MathPrompter: Mathematical Reasoning using Large Language Models

Shima Imani, Liang Du, Harsh Shrivastava
Microsoft Research, Redmond, USA
Contact: shimaimani@microsoft.com

Abstract

Large Language Models (LLMs) have limited performance when solving arithmetic reasoning tasks and often provide incorrect answers. Unlike natural language understanding, math problems typically have a single correct answer, making the task of generating accurate solutions more challenging for LLMs. To the best of our knowledge, we are not aware of any LLMs that indicate their level of confidence in their responses which fuels a trust deficit in these models impeding their adoption. To address this deficiency, we propose 'MathPrompter', a technique that improves performance of LLMs on arithmetic problems along with increased reliance in the predictions. MathPrompter uses the Zero-shot chain-of-thought prompting technique to generate multiple Algebraic expressions or Python functions to solve the same math problem in different ways and thereby raise the confidence level in the output results. This is in contrast to other prompt based CoT methods, where there is no check on the validity of the intermediate steps followed. Our technique improves over state-of-the-art on the MultiArith dataset (78.7% $\rightarrow$ 92.5%) evaluated using 175B parameter GPT-based LLM.

1 Introduction

Recent advancements in natural language processing (NLP) can be attributed to massive scaling of Large Language Models (LLMs) (Vaswani et al., 2017; Devlin et al., 2018; Raffel et al., 2020; Brown et al., 2020; Rae et al., 2021; Chowdhery et al., 2022; Thoppilan et al., 2022). A very interesting recent discovery that the LLMs are naturally good (in-context) Zero-shot or few-shot learners turned out to be very useful (Brown et al., 2020; Liu et al., 2021, 2023). This led to the development of 'prompting' technique, where the user provides a small context for solving the task at-hand to the LLM. This conditioning of the models on a few examples is termed as few-shot prompting, while providing instructions to solve a task is known as Zero-shot prompting. Extensive research efforts are being poured into designing these prompts, either manually (Schick and Schütze, 2020; Reynolds and McDonell, 2021) or automatically (Shin et al., 2020; Gao et al., 2020). Although quite successful for single-step system-I tasks (Stanovich and West, 2000; Liu et al., 2023), the prompting techniques were inadequate in their performance on system-II tasks where multi-step reasoning is required (Rae et al., 2021). As humans, we tend to break down a problem and attempt to solve them step-by-step. Extending this intuition to LLMs led to the development of 'chain-of-thought' (CoT) prompting technique (Wei et al., 2022; Wang et al., 2022). The use of CoT has led to improved performance on a range of NLP tasks (Talmor et al., 2018; Gao et al., 2020; Patel et al., 2021; Cobbe et al., 2021; Geva et al., 2021; Chowdhery et al., 2022; Srivastava et al., 2022)

In this work, we investigate Zero-shot-CoT methods for solving mathematical reasoning tasks. To the best of our knowledge, we found the recent work by (Kojima et al., 2022) that proposed a Zero-shot-CoT technique to be the state-of-the-art where they demonstrated a remarkable accuracy improvement on the 'MultiArith' (Roy and Roth, 2016) data (17.7% $\rightarrow$ 78.7%). Now, we identify two key aspects that lacks in the previous CoT prompting based SOTA, namely (1) Although, the chain-of-thought followed by the model improved the results, but there is **no check on the validity of the steps followed** by the chain-of-thought prompting and (2) The **confidence in the predictions** of LLMs are often not provided. In order to address these gap to some extent, we derive inspiration from how we humans solve a math question by breaking it down to a simpler multi-step procedure and make use of multiple ways to validate our approach at each step. Specifically, given a question Q, (I) *Generating Algebraic template*: We first gen-

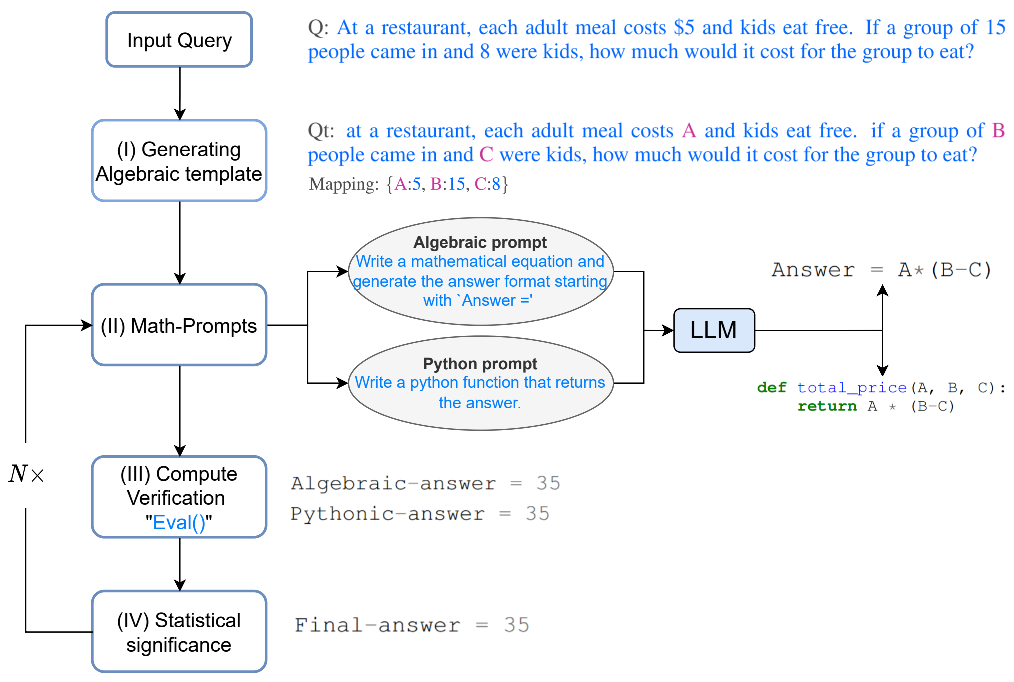
Input Query
Q: At a restaurant, each adult meal costs $5 and kids eat free. If a group of 15 people came in and 8 were kids, how much would it cost for the group to eat?
(I) Generating Algebraic template
Qt: at a restaurant, each adult meal costs A and kids eat free. if a group of B people came in and C were kids, how much would it cost for the group to eat?
Mapping: {A:5, B:15, C:8}
(II) Math-Prompts
Algebraic prompt
Write a mathematical equation and generate the answer format starting with `Answer ='
Python prompt
Write a python function that returns the answer.
LLM
Answer = A*(B-C)
def total_price(A, B, C):
return A * (B-C)
N×
(III) Compute Verification "Eval()"
Algebraic-answer = 35
Pythonic-answer = 35
(IV) Statistical significance
Final-answer = 35

Figure 1: **MathPrompter flow.** We outline the MathPrompter process with an example alongside.

erate its corresponding Algebraic expression Qt that replaces the numerical entries by variables. (II) *Math-prompts*: Then, we provide multiple prompts P to the LLM that can solve Qt analytically in different ways. For eg. P can be 'Derive an Algebraic expression' or 'Write a Python function' etc. Following this procedure, we end up with P expressions that analytically solves Qt in terms of its variables. (III) *Compute verification*: We then evaluate the P analytical solutions by allotting multiple random values to the Qt variables. (IV) *Statistical significance*: If the solutions of the P analytical functions are in '*consensus*' over $N \sim 5$ different variable choices, then we substitute the original values from Q to obtain the final solution. In the case where there is no definite consensus, we repeat the steps (II), (III) & (IV). Our method, MathPrompter, uses 175B parameter LLM called GPT3 DaVinci completion engine (Brown et al., 2020). We were able to improve the accuracy on the MultiArith data from $78.7\% \rightarrow 92.5\%$.

2 Method

Since the LLMs are generative models, it becomes very tricky to ensure that the generated answers are accurate, especially for mathematical reasoning tasks. We take clues from the process followed by students to solve arithmetic problems. We narrowed down a few steps that students take in order to verify their solutions, namely

- *Compliance with known results*: By comparing the solution to a known result, one can assess its accuracy and make necessary adjustments. This is particularly useful when the question is a standard problem with a well-established solution.

- *Multi-verification*: By approaching a problem from multiple perspectives and comparing the results helps to confirm the validity of the solution and ensure that it is both sound and accurate.

- *Cross-checking*: The process of solving a problem is just as necessary as the final answer. Verifying the correctness of the intermediate steps of the process provide a clear understanding of the thought process behind the solution.

- *Compute verification*: Utilizing a calculator or computer to perform arithmetic calculations can assist in verifying the accuracy of the final answer.

2.1 MathPrompter

Our proposed method, MathPrompter, is an attempt to transfer some of this thought process to the LLM answer generation process. Fig. 1 provides a high-level overview of steps followed by MathPrompter to solve a mathematical reasoning problem. We use the state-of-the-art GPT-3 DaVinci completion engine (Brown et al., 2020) for the question-answering tasks.

We use the following question 'Q' from the MultiArith dataset to demonstrate the problem solving process followed by MathPrompter.

> Q: At a restaurant, each adult meal costs \$5 and kids eat free. If a group of 15 people came in and 8 were kids, how much would it cost for the group to eat?

(I) *Generating Algebraic template*: We begin by transforming the question into its Algebraic form by replacing the numeric entries with variables using a key-value mapping. In this particular instance, the modified question 'Qt' becomes:

> Qt: at a restaurant, each adult meal costs A and kids eat free. if a group of B people came in and C were kids, how much would it cost for the group to eat?
>
> Mapping: {A:5, B:15, C:8}

(II) *Math-prompts*: We build up on the intuition provided by the multi-verification and cross-checking thought processes mentioned above. We generate analytical solutions of Qt using two different approaches, Algebraic way and Pythonic way. We give the following prompts to the LLM to generate additional context for Qt

> Algebraic prompt: Write a mathematical equation and generate the answer format starting with 'Answer ='
>
> Python prompt: Write a Python function that returns the answer.

The LLM model in response to the above prompts generated the following output expressions

```
# Algebraic expression output
Answer = A*(B-C)

# Python expression output
def total_price(A, B, C):
    return A * (B-C)
```

The above generated analytical solutions gives the user a hint into the 'intermediate thought process' of the LLM. Incorporating additional prompts will improve the accuracy and consistency of the results. This will, in turn, enhance the MathPrompter's ability to generate more precise and effective solutions.

(III) *Compute verification*: We evaluate the expressions generated in the previous step using multiple randomized key-value mappings of the input variables in Qt. To evaluate the expressions, we used the Python's `eval()` method. We compare the outputs to see if we can find a consensus among the answers. This also provides us with a higher level of **confidence** that the answers are correct and reliable. Once the expressions agree on their outputs, we use the values of the variables in the input Q to compute the final answer, as below

```
Algebraic-answer = 35
Pythonic-answer = 35
```

(IV) *Statistical significance*: In order to ensure that consensus is reached among various expressions' output, in our experiments, we repeat the steps (II) & (III) for $N \sim 5$ times and report the most frequent value observed for the answer.

3 Experiment

3.1 Dataset

We evaluate MathPrompter on MultiArith dataset (Roy and Roth, 2016), which is a subset of the Math World Problem Repository (Koncel-Kedziorski et al., 2016). This dataset is a collection of mathematical problems that are specifically designed to test the ability of machine learning models to perform complex arithmetic operations and reasoning. These problems demand the application of multiple arithmetic operations and logical reasoning to be sucessfully solved.

3.2 Baseline

One of the popular baselines is the standard Zero-shot model by (Brown et al., 2020). Their train their models in a way that it is able to recognize and classify new objects or classes that it has never seen before during training. This was achieved by utilizing the semantic relationships between classes.

We also compared against the state-of-the-art Zero-shot-CoT prompting model by (Kojima et al., 2022). This is a very recent approach that addresses the limitations of the standard Zero-shot learning by incorporating a 'context of the task' using CoT to improve the performance. Briefly, their method follows this procedure. Given a question Q, the authors use the prompt 'Lets think step-by-step' followed by Q to generate a response Z. Then, they use the prompt 'The answer (Arabic numericals) is' followed by Z to get their final result.

Model	**Accuracy**
Zero-shot	17.7
Zero-shot (PaLM 540B)	25.5
Zero-shot-CoT	78.7
Zero-shot-CoT (PaLM 540B)	66.1
Zero-shot-CoT + self consistency (PaLM 540B)	89.0
Zero-shot-CoT (**MathPrompter**)	**92.5**
Few-Shot (2 samples)	33.7
Few-Shot (8 samples)	33.8
Few-Shot-CoT (2 samples)	84.8
Few-Shot-CoT (4 samples)	90.5
Few-Shot-CoT (8 samples)	93.0
Zero-Plus-Few-Shot-CoT (8 samples)	92.8

Table 1: **Accuracy on MultiArith dataset**. MathPrompter outperforms all the Zero-shot & Zero-shot-CoT baselines. We emphasize that our model's performance is comparable to 540B parameter models as well as the SOTA Few-shot-CoT approaches. (If not mentioned explicitly, the models in each row consists of 175B parameters. Results are borrowed from (Kojima et al., 2022). They used Textdavinci-002 (175B) model along with the same 8 examples as described in (Wei et al., 2022) for Few-shot and Few-shot-CoT settings.)

3.3 Results

3.3.1 Accuracy comparisons

Table 1 compares the performance of the MathPrompter against the baseline models. The results of few-shot & zero-shot learning based approaches are shown. Furthermore, we add the results for models with different number of parameters to get better highlight the significance of our approach. Since, MathPrompter is a Zero-shot-CoT (175B parameters) method, we choose the state-of-the-art Zero-shot-CoT (175B parameters) model by (Kojima et al., 2022) and a Zero-shot(175B parameters) by (Brown et al., 2020) for fair comparison. We report an accuracy of 92.5% which is a huge improvement to the other SOTA models with 78.7% and 17.7% accuracy, respectively.

3.3.2 Example comparisons

Table 2 presents a sample set of questions and their respective outputs, intermediate steps, and final answers generated by both MathPrompterand the current state-of-the-art model (Kojima et al., 2022). For simplicity, only one output of MathPrompter for each question is shown for both the Algebraic and Pythonic outputs.

The table highlights areas where (Kojima et al., 2022) technique falls short, and where these can be remedied with MathPrompter , which was designed to address these issues. For example, the generated answers sometimes have one step of error, which can be avoided by running the model multiple times and reporting the consensus results. Additionally, the reasoning steps in (Kojima et al., 2022) can be excessively lengthy, but the Pythonic or Algebraic methods can address this by typically requiring fewer tokens. Furthermore, the reasoning steps may be correct, but the final computation is incorrect. MathPrompter address problem by using the Python's `eval()` method function.

In many cases, the MathPrompter generates correct intermediate and final answers. However, there are a few cases, such as the last question in Table 2, where both the Algebraic and Pythonic outputs are in agreement, yet erroneous. We plan to address these issues by incorporating additional methods to further enhance the performance of MathPrompter .

4 Conclusions & Discussions

We introduced MathPrompter, a novel approach that improves LLM performance on mathematical reasoning problems. It also addresses an important concern of building the user trust to some extent in the LLM predictions. We translated our intuition on how students solve arithmetic problems to a LLM model by utilizing the Zero-shot chain-of-thought prompting technique. MathPrompter incorporates ideas like cross-checking the intermediate steps and solving the same math problem using multiple approaches in its design. We empirically show that our model is comparable to SOTA Few-shot-CoT models as well as the larger Zero-shot-CoT models that have 540B parameters. In future, we plan to further evaluate performance on additional datasets and explore incorporating additional prompts into

Table 2:

Example-based comparison between current state of art Zero-shot-CoT and MathPrompter .

Question	**SOTA Zero-shot-CoT**	**MathPrompter**

Q: At the fair Adam bought 13 tickets. After riding the ferris wheel he had 4 tickets left. If each ticket cost 9 dollars, how much money did Adam spend riding the ferris wheel?	A (ONE UNNECESSARY STEP ERROR): Let's think step by step. Adam bought 13 tickets. That means he spent 13 * 9 = 117 dollars on tickets. After riding the ferris wheel, he had 4 tickets left. That means he used 9 tickets to ride the ferris wheel. That means he spent 9 * 9 = 81 dollars on riding the ferris wheel. So the answer is 117 - 81 = 36 dollars. Therefore, the answer (arabic numerals) is 36. ✗	Mapping = {A:13, B:4, C:9}
		`# Algabraic answer`
		`Answer = A*(B-C)`
		`#Python code`
		`def total_price(A, B, C):`
		`    return A * (B-C)`
		Answer = 81 ✓

Q: Each chocolate bar in a box cost \$3. If a box had 9 bars total and Wendy sold all but 3 bars, how much money would she have made?	A (OTHERS; MAXIMUM LIMIT OF TEXT LENGTH): Let's think step by step. Each chocolate bar in a box costs \$3. If a box had 9 bars total, then Wendy sold all but 3 bars. How much money would she have made? We can solve this problem using algebra. Let's start by creating a variable to represent the number of chocolate bars Wendy sold. We'll call this variable "x". If Wendy sold "x" chocolate bars, then she would have 9 - x chocolate bars left in the box. We know that each chocolate bar costs \$3, so the total cost of the chocolate bars Wendy sold Therefore, the answer (arabic numerals) is: Wendy would have made \$12. ✗	Mapping = {A:3, B:9, C:3}
		`# Algabraic answer`
		`Answer = A x (B - C)`
		`#Python code`
		`def money_made(A, B, C):`
		`  return (B-C)*A`
		Answer = 18 ✓

Q: Jerry was helping the cafeteria workers pick up lunch trays, but he could only carry 8 trays at a time. If he had to pick up 9 trays from one table and 7 trays from another, how many trips will he make?	A (COMMONSENSE MISTAKE): Let's think step by step. Jerry can carry 8 trays at a time. There are 9 trays on the first table. That means Jerry will need to make 2 trips to the first table. There are 7 trays on the second table. That means Jerry will need to make 1 trip to the second table. In total, Jerry will make 3 trips. Therefore, the answer (arabic numerals) is 3. ✗	Mapping = {A:8, B:9, C:7}
		`# Algabraic answer`
		`Answer = (B+C)/A`
		`#Python code`
		`def numTrips(A, B, C):`
		`  trips = (B + C) // A`
		`  if (B + C) % A != 0:`
		`    trips += 1`
		`  return trips`
		Answer = 2 ✓

Q: Kaleb bought 14 boxes of chocolate candy and gave 5 to his little brother. If each box has 6 pieces inside it, how many pieces did Kaleb still have?	A: Let's think step by step. Kaleb bought 14 boxes of chocolate candy. This means that he had 84 pieces of chocolate candy. He gave 5 boxes to his little brother, which means that he gave his little brother 30 pieces of chocolate candy. Kaleb still had 54 pieces of chocolate candy. Therefore, the answer (arabic numerals) is 54 ✓.	Mapping = {A:14, B:5, C:6}
		`# Algabraic answer`
		`Answer = A*C - B`
		`#Python code`
		`def candy_pieces(A, B, C):`
		` return A*C - B`
		Answer = 79 (COMMONSENSE MISTAKE)✗

MathPrompter.

5 Limitation

One of the limitations of our work is that while we are running the MathPrompter multiple times in different ways to increase the accuracy of our results, this does not always guarantee the correctness of the output. Both Algebraic and Pythonic expressions have the potential to produce the incorrect results, even if the prompt outputs match each other. This is the fail case as shown in the last row of Table 2. Increasing the number of prompts will mitigate this issue. We are currently investigating techniques that can address this issue in a more principled manner.

References

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

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*.

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*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2020. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

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

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pages 1152–1157.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Timo Schick and Hinrich Schütze. 2020. It's not just size that matters: Small language models are also few-shot learners. *arXiv preprint arXiv:2009.07118*.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

Keith E Stanovich and Richard F West. 2000. 24. individual differences in reasoning: Implications for the rationality debate? *Behavioural and Brain Science*, 23(5):665–726.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

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*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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

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*.