Reasoning in Large Language Models Through Symbolic Math Word Problems

Vedant Gaur*
Aragon High School
vedantgaur101@gmail.com

Nikunj Saunshi†
Google Research, New York
nsaunshi@google.com

Abstract

Large language models (LLMs) have revolutionized NLP by solving downstream tasks with little to no labeled data. Despite their versatile abilities, the larger question of their ability to reason remains ill-understood. This paper addresses reasoning in math word problems (MWPs) by studying symbolic versions of the numeric problems, since a symbolic expression is a "concise explanation" of the numeric answer. We create and use a symbolic version of the SVAMP dataset and find that GPT-3's davinci-002 model also has good zero-shot accuracy on symbolic MWPs. To evaluate the faithfulness of the model's reasoning, we go beyond accuracy and additionally evaluate the *alignment* between the final answer and the outputted reasoning, which correspond to numeric and symbolic answers respectively for MWPs. We explore a *self-prompting* approach to encourage the symbolic reasoning to align with the numeric answer, thus equipping the LLM with the ability to provide a *concise and verifiable* reasoning and making it more interpretable. Surprisingly, self-prompting also improves the symbolic accuracy to be higher than both the numeric and symbolic accuracies, thus providing an ensembling effect. The SVAMP-Sym dataset will be released for future research on symbolic math problems.

1 Introduction

Large language models (LLMs), with hundreds of billions of parameters, can solve a wide range of NLP tasks such as machine translation, question-answering, etc., taking us closer to general-purpose intelligent agents. The initial success of GPT-3 (Brown et al., 2020) has led to many other LLMs (Rae et al., 2021; Smith et al., 2022; Chowdhery et al., 2022) which have, perhaps surprisingly, taken big strides in solving difficult tasks like common sense reasoning, math and science problems (Lewkowycz et al., 2022), and writing code (Li et al., 2022).

Despite the incredible successes, we have little understanding of why LLMs are effective at problems that require reasoning. In fact we have limited techniques to quantifiably study the question of reasoning beyond just evaluating accuracy. Recent ideas like Chain-of-Thought prompting (CoT) (Wei et al., 2022b; Kojima et al., 2022) encourage the model to "think step by step" and output a verbose reasoning in text. However, verifying such reasoning at scale will incur the infeasible cost of manually going over the text outputs. Furthermore, we would like the model's reasoning to be consistent with its outputted answer, in order to trust the presented reasoning. For these considerations, we would like our models to output a *concise reasoning* or explanation for its answer that can be *automatically verified*. In particular, we desire reasoning in the form of explanations that are

- Verifiable: For ease of evaluating correctness of the outputted reasoning, and
- Concise: For scalability of verification. Manually going through text reasoning can quickly get cumbersome

For instance, instead of a text description of an algorithm to solve a problem, a Python implementation of the algorithm would be a more concise explanation for the reasoning behind the algorithm[1]. Similarly, a simple linear model or decision tree explaining the answers of a black-box neural network also achieves the same goal (Ribeiro et al., 2016). Concise explanations can provide clearer insights into the reasoning abilities of models, and verifiable explanations aid interpretability and help foster trust in models, in

*Some clarification on affiliation.

† Most of the work was performed while at Princeton University and after graduating, but before joining Google.

[1]We can automatically verify the answer not just for one problem, but for all instance of that problem

Findings of the Association for Computational Linguistics: ACL 2023, pages 5889–5903

line with explainable AI (Samek et al., 2019).

In this work we use concise and verifiable explanations to study reasoning abilities of LLMs in math word problems (MWPs). LLMs have shown to achieve good zero-shot accuracy on many numeric MWP benchmarks (Kojima et al., 2022). Chain-of-thought like ideas encourage LLMs to first general a step-by-step explanation (in text) before generating the answer. However, this does not satisfy the criteria of being concise or easily verifiable[2]. We address reasoning by considering symbolic versions of numeric MWPs, because a symbolic expression can be viewed as a concise explanation for a numeric answer and can also be automatically evaluated. Thus in this reasoning framework for MWPs, we require an LLM to output both, a numeric answer and a concise symbolic expression, such that we have: (1) high accuracy for the predicted numeric answer, (2) high alignment of the symbolic expression with the predicted numeric answer. While most prior studies focus on goal (1), we argue that goal (2) is equally important for interpretability of these models and to trust the its reasoning. Our main finding is that LLMs can also do reasonably well on goal (2), either by generating a numeric answer and symbolic explanation together, or by generating the answer first and then a post-hoc symbolic explanation. In this context, we make the following contributions:

Symbolic evaluation. We construct a symbolic version of the SVAMP dataset (Patel et al., 2021) called SVAMP-Sym to evaluate LLMs. Firstly we find, perhaps surprisingly, that GPT-3's davinci-002 model already achieves good zero-shot accuracy on symbolic problems (64.2%), comparable to the numeric accuracy of 68.9%. Secondly, this observation provides a simple way to get good accuracy and alignment for numeric problems by first solving symbolic versions and then substituting back the values for variables. This approach generates the numeric answer and a symbolic explanation in one go, thus trivially achieving[3] an accuracy of 64.2% and alignment of 100%.

Self-prompting. There are two key drawbacks with the above approach: (a) symbolic accuracy of 64.2% is lower than the numeric accuracy (68.9%), (b) alignment of symbolic expressions, as post-hoc explanation to the original numeric answers, is very low ($\sim$ 50%). To get a better post-hoc explanation, we propose a novel *self-prompting* approach that first prompts the LLM with the numeric problem and its response to the problem, and then asks it to solve the symbolic problem; see Figure 1. Self-prompting significantly improves alignment with numeric answers to 74% (a 24% absolute improvement). Surprisingly, self-prompting also improves the symbolic accuracy to 71.7%, higher than both the raw numeric and symbolic accuracies of 68.9% and 64.2% respectively. This suggests that self-prompting has an ensembling effect.

We perform further ablation studies and analyses and hope that these insights will aid future work on using LLMs for reasoning problems.

1.1 Related Work

Language models like GPT-3 (Brown et al., 2020) and MLMs like BERT (Devlin et al., 2019) have demonstrated impressive emergent behaviors (Wei et al., 2022a) at scale. For math problems, Minerva (Lewkowycz et al., 2022) was fine-tuned from PaLM (Chowdhery et al., 2022) to do well on many MWP benchmarks. Instead of fine-tuning, Wei et al. (2022b) uses in-context learning and finds that asking the model to "think step by step" (CoT prompting) improves few-shot accuracy on MWPs; Kojima et al. (2022) verify this for zero-shot setting as well, which is the focus of our work.

There is limited theoretical work for the downstream success of LMs (Saunshi et al., 2021; Xie et al., 2022) and the emergent behaviors of LLMs through scaling laws (Kaplan et al., 2020). Our idea of self-prompting is motivated by the efficacy of in-context learning (Brown et al., 2020) and prompting (Liu et al., 2023) in LMs. The ensembling effect of self-prompting idea could be related to self-calibration abilities of LMs (Kadavath et al., 2022). Finally, Ho et al. (2022) survey the progress of LMs on various notions of reasoning; we consider a weaker notion of "concise post-hoc explanations" here.

2 Math Word Problems with LLMs

2.1 SVAMP-Sym Dataset

We choose the SVAMP dataset (Patel et al., 2021) for testing LMs on MWPs because it provides numeric answers in the form of numeric expressions (rather than just numeric values). This

[2]It is not uncommon for the outputted reasoning to be inconsistent with the final answer

[3]If a "calculator" can evaluate symbolic expressions.

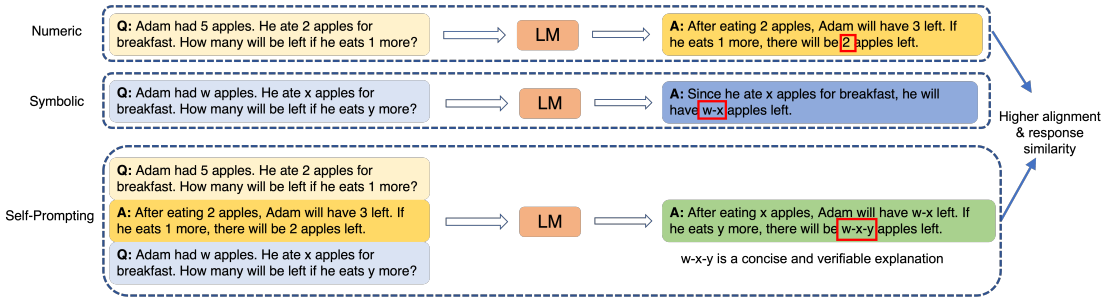


Figure 1: LMs can be queried to solve numeric/symbolic math problems. Self-prompting includes the numeric problem and the LM's solution to it before passing the symbolic problem. This encourages the model to output the answer that aligns with the numeric answer. The symbolic expression `w-x-y` serves as a concise explanation/reasoning for the numeric answer of 2.

lets us automatically convert the dataset into a symbolized version, without manual annotation. The main idea is to replace all occurrences of numbers in the problem statement with newly introduced variables, e.g. `(w,x,y,z)`. Appendix A provides more details on the dataset construction. The dataset is released in https://github.com/vedantgaur/Symbolic-MWP-Reasoning.

2.2 Querying and Evaluating LMs

Broadly, our evaluation pipeline has four phases: (1) get a verbose response from the LLM for the math problem, (2) prompt the LLM to extract just the answer (number or symbolic expression) from its initial response, (3) refine the extracted answer using a novel *filtering* step, (4) compare the filtered answer to the ground-truth answer.

Initial response. We query the LM with the problem statement and an optional CoT prompt, i.e. `"Q: <Problem> A:"` or `"Q: <Problem> A: Let's think step by step."`. `<Problem>` could either be a numeric or symbolic problem. Table 3 summarizes the prompts used for various settings.

Answer extraction. Since the LLM outputs a long text response (Figure 1), we use an extraction prompt to isolate the answer, similar to Kojima et al. (2022). We query the LM with the transcript so far, followed by the question and the prompt `"The final answer (only the number) is:"` to isolate the numeric answer. Table 3 has the similar prompt for symbolic problems.

Answer filtering. The extraction prompt does not always isolate the final answer and sometimes outputs a sentence, especially for symbolic problems. Thus we add a LM-independent filtering step which includes stripping escape sequences, removing commas, de-latexifying equations, picking the longest symbolic expression, among others; more details in Appendix C.2.

Answer evaluation. We compare the filtered answer to the ground-truth answer (symbolized expression or numeric value). Since there are multiple ways to express the same symbolic expression (e.g. `"w + (y + x)"` and `"w + x + y"`), we compare two expressions through their evaluations on 20 random variable assignments. If they match on all 20 assignments, we adjudge them to be equivalent, making a (reasonable) assumption that 20 random assignments will avoid false positives.

3 Experimental Results

We pick 150/1000 examples from the SVAMP dataset (due to budget constraints) and run each examples 5 times. We use GPT-3's davinci-002 model with temperature 0.0 for (mostly) deterministic outputs, with a max token length of 256.

3.1 Numeric and Symbolic Evaluations

We discuss the accuracies for solving numeric and symbolic math problems from SVAMP and SVAMP-Sym respectively.

Numeric accuracy. The zero-shot numeric accuracy both with chain-of-thought (CoT) prompt and without (vanilla) are presented in Table 1; they are 68.9% and 65.6% respectively. This good performance is unsurprising given prior work (Kojima et al., 2022). Our accuracies are $\sim$ 5-7% higher than Kojima et al. (2022), due in part to better answer extraction and filtering.

Symbolic accuracy. We also evaluate raw symbolic problems from SVAMP-Sym in the vanilla and CoT settings with 3 natural choices for variables: `(w,x,y,z)`, `(i,j,k,l)` and `(p,q,r,s)`. Firstly we observe, in Table 1, that GPT-3 can

		Numeric	Symbolic				
			(w,x,y,z)			(p,q,r,s)	(i,j,k,l)
	Evaluation	Raw (-F)	Raw (-F)	**SP** (-F)	**SP + AP**	Raw	Raw
Accuracy	*Vanilla*	65.6 (61.6)	59.7 (47.6)	61.9 (40)	**68.3**	62.3	53.5
	CoT	68.9 (65.9)	64.2 (48.8)	67.9 (48.6)	**71.7**	64.4	58.4
Alignment	*Vanilla*	-	52.9 (40.7)	60.3 (40)	**64.9**	56.3	44.7
	CoT	-	51.2 (39.1)	63.1 (44.9)	**74**	51.9	47.1
Similarity (BLEU)	*Vanilla*	-	27.8	44.2	**49.8**	27.1	26.8
	CoT	-	21.3	53.9	**57.6**	22.7	21.4
Similarity (Levenshtein)	*Vanilla*	-	56.5	65.2	**71.3**	56.8	55.4
	CoT	-	44.9	75.6	**79.8**	45.4	43.9

Table 1: Zero-shot accuracy and alignment evaluations using GPT-3. All values are reported in %. "Raw" refers to evaluation on the SVAMP and (SVAMP-Sym) dataset for numeric (symbolic) MWPs; (-F) refers to the output before the filtering step. "SP" is the new self-prompting method and "SP + AP" refers to two-stage self-prompting where we an additional "Alignment Prompt" is added when needed; see Section 3.3. CoT prompting consistently elicits higher accuracy from the model for numeric and symbolic problems. While accuracy and alignment only look at the final answers, we also measure similarity between the full responses for numeric and symbolic problems. As evident, self-prompting significantly improves the similarity under BLEU score and Levenshtein metric; Appendix B.1 has more details on these metrics.

achieve pretty high symbolic accuracies with variables `(w,x,y,z)`: vanilla and CoT settings achieve 59.7% and 64.2% respectively, which is just 4-5% lower than numeric accuracy. Furthermore, we notice that variables `(i,j,k,l)` have slightly worse accuracy than other variable settings, possibly because `(w,x,y,z)` and `(p,q,r,s)` are more popular choice for variables in the training data for language models.

Effect of filtering. We report the accuracies without the filtering step in Table 1; these are the (-F) entries. While there is a 4-5% drop in the numeric accuracy without filtering, the drop is 12-14% for symbolic problems, suggesting that filtering is much more crucial for symbolic problems[4]. Our extraction and filtering steps still have issues and there is scope for improvement.

3.2 Reasoning and Alignment

While prior work only cares about the accuracy on MWPs, we also study of reasoning abilities of LLMs by requiring them to generate a concise explanation for numeric answers in the form of a symbolic expressions. We evaluate "reasoning ability" through an alignment metric that checks if the outputted numeric answer and symbolic expression compute to the same value. In general there is no consistent zero-shot method to return a perfectly aligned symbolic expression. A natural attempt to generate such an expression is to directly solve the symbolic versions of numeric problem. However this approach has very low alignment, i.e. the symbolic output does not reflect the way in which the model solved the numeric problem. Specifically in Table 1, the average alignment score for raw symbolic outputs is only 52.9% and 51.2% for Vanilla and CoT respectively. This motivates self-prompting.

3.3 Self-prompting

In order to improve alignment, we propose a two-step procedure that first inputs the numeric MWP and the LM's response to it, followed by the symbolic version of the MWP. In particular the prompt looks like `"Q: <Numeric Question> A: <Model Response> Q: <Symbolic Question> A:"`. Given the in-context tendencies of LMs, we hope that this encourages the symbolic response to imitate the numeric response and thus return a well aligned expression. We find in Table 1 that this approach (termed **SP**) indeed improves the alignment by $\sim$ 10% over the naive approach.

We take this one step further: whenever the numeric and symbolic answers do not align, we add another "alignment prompt" before the symbolic problem that explicitly asks the model to copy the numeric answer; see Table 3 for the exact format. Results in the **SP+AP** column of Table 1 verify that this leads to another 11% improvement over **SP** and $\sim$ 22% improvement

[4]Intuitively it makes sense that extracting an expression/equation is harder than extracting a single number

over raw symbolic. Surprisingly we find that **SP+AP** has higher accuracy than raw numeric and raw symbolic, suggesting a "best of both worlds" or ensembling phenomenon in action. Further analysis in Figure 7 reveals how self-prompting combines the benefits of numeric and symbolic.

We also compute the similarity between the full numeric and symbolic responses. Table 1 reveals that the average similarity is significantly higher for **SP** and **SP+AP** compared to raw symbolic. So not only do the answers align more but also the full text responses are very similar. Histograms of similarity scores can be found in Appendix B.1. Additional analyses and results can be found in Appendix B.

4 Conclusions and Future Work

This paper studies reasoning in LLMs for MWPs and results suggest that LMs are good at zero-shot solving of symbolic MWPs, and that this ability can lead to concise explanations. Self-prompting emerges as a promising idea to generate better explanations and the ensembling effect demonstrated by it can potentially have other applications (left for future work). Alignment with self-prompting, while significantly better than with raw symbolic outputs, still has a lot of scope for improvement. Aspects that are not considered are few-shot learning of explanations and the role of temperature, which could improve accuracy and alignment. Finally the notion of "concise explanation" to study reasoning can have implications beyond MWPs.

Broader Impact Statement. Given the incredible successes of LLMs, it is becoming increasingly important to study why they work and how to debug them when they are wrong. There are ongoing debates and discussions about whether LMs are simply "stochastic parrots" (Bender et al., 2021) or they actually "understand" language. Besides there are also privacy concerns (Carlini et al., 2021) associated with LLMs trained on extremely large corpora. Our work attempts to formalize a weak notion of "reasoning" in math problems that could help with improving the intepretability, and thus trustworthiness, of such models. This is extremely important if LLMs are to be deployed in real-life applications. That said, any preliminary notion or definition of "reasoning in LLMs", including the one in this paper, should be taken with a healthy dose of skepticism.

References

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*.

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*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. 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)*. Association for Computational Linguistics.

Vedant Gaur and Nikunj Saunshi. 2022. Symbolic math reasoning with language models. In *2022 IEEE MIT Undergraduate Research Technology Conference (URTC)*.

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2022. Large language models are reasoning teachers. *arXiv preprint arXiv:2212.10071*.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield Dodds, Nova DasSarma, Eli Tran-Johnson, et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

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

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*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *arXiv preprint arXiv:2203.07814*.

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*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

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*.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller. 2019. *Explainable AI: interpreting, explaining and visualizing deep learning*, volume 11700. Springer Nature.

Nikunj Saunshi, Sadhika Malladi, and Sanjeev Arora. 2021. A mathematical exploration of why language models help solve downstream tasks. In *International Conference on Learning Representations*.

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

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

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

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2022. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*.

A Symbolized Dataset

We employ a multi-step process to convert the original SVAMP (Patel et al., 2021) prompts into its symbolic version SVAMP-Sym, motivated by the symbolic construction in Gaur and Saunshi (2022). The SVAMP dataset is under the MIT License. Our SVAMP-Sym dataset has exactly the same set of 1000 problems as SVAMP. Given a common math word problem (MWP) from SVAMP, we parse through the given text for all numbers, which are stored in a list. Using regex, the index of the numbers can be found and replaced with keys for future replacement with variables. We use <i> (where $i \in [1, 4]$ as there are at most four numbers in each problem definition) as keys. As shown in Figure 4, by generalizing the converted prompt, we allow for easy manipulation of prompts to whatever variable a user wants to use and test for downstream tasks. We then convert the keys to their respective variables. For our tests we primarily use the variables `(w,x,y,z)` for a few main reasons:

1. This set of variables is the most common in general mathematical word problems and thus makes the most sense to use as variables as opposed to an arbitrary sequence of random, or even consecutive letters.

2. We find that the use of variables such as x_1, x_2, ..., x_n (x1, x2, ..., xn when inputted into the model) many times confuses the model into conflating the simulated subscript as a coefficient.

3. We are able to see that the model achieves similar, if not greater accuracies with the use of `(w,x,y,z)` as opposed to other sequences of variables, see Table 1.

Moreover, the use of a predetermined length of variables is also possible due to the aforementioned maximum number of four numbers for each prompt in the SVAMP dataset.

See Figure 4 for an example problem, its answer, and our symbolized version of it.

B Ablations and Analysis

B.1 Response Similarity

To find the syntactical similarity between the numeric and symbolic responses, we employ two main metrics: BLEU Scores and Levenshtein Distances. BLEU score is a standard metric used to judge similarity between sentences based on the n-grams they share. Levenshtein distance (also known as edit distance) is a standard metric to distance between two strings: the minimum of swap/deletion/insertion operations that are needed to convert one string to another. To measure similarity between s_1 and s_2, we use $(\texttt{maxlen}(s_1, s_2) - \texttt{Levenshtein}(s_1, s_2))/\texttt{maxlen}(s_1, s_2)$/ Using the nltk.translate.bleu_score module, we define the average of BLEU-1, BLEU-2 and BLEU-3 metrics by passing `weights=[1/3, 1/3, 1/3]` in the sentence_bleu function. For computing Levenshtein Distances, we utilize the python-Levenshtein package's distance function. As described in the histograms presented in Figure 5 and Figure 6, we find much higher similarity scores when employing self-prompting. This logically follows the higher alignment values of such runs. More specifically, however, the similarity of the two scores is ultimately more contingent on the verbiage of the output. As indicated in Figure 1, the SP often closely tracks the exact output of the numeric response and simply replaces the numbers with the respective variables/symbolic expressions, and outputs an expression instead of a final number. While metrically evident in the provided plots, we see that this "mirroring" phenomenon occurs frequently with the use of SP, evident through the high density of similarity scores close to 1 in Figure 5.

B.2 More on Alignment

While we find that the use of the alignment prompt is effective in raising both the accuracy and alignment of a symbolic problem, we run a few supplementary experiments to investigate this behavior even further. When giving the model the alignment prompt (see Table 3) from the beginning, not simply when the numeric and symbolic outputs do not align, we actually find a decrease in accuracy from the self-prompting + alignment prompt run. CoT accuracy is 62% and vanilla accuracy is 60.9%. Similarly, alignment accuracies are 61.5% and 60.4% for CoT and vanilla, respectively. When evaluating alignment for the base self-prompting run, we find that the model aligns 83.9% when the numeric output is correct, and 29.7% when it is wrong. Such numbers possibly suggest the model's cognizance of whether or not the numeric evaluation was performed correctly; an implicit understanding of mathematical problem solving.

B.3 Difficulty of Problems

We highlight a metric for the difficulty of a problem with respect to the primary operation performed in the answer to the prompt. The SVAMP dataset stores a "Type" key that denotes the primary elementary mathematical operation performed to get to the answer (primary in cases where there is more than one operation). We see that when graphing the accuracies of various evaluation methods while isolating the operation of the problem that the numeric and symbolic runs exhibit a somewhat complementary behavior. While numeric does on average better on problems with division, symbolic runs have higher accuracy on multiplication, see Figure 7. Table 2 has breakdowns of the exact accuracies per each tag. Interestingly, the self-prompting approach seems to do well on both multiplication and division, and its performance is close to the max of the numeric and symbolic performance for each category, thus hinting to a "best of both worlds" phenomenon.

C Additional Details

C.1 Prompt formats

In the SVAMP dataset, each problem contains a problem statement and a question. For both raw numeric and symbolic evaluations, we input the problem into the model with the CoT prompt if appropriate. For self-prompting, however, in order to increase alignment between the numeric and symbolic outputs, we add the entire transcript of the numeric evaluation (problem, answer prompting, symbolic problem). A detailed transcript of each of the different prompts and use cases can be found in Table 3.

C.2 Filtering

Since there is high variability in the LM's outputs, due to the necessity to reason when solving a MWP, we employ several filtering techniques in a `filter()` function that cleans up the extracted numeric or symbolic output. A few main steps in the filtering pipeline are as follows:

- Character replacing

		Accuracy (%)			
	Evaluation	Addition	Subtraction	Multiplication	Division
Numeric	*CoT*	64.7	64.3	68	88.1
	Vanilla	54.1	62.8	68	87.4
Symbolic {w, x, y, z}	*CoT*	64.1	58.8	90	70.4
	*Vanill*a	41.2	63	90	62.2
Self-prompting {w, x, y, z}	*CoT*	67.6	66.1	94	85.2
	Vanilla	60	61.3	80	73.3

Table 2: While the accuracies presented are fairly consistent within each separate evaluation run, we see that there are clear biases in which the model is able to perform certain types of problems better depending on the context of the run. Significantly, it should be noted that the self-prompting is able to employ both the efficiencies of numeric, *and* symbolic runs with the increased alignment.

Example	<Numeric Setup> = "Adam had 5 apples. He ate 2 of them for breakfast." <Numeric Question> = "How many apples will he have left if he eats 1 more?" <Symbolic Setup> = "Adam had w apples. He ate x of them for breakfast." <Symbolic Question> = "How many apples will he have left if he eats y more?"
Prompts	<CoT Prompt> = "Let's think step by step." <Numeric Extract Prompt> = "The final answer (only the number) is:" <Symbolic Extract Prompt> = "The final answer (only the expression in terms of given variables) is:" <Align Prompt> = "Copy the above numeric response word to word but replace numbers with the right symbolic expression."
Numeric	Q: <Numeric Setup> <Numeric Question> A: <CoT Prompt> <Numeric Response> // language model's verbose response <Numeric Question> <Numeric Extract Prompt> <Numeric Extracted>
Symbolic	Q: <Symbolic Setup> <Symbolic Question> A: <CoT Prompt> <Symbolic Response> // language model's verbose response <Symbolic Question> <Symbolic Extract Prompt> <Symbolic Extracted>
Self-prompt	Q: <Numeric Setup> <Numeric Question> A: <CoT Prompt> <Numeric Response> <Align Prompt> // [optional] only if alignment fails without it Q: <Symbolic Setup> <Symbolic Question> A: <CoT Prompt> <Symbolic Response> <Symbolic Question> <Symbolic Extract Prompt> <Symbolic Extracted>

Table 3: We present the prompting pipeline for various methods. Prompts in blue are the ones we pass to the model, while the text in green are the output of the language model. In each of these methods, we include a final filtering step on top of the extracted answers.

  - Dollar signs
  - Percentages
- Cleaning up the output by removing all words besides the expression and/or final number
- Addressing cases of outputs such as code or LaTeX
- Isolating the outputted/final expression if the answer is given in terms of an equation (say "z = w + x")

The detailed (pseudo) code of the function can be found at the end.

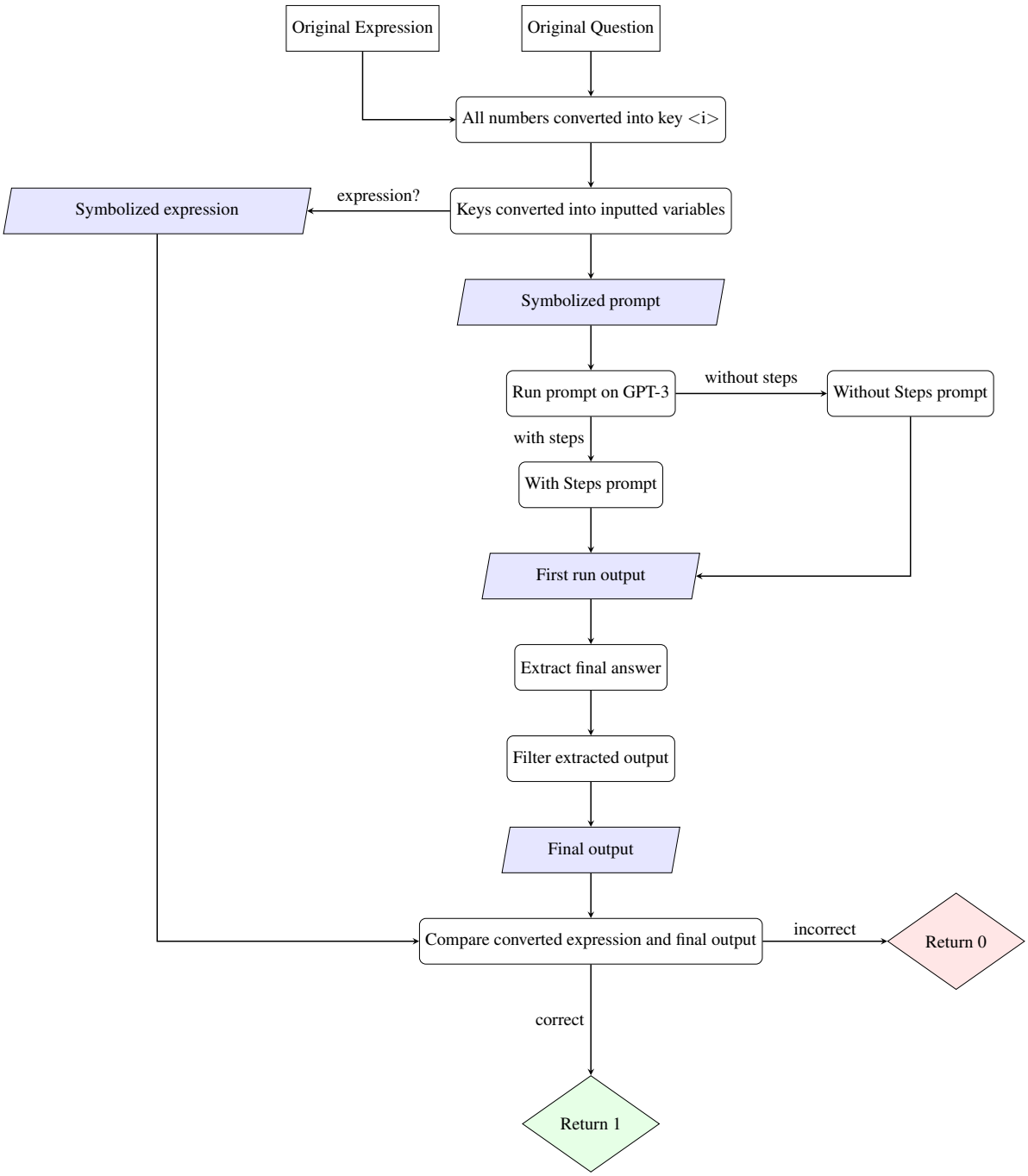


Figure 2: Flowchart of the pipeline from an original expression to correct or incorrect outputs. The purple cells represent the outputs of the GPT-3 model as well as output processing. Both the "Original Expression" and "Original Question" at the top in rectangular cells are numeric, baseline prompts.

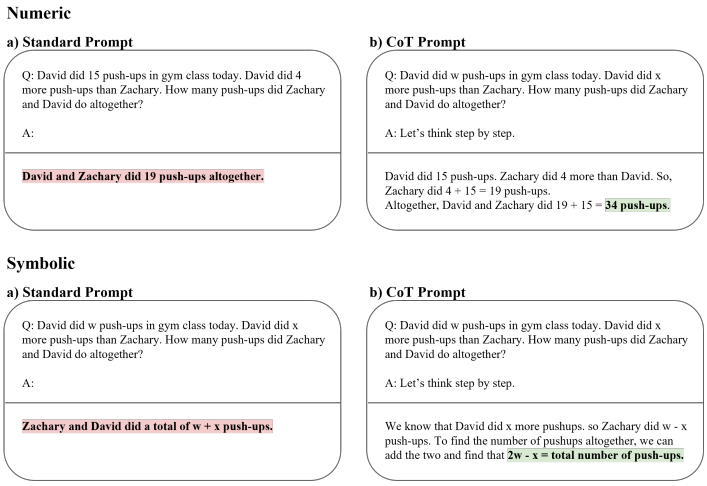


Figure 3: Examples of the input-output GPT-3 sequence of both numeric (above) and symbolic (below) runs. CoT prompting, as reported in previous papers, elicits much more detailed, and oftentimes correct outputs from the model through the additional reasoning step. We find that the use of the prompt is not exclusive to numeric reasoning, and are able to identify similar processes in symbolic runs.

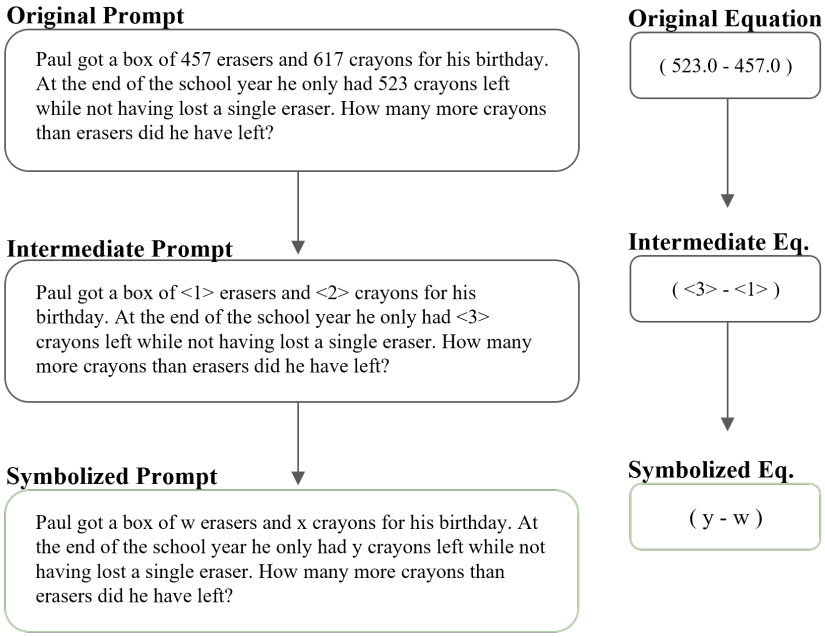
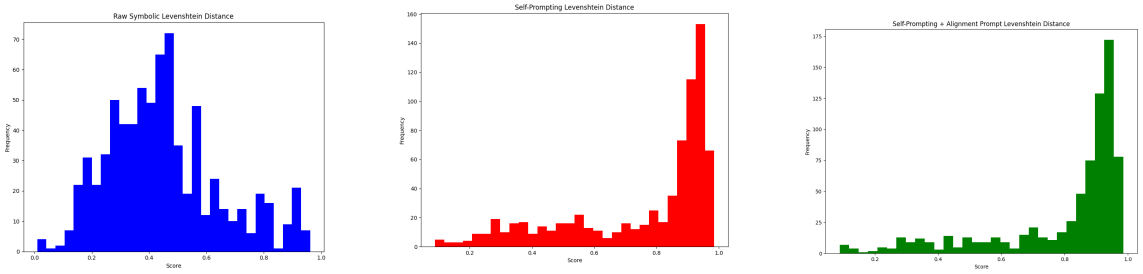


Figure 4: The process of converting a numeric problem into a symbolic one. The answer to the problem is an expression given by the SVAMP dataset, so we can easily convert it to a symbolic equation. Appendix A has more details on how this symbolization was implemented.



(a) Raw symbolic (b) Self-Prompting (c) Self-Prompting + Alignment Prompt

Figure 5: Levenshtein distance calculated on raw symbolic, self-prompting, and self-prompting with an additional alignment prompt outputs. Values near 1.0 (to the right) denote two sentences with very similar syntactic similarity. As evident in the graphs above, the distribution of both (b) and (c) are much more heavily skewed to the right with unimodal peaks near 1.0, whereas the distribution in (a) is shifted much more to the left. This means that both (b) and (c) are much more similar to the outputs they were compared with (numeric) than (a), highlighting the efficacy of self-prompting in mirroring numeric responses.

(a) Raw symbolic (b) Self-Prompting (c) Self-Prompting + Alignment Prompt

Figure 6: BLEU Scores calculated on raw symbolic, self-prompting, and self-prompting with an additional alignment prompt outputs. As with Figure 5, values near 1.0 (to the right) denote two sentences with very similar syntactic similarity. In this instance, the BLEU Score was calculated by tokenizing and comparing the numeric outputs with respective outputs in (a), (b), and (c). This value was then normalized and plotted as described in the figures above. Both (b) and (c) both show more left-skewed distributions, while (a) models a right-skewed one. Similar to Figure 5, the use of BLEU Scores highlights how self-prompting helps with the alignment of numeric and symbolic outputs.

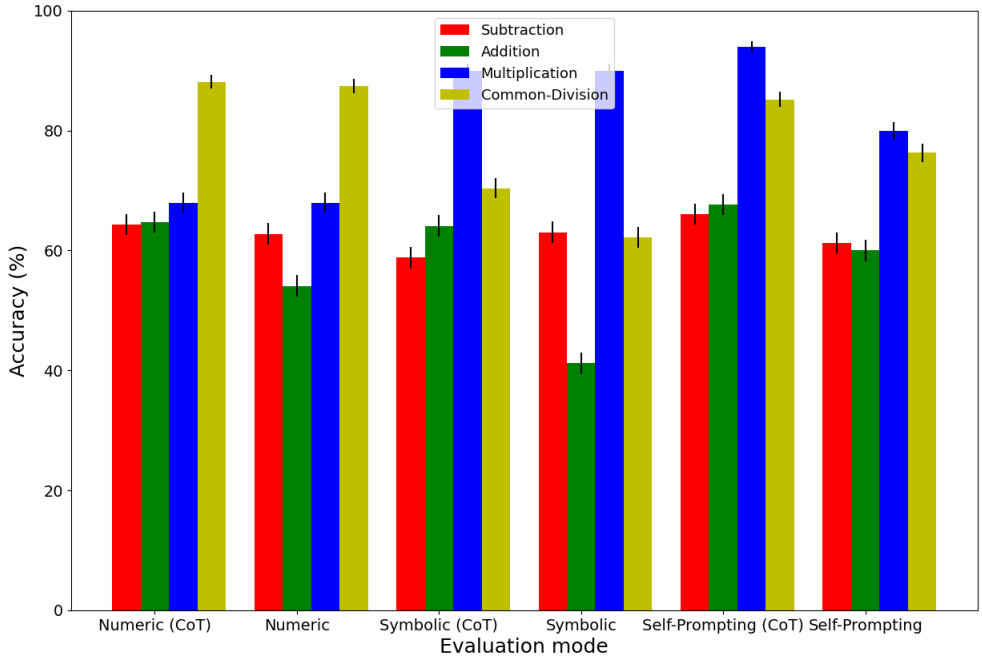


Figure 7: Accuracies of the "tag" of the prompt inputted into the model based on the evaluation method of the model. We observe that numeric consistently performs above average at division, symbolic at multiplication, and self-prompting at both. By combining the strengths of both numeric and symbolic evaluation, we see that self-prompting is able to perform as well, if not better than both numeric and symbolic prompting. Furthermore, as with general accuracy CoT also seems to provide boosts to addition accuracies, emphasized especially when comparing symbolic evaluations (Vanilla and CoT).

```
def filter_symbolic(response):

    response = response.lower()
    response = response.strip('\n')
    print(f"Original Output: {response}")

    # De-latexifying
    response = LatexNodes2Text().latex_to_text(response)
    response = response.replace("$", "")

    # Using * as multiplication operator
    response = response.replace('.', '*')

    # Handling the division symbol
    response = response.replace("%", "")
    response = response.replace('\u00F7', '/')

    # Remove spaces and construct a boolean array denoting whether
    # the character is in the set {'w', 'x', 'y', 'z', '/', '*', '+', '-', '(', ')'}
    math_sym_set = set(['w', 'x', 'y', 'z', '/', '*', '+', '-', '(', ')'] +\
    [str(a) for a in range(10)])

    # Check for "words" that only contain chars from math_sym_set
    response = response.replace("=", " = ")
    words = response.lower().split()
    is_math_sym = np.array([np.all([c in math_sym_set for c in word])*len(word) for
                                            word in words])


    # Pick the substring with non-zero entries that has the largest sum,
    # i.e. the largest substring of the original string that is an equation/
                                            expression
    idx, len_ = longest_sum(is_math_sym)
    response = ''.join(words[idx:idx+len_])
    print(response)

    # Add multiplication operator * if needed.
    # Logic: If neither of two consecutive characters is an operator
    # then likely a multiplication operator needs to be added between them.
    # Some edges cases like '(p' or 'q)' are handled
    op_set = set(['/', '*', '+', '-'])
    digit_set = set([str(a) for a in range(10)])
    new_response = []
    for i in range(len(response)):
        new_response.append(response[i])
        # Check if '*' needs to be added
        if i < len(response)-1 and response[i] not in op_set and response[i+1] not
                                                    in op_set:
            # No need to add '*' if the consecutive chars of the type '(p' or 'q)'
                                                        of '25'
            if (response[i] != '(' and response[i+1] != ')') and (response[i] not in
                                                        digit_set or response[i+1]
                                                        not in digit_set):
                new_response.append('*')

    print(f"Final Output: {new_response}")
    return ''.join(new_response)
    return output


def filter_numeric(response):
    output = str(response).replace(",", "")
    output = output.replace("$", "")
    output = output.strip('\n')
    try:
        output = int(re.findall('\d+', output)[0])
    except:
        output = output
    return output
```

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Section 4

☐ A2. Did you discuss any potential risks of your work?
Not applicable. The paper mostly deals with fundamental understanding of LLMs, which can help mitigate potential risks of LLMs

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Left blank.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 2.1

☑ B1. Did you cite the creators of artifacts you used?
Section 2.1

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Section A

☑ 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)?
Section A

☒ 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?
We directly adapted an existing dataset and replaced numbers with variables

☒ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
The dataset is a derivate of another dataset, and thus imports all of its properties

☑ 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.
Section A

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?
Not applicable. Left blank.

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?
Not applicable. Left blank.

☑ 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?
Left blank.

☑ 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.)?
Section B.1

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.