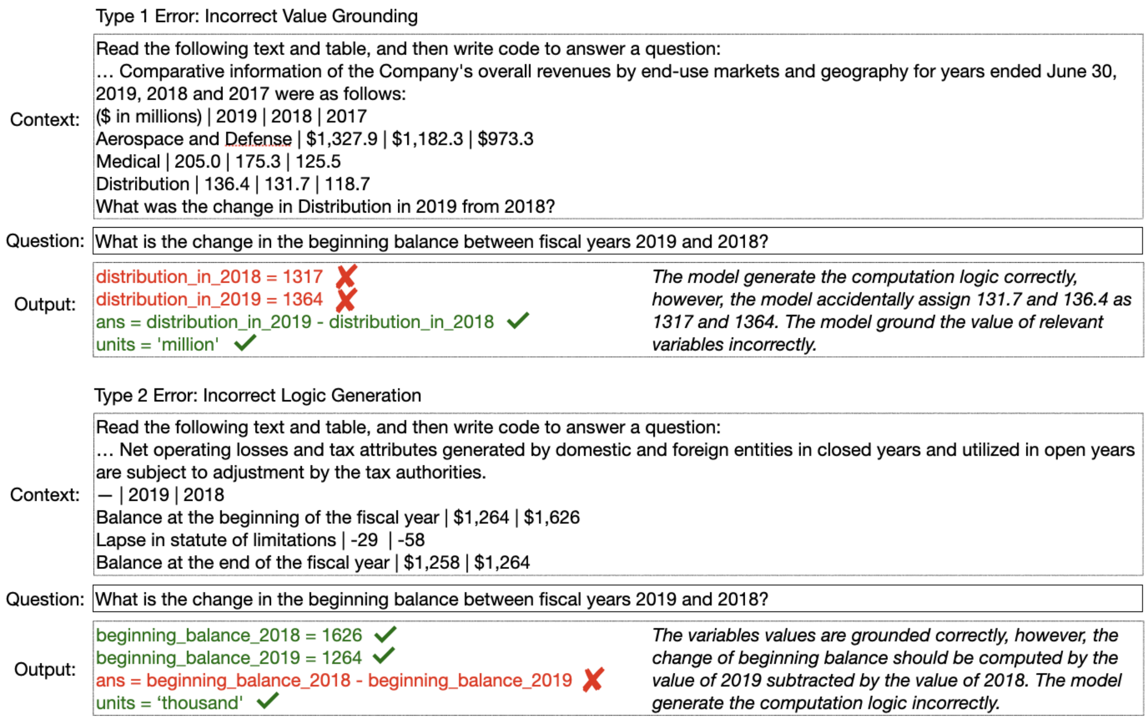

Figure 6: Error cases on TAT-QA dev set using PoT-greedy method.

grounding errors and 33% have logic errors. In 15% both types of errors occurred and in 5% we believe the answer is actually correct. We found that the majority of the errors are value grounding errors, which is also common for other methods such as CoT.

4 Related Work

4.1 Mathematical Reasoning in NLP

Mathematical reasoning skills are essential for general-purpose intelligent systems, which have attracted a significant amount of attention from the community. Earlier, there have been studies in understanding NLP models' capabilities to solve arithmetic/algebraic questions Hosseini et al. (2014); Koncel-Kedziorski et al. (2015); Roy & Roth (2015); Ling et al. (2017); Roy & Roth (2018). Recently, more challenging datasets Dua et al. (2019); Saxton et al. (2019); Miao et al. (2020); Amini et al. (2019); Hendrycks et al. (2021); Patel et al. (2021) have been proposed to increase the difficulty, diversity or even adversarial robustness. LiLA Mishra et al. (2022) proposes to assemble a large set of mathematical datasets into a unified dataset. LiLA also annotates Python programs as the generation target for solving mathematical problems. However, LiLA Mishra et al. (2022) is mostly focused on dataset unification. Our work aims to understand how to generate 'thoughtful programs' to best elicit LLM's reasoning capability. Besides, we also investigate how to solve math problems without any exemplars. Austin et al. (2021) propose to evaluate LLMs' capabilities to synthesize code on two curated datasets MBPP and MathQA-Python.

4.2 In-context Learning with LLMs

GPT-3 Brown et al. (2020) demonstrated a strong capability to perform few-shot predictions, where the model is given a description of the task in natural language with few examples. Scaling model size, data, and computing are crucial to enable this learning ability. Recently, Rae et al. (2021); Smith et al. (2022); Chowdhery et al. (2022); Du et al. (2022) have proposed to train different types of LLMs with different training recipes. The capability to follow few-shot exemplars to solve unseen tasks is not existent on smaller

LMs, but only emerge as the model scales up Kaplan et al. (2020). Recently, there have been several works Xie et al. (2021); Min et al. (2022) aiming to understand how and why in-context learning works. Another concurrent work similar to ours is BINDER Cheng et al. (2022), which applies Codex to synthesize 'soft' SQL queries to answer questions from tables.

4.3 Chain of Reasoning with LLMs

Although LLMs have demonstrated remarkable success across a range of NLP tasks, their ability to reason is often seen as a limitation. Recently, CoT Wei et al. (2022); Kojima et al. (2022); Wang et al. (2022b) was proposed to enable LLM's capability to perform reasoning tasks by demonstrating 'natural language rationales'. Suzgun et al. (2022) have shown that CoT can already surpass human performance on challenging BIG-Bench tasks. Later on, several other works Drozdov et al. (2022); Zhou et al. (2022); Nye et al. (2021) also propose different approaches to utilize LLMs to solve reasoning tasks by allowing intermediate steps. Our method can be seen as an extension to CoT by leveraging an extra step of symbolic execution. Another contemporary work Gao et al. (2022) was proposed at the same time as ours to adopt hybrid text/code reasoning to address math questions.

5 Conclusions

In this work, we investigate how to disentangle computation from reasoning in solving numerical problems. By 'program of thoughts' prompting, we are able to elicit LLMs' abilities to generate accurate programs to express complex reasoning procedure, while also allows computation to be separately handled by an external program interpreter. This approach is able to boost the performance of LLMs on several math datasets significantly. We believe our work can inspire more work to combine symbolic execution with LLMs to achieve better performance on other symbolic reasoning tasks.

Limitations

Our work aims at combining LLM with symbolic execution to solve challenging math problems. PoT would require execution of 'generated code' from LLMs, which could contain certain dangerous or risky code snippets like 'import os; os.rmdir()', etc. We have blocked the LLM from importing any additional modules and restrict it to using the pre-defined modules. Such brutal-force blocking works reasonable for math QA, however, for other unknown symbolic tasks, it might hurt the PoT's generalization. Another limitation is that PoT still struggles with AQuA dataset with complex algebraic questions with only 58% accuracy. It's mainly due to the diversity questions in AQuA, which the demonstration cannot possibly cover. Therefore, the future research should discuss how to further prompt LLMs to generate code for highly diversified Math questions.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2357–2367, 2019.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021a.

Wenhu Chen. Large language models are few (1)-shot table reasoners. *arXiv preprint arXiv:2210.06710*, 2022.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R Routledge, et al. Finqa: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3697–3711, 2021b.

Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. Convfinqa: Exploring the chain of numerical reasoning in conversational finance question answering. *arXiv preprint arXiv:2210.03849*, 2022.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*, 2022.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. Compositional semantic parsing with large language models. *arXiv preprint arXiv:2209.15003*, 2022.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 2022.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, 2019.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*, pp. 523–533, 2014.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

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

Fangyu Lei, Shizhu He, Xiang Li, Jun Zhao, and Kang Liu. Answering numerical reasoning questions in table-text hybrid contents with graph-based encoder and tree-based decoder. In *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 1379–1390, 2022.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 158–167, 2017.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. *arXiv preprint arXiv:2209.14610*, 2022.

Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. 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*, pp. 975–984, 2020.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. Lila: A unified benchmark for mathematical reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

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

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL `https://aclanthology.org/2021.naacl-main.168`.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1743–1752, 2015.

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

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

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

Bin Wang, Jiangzhou Ju, Yunlin Mao, Xin-Yu Dai, Shujian Huang, and Jiajun Chen. A numerical reasoning question answering system with fine-grained retriever and the ensemble of multiple generators for finqa. *arXiv preprint arXiv:2206.08506*, 2022a.

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

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

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2021.

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

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3277–3287, 2021.