Reflection of Thought: Inversely Eliciting Numerical Reasoning in Language Models via Solving Linear Systems

Fan Zhou[1*], **Haoyu Dong**[2*†], **Qian Liu**[3], **Zhoujun Cheng**[1], **Shi Han**[2], **Dongmei Zhang**[2]
[1]Shanghai Jiao Tong University, [2]Microsoft Research Asia, [3]Sea AI Lab
{zhoufan98,blankcheng}@sjtu.edu.cn,
{hadong,shihan,dongmeiz}@microsoft.com,
liuqian@sea.com

Abstract

Recent language models have struggled to generalize to a large range of numbers in numerical reasoning. In this paper, we propose a novel method that leverages simple numbers as anchors to elicit the implicitly inferred arithmetic expressions from language models, and then explicitly applies the expressions to original numbers to get the answers. Experimental results on several numerical reasoning benchmarks demonstrate that our approach is highly effective. More importantly, our approach works in the inference phase without extra model training, making it highly portable and achieving significant and consistent performance benefits across a variety of language models in zero-shot, few-shot, and fine-tuning scenarios.

1 Introduction

Language Models (LMs) have demonstrated great success on a wide range of natural language tasks [5, 2, 3], but their performance slumps when it comes to reasoning about numbers. Even rational numbers, a small subset of real numbers, comprise an infinite space that pre-training corpora cannot cover entirely, creating a severe obstacle to LMs. Recent works have shown strong context understanding capabilities of LMs in numerical reasoning datasets [6, 4], but LMs are still far from being reliable on end-to-end numerical calculation [12, 13]. According to our preliminary study, whose details can be found in Appendix A, the performance of LMs drops significantly as the input numbers get more complex. Similar observations are also reported by Razeghi et al. [16].

Although existing LMs easily fail to calculate complex numbers, there is still a silver lining: with the same context, LMs are more accurate and stable on simple numbers than complex numbers, demonstrating that LMs have a strong aptitude for applying arithmetic principles to simple numbers. This motivates us to *leverage simple numbers as "anchors" to probe the implicitly inferred arithmetic expressions from language models, and then explicitly apply the expressions on original complex numbers*. Specifically, as illustrated in Figure 1, when detecting complex numbers (e.g., $10,477$ and $7,459$) that are challenging for LMs, we first replace them by anchor numbers (10 and 7, etc) and use LMs to output answers (3, etc). Then we inversely elicit the secret arithmetic relationship ($x_1 - x_2$) inside LMs through anchor numbers and their corresponding answer, and finally explicitly applying the arithmetic relationship on the initial complex numbers ($10,477 - 7,459$) to produce the answer ($3,018$). In this way, our method combines the advances of LMs on understanding complex context and memorizing simple numbers for reliable numerical reasoning.

In this paper, we present a novel framework to elicit the numerical reasoning knowledge hidden in LMs. With simple numbers as anchors, we tackle the problem of inversely elicit the arithmetic expressions by analytically solvable linear systems. Meanwhile, since the linear systems may contain noise, alternative search-based method is further developed to increase robustness. Experimental results on several representative numerical reasoning datasets demonstrate that our method is highly effective. More

[*]Equal contributions. †Corresponding author.

36th Conference on Neural Information Processing Systems (NeurIPS 2022) Workshop on MATH-AI.

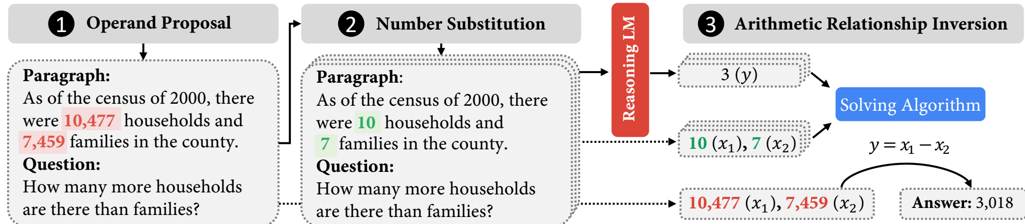

Figure 1: The illustration of our proposed framework, which elicits numerical reasoning in language models via **So**lving **Li**near **S**ystems (SOLIS).

importantly, our framework does not need any additional training or annotation efforts, since it works at test time, making it highly portable to different kinds of LMs and delivering consistent gains over various LMs in zero-shot, few-shot and fine-tuning scenarios.

2 Numerical Reasoning via Solving Linear Systems

As mentioned above, current language models are vulnerable to complex numbers, whereas they work consistently well when the operands are simple, i.e., relatively small integers. Such observations motivate us to *simplify the numbers before feeding them into language models*, thus enabling reliable neural-based numerical reasoning. In this section, we first provide an overview of our framework SOLIS, and then we elaborate on each part of our framework in detail.

Method Overview Our method can be integrated into different language models at test time. For the sake of clarification, in the following we refer to LMs that can steadily perform numerical reasoning as reasoning LMs. As shown in Figure 1, SOLIS involves three stages: (1) *Operand Proposal*: given a paragraph, we first identify the numbers which are necessary for the reasoning LM to perform numerical reasoning (e.g., **10, 477**); (2) *Number Substitution*: these proposed operands are generally complex for language models, and thus they need to be substituted with randomly chosen simple numbers (e.g., **10**) to make the model input simpler. Using the reasoning LM, we can obtain a set of predicted answers with respect to each substituted paragraph after several substitutions. (3) *Arithmetic Relationship Inversion*: using these paragraphs and their answers, we can inversely derive the secret reasoning flow in the reasoning LM, i.e., the arithmetic expression between operands (e.g., $y = x_1 - x_2$). The final answer is obtained by applying the expression on the original numbers.

Operand Proposal There are often many numbers involved in a paragraph, and it is hard to derive how all of these numbers relate to each other arithmetically at the same time. So, it is important to trim the prospective operands to a manageable size in the operand proposal stage. To address the issue, inspired by prior works [18], we provide a novel technique that employs number perturbation and the reasoning LM to measure the relevance. In prior works, relevance is assessed by the degradation of the classifier score after erasing each pixel, where a substantial degradation indicates a strong relevance. Similarly, we consider a number to be essential to the final answer if there is a difference between the model predictions before and after perturbing it. Regarding perturbations, we implement it by adding a small adjustment to each number in the paragraph (e.g., $98.5 \rightarrow 98.6$). More details about the operand proposal mechanism can be found in Appendix B.

Number Substitution After the operand proposal stage, a random set of numbers is generated to substitute the proposed operands sequentially. These numbers are referred to as anchor numbers below. Each anchor number is an integer between 1 and 20, a range that we believe reasoning LMs can easily handle. Meanwhile, to minimize the effects of number substitution, we strive to maintain the order relationships among the numbers. Taking the example from Figure 1, we make the substitution number corresponding to $10,477$ larger than the one corresponding to $7,459$ since $10,477$ is larger than $7,459$. Notably, the random number substitution must be repeated several times (e.g., three times in Figure 1) to obtain a group of anchor numbers. Along with the original question, each of these paragraphs is fed into the reasoning LM to predict the answer, which we call the anchor answer. Typically, the number of anchor answers must exceed the number of operands for the subsequent arithmetic relationship inversion stage to be feasible.

Arithmetic Relationship Inversion Given a collection of anchor numbers and anchor answers, the arithmetic relationship inversion stage investigates the relationship between these numbers and induces an expression to reflect it. Taking the example from Figure 1, a typical expression can be $y = x_1 - x_2$, where x_1 and x_2 are both anchor numbers while y is the anchor answer. Although the example expression appears intuitive, deriving such an expression from data points is tremendously difficult

because the solution space is theoretically infinite. To make it practicable, as a first step, we begin by limiting the problem-solving space to compositions of binary operators, where each operator can be addition, subtraction, multiplication or division, the most prevalent operators in numerical reasoning [6]. Meanwhile, there can be up to three compositions, which means the expression contains a maximum of four operands. With such priors, the insoluble expression induction problem can be turned into a linear system solving problem, where the anchor numbers, the anchor answers, and their compositions constitute a linear system. In this way, the problem of expression induction can be tackled by the *solving algorithms* for linear systems. Finally, the answer can be reached in a trustworthy and interpretable manner by applying the derived expression to the original numbers.

3 Solving Algorithm

Given a paragraph and a question, we denote a group of anchor numbers as $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and the arithmetic relationship as an expression f, which should produce the answer y by $y = f(\boldsymbol{x})$. The objective of solving algorithms is to recover f from different groups of anchor numbers $\mathbf{X}$ and corresponding anchor answers $\boldsymbol{y}$. We propose to transform and formulate the arithmetic relationship inversion as solving a system of linear equations. Given expression $f(\boldsymbol{x})$ with four fundamental arithmetic operations, we transform the equation $y = f(\boldsymbol{x})$ by multiplying denominators on both sides when operator division exists, then we get:

$$a_0 \cdot C + a_1 \cdot x_1 + a_2 \cdot x_2 + a_3 \cdot y + a_4 \cdot x_1 x_2 + \ldots + a_k \cdot (x_1 x_2 \ldots\ x_n y) = 0 \quad (1)$$

For example, $y = 1 - x_1/x_2$ can be transformed to $x_2 - x_1 - x_2 y = 0$. Then uncovering $f(\boldsymbol{x})$ is equivalent to solving $\boldsymbol{a} = (a_0, a_1, \ldots, a_k)$, which are coefficients of all possible polynomial basis combined by $x_1, , x_n$ and y, denoted as $\boldsymbol{p}$, where $k = 2^{n+1} - 1$. Multiple groups of anchors $\mathbf{X}$ and $\boldsymbol{y}$ constitute multiple groups of values of polynomial basis, denoted as $\mathbf{P}$, then Equation 1 can be denoted as $\mathbf{P}\boldsymbol{a} = \mathbf{0}$, which is a typical set of linear equations. Next, we introduce two different algorithms that can solve $\boldsymbol{a}$, namely *analytical-based*, and *search-based* algorithm.

3.1 Analytical-based Algorithm

To solve $\mathbf{P}\boldsymbol{a} = \mathbf{b}$, we can simply generate k+1 groups of anchor numbers as $\mathbf{X}$ and LMs' answers as $\boldsymbol{y}$, compute $\mathbf{P}$ based on $\mathbf{X}$ and $\boldsymbol{y}$, and finally get $\boldsymbol{a} = (\mathbf{P})^{-1}\boldsymbol{b}$ when $\mathbf{P}$ is in full rank. But notice that y can be a linear weighted summation of $x_0, \ldots, x_n$, the coefficient matrix $\mathbf{P}$ may not be full-ranked. To address this issue, we generate k groups of anchor numbers and add an additional constraint by setting $|\boldsymbol{a}| = \sum_{i=0}^{k} a_i = 1$. So we augment $\mathbf{P}$ with an all-one vector to $\mathbf{P}^*$ and finally get $\boldsymbol{a} = (\boldsymbol{P}^*)^{-1}\boldsymbol{b}$, where $\boldsymbol{b} = (0, 0, \ldots, 0, 1)$. In practice, randomly sampled groups of anchor numbers can form a full-ranked $\boldsymbol{P}^*$ with a very high probability, and one can even add a buffer by sampling a bit more groups of anchor numbers than k to constitute different $\boldsymbol{P}^*$s for cross validation. The analytic algorithm is theoretically complete to derive arithmetic expressions in our pre-defined problem space. However, in practice, LMs may generate incorrect answers even for anchor numbers, especially the expression is complex, violating the analytic method which necessitates correct answers. To tolerate the constraint, we present a search-based algorithm to solve a noisy linear system. The analytical theory can also well for guiding the hyper-parameter of the search-based algorithm.

3.2 Search-based Algorithm

The search-based algorithm exhaustively explores the search space and finds out the most preferable arithmetic expression. We constrain the search space of $\boldsymbol{a}$ in Equation 1 by: requiring $a_{1-n} \in \{-1, 0, 1\}$ for all coefficients of the non-constant terms, and for coefficient a_0 of constant term C, one can restrict the search range to a pre-defined set, e.g., $a_0 \in \{-100, -1, 0, 1, 100\}$ in our experiments for efficiency, and this is different from the analytic method that can easily solve constants in expressions. Constraints here mean that we only let this search algorithm cover $f(\boldsymbol{x})$ with no more than one constant for efficiency. We then transform all searched polynomial-basis-based equations backwards into expressions because they have one-to-one mappings, e.g., from $x_2 - x_1 - x_2 y = 0$ to $y = 1 - x_1/x_2$. We denote the space of expressions as $\mathbb{F}$, and for each $f_i \in \mathbb{F}$ and each group of anchor numbers $\mathbf{X}_j$ (using m to denote the number of groups), we get y_{ij} by applying f_i to $\mathbf{X}_j$.

We define the prediction error between the target expression $\hat{f}$ and f_i as $\epsilon(\hat{f}, f_i)$, which is calculated by $\epsilon(\hat{f}, f_i) = \sum_j \epsilon_{ij} = \sum_j \mathrm{abs}(\hat{y}_j - y_{ij})$, and the number of occurrence of exact matching as $\boldsymbol{c}_i$. We then find the most preferable expression with the minimum prediction error and the maximum number of exact matching. The corresponding Algorithm can be found in Appendix C.

Table 1: Experimental results on the validation set of DROP dataset.

Language Models	EM(%)	F1(%)
BART [9]	67.4	70.6
w. SOLIS	72.9 (+5.5)	76.1 (+5.5)
T5 [15]	61.0	64.6
w. SOLIS	69.9 (+8.9)	73.5 (+8.9)
TAPEX [10]	76.3	79.3
w. SOLIS	78.5 (+2.2)	81.6 (+2.3)
POET-SQL [14]	76.9	80.0
w. SOLIS	78.2 (+1.3)	82.0 (+2.0)

Table 2: Experimental results of SOLIS *w.* various solving algorithms on the DROP numeric subset.

Language Models	Algorithm	F1(%) on Hard	F1(%) on Total
BART	–	30.4	66.4
	Analytical	46.4 (+16.0)	69.3 (+2.9)
	Search	64.8 (+30.4)	75.2 (+8.8)
POET-SQL	–	66.8	78.4
	Analytical	73.3 (+6.5)	80.0 (+1.6)
	Search	76.9 (+10.1)	81.4 (+3.0)

Table 3: Experimental results of different methods on AddSub and MultiArith.

LLMs	Setting	AddSub	MultiArith
PaLM (540B)	Standard [3]	–	42.2
	Chain [20]	91.9	94.7
GPT-3 (175B)	Zero-Chain [8]	66.6	63.8
	w. SOLIS	89.4 (+22.8)	80.0 (+16.2)
	Chain [20]	88.4	96.7
	w. SOLIS	90.9 (+2.5)	98.7 (+2.0)

Table 4: Case study on derived expressions on DROP.

Intention	Example Question with [Derived Expression]	Proportion
Addition	How many **total** ... were there? $[y = x_1 + x_2 + x_3]$	8.92%
Diff Constant	How many **in percent** ... **weren't** ...?$[y = 100 - x]$	36.49%
Subtraction	How many **more** percentages ... **compared to** ...? $[y = x_1 - x_2]$	54.25%
Composition	How many **more** ... **compared to** ... and ... **combined ?** $[y = x_0 - (x_1 + x_2)]$	0.34%

4 Experiments

Datasets We perform experiments on DROP [6], AddSub and MultiArith, of which the latter two are widely used subsets from MAWPS [17]. DROP is a reading comprehension benchmark that focuses on numerical reasoning.As for MAWPS, it consists of math word problems which also require numerical reasoning ability. On DROP, we inherit the official Exact Match(EM) and F1 to evaluate results; On MAWPS, we use EM to evaluate results. More details can be found in Appendix D.

Backbone and Baselines On DROP, we adopt two kinds of LMs as our backbones, including (i) Vanilla LMs: BART [9] and T5 [15], (ii) Reasoning LMs: TAPEX [10] and POET [14]. All models are fine-tuned on the DROP train set. On AddSub and MultiArith, we adopt GPT-3 [1] with different prompting techniques as our backbones: Chain-of-Thought Prompting (Chain) [20] and the Zero-shot Chain-of-Thought Prompting (Zero-Chain) [8]. We also compare our results to the PaLM model [3].

Experimental Results We first evaluate suggested solving algorithms via their performance on the DROP subset whose answers are numbers (i.e., numeric subset). Meanwhile, we select cases in which the answer is greater than 1000, identify them as "hard" cases, and additionally report the performance. As shown in Table 2, all of our proposed algorithms significantly improve the performance of LMs, especially in hard cases. The full results of the performance comparison can be found in Appendix E. Notably, since the search-based algorithm is the most effective, we apply it as the default algorithm in SOLIS. Table 1 shows the experimental results of different models on DROP dataset. As shown, SOLIS can bring consistent and significant improvements over all backbone LMs, especially for the vanilla LMs. On T5, for instance, it could be boosted by a maximum of 8.9% with SOLIS. Table 3 presents the experimental results on AddSub and MultiArith. The results indicate that our approach is surprisingly effective for giant LMs, and can further boost the chain-of-thought.

Model Analysis In addition to performance improvement, SOLIS features the ability to derive an arithmetic expression for each question, whereas no such information is available during training. To better understand if these expressions align with question intentions, we collect all derived expressions on DROP and categorize them into four types in Table 4. As demonstrated, the majority of expressions contain addition and subtraction between variables and constants, which are largely consistent with the question intention, highlighting the superior interpretability of SOLIS.

5 Conclusion

In this work, we present SOLIS, a framework which can elicit numerical reasoning in language models at test time. Motivated by the fact that language models excel at simple numbers, SOLIS uses simple numbers as anchors to inversely derive the implicitly inferred arithmetic expressions from language models, and subsequently apply these expressions to the original numbers to perform numerical reasoning. Experimental results on several numerical reasoning benchmarks demonstrate that SOLIS can be integrated to a variety of language models, and can greatly improve their performance in zero-shot, few-shot, and fine-tuning scenarios.

References

[1] 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. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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)*, pages 2368–2378, 2019.

[7] Mor Geva, Ankit Gupta, and Jonathan Berant. Injecting numerical reasoning skills into language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 946–958, 2020.

[8] 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.

[9] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, 2020.

[10] Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. TAPEX: Table pre-training via learning a neural SQL executor. In *International Conference on Learning Representations*, 2022.

[11] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`.

[12] Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. NumGLUE: A suite of fundamental yet challenging mathematical reasoning tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3505–3523, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.246. URL `https://aclanthology.org/2022.acl-long.246`.

[13] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. *arXiv preprint arXiv:2102.13019*, 2021.

[14] Xinyu Pi, Qian Liu, Bei Chen, Morteza Ziyadi, Zeqi Lin, Yan Gao, Qiang Fu, Jian-Guang Lou, and Weizhu Chen. Reasoning like program executors. *arXiv preprint arXiv:2201.11473*, 2022.

[15] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67, 2020.

[16] Yasaman Razeghi, Robert L Logan IV, Matt Gardner, and Sameer Singh. Impact of pretraining term frequencies on few-shot reasoning. *arXiv preprint arXiv:2202.07206*, 2022.

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

[18] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K. Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2660–2673, 2017. doi: 10.1109/TNNLS.2016.2599820.

[19] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do nlp models know numbers? probing numeracy in embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5307–5315, 2019.

[20] 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.

[21] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[22] Ori Yoran, Alon Talmor, and Jonathan Berant. Turning tables: Generating examples from semi-structured tables for endowing language models with reasoning skills. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6016–6031, 2022.

A Preliminary Study Details

In this section, we will first demonstrate the brittleness of language models' ability on arithmetically-related tasks. Unlike arithmetic benchmarks such as AddSub or MultiArith [17] which contain natural language context for each sample, we directly generate and feed the arithmetic expressions and test the performance on language models. This is done to reduce potential perturbing factors and highlight the models' calculating ability. We impose constraints on the complexity of the expressions: we only study the four fundamental operations, and demand no more than 4 operands, where each operand's integer range is less then $10,000$ and floating point precision is less than 4. To conduct a systematic investigation, we first produce $\mathbb{F}$ which represents the set of all the expressions satisfying our constraints. We randomly sample numbers within the limits of range and precision as the operands. For one expression $f \in \mathbb{F}$ with a specified range and precision, we randomly generate 50 samples. We evaluate the language model on these samples and denote this synthesized task as MathExp which stands for **Math Exp**ressions.

We sample a maximum of 50 expressions for each different settings of complexity, and test these samples using large scale language model GPT-3 [1]. We conduct the study on GPT-3 in a few-shot manner: to unleash its potential, we pre-pend 10 to 20 expressions (having the same f, integer range, and floating point precision as the tested sample) together with the answers as the prompt. We then call the OpenAI API[2] to get all the predictions, and evaluate the performance accordingly.

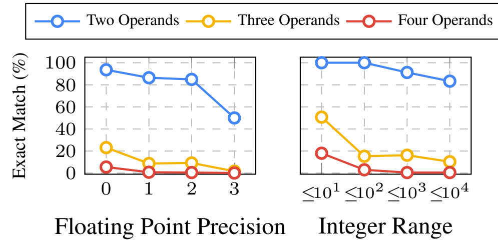

Figure 2: Performance with different floating point precision (left) and integer range (right).

Results in Figure 2 indicate that even the latest powerful GPT-3(*Code-Davinci-002*) fails to achieve a satisfactory performance: (i) the prediction accuracy decreases largely as the number gets more complex, i.e., integer range or floating point precision of operands increases; (ii) the prediction accuracy also drops dramatically as the arithmetic relationship getting more complex, i.e., number of operands increases.

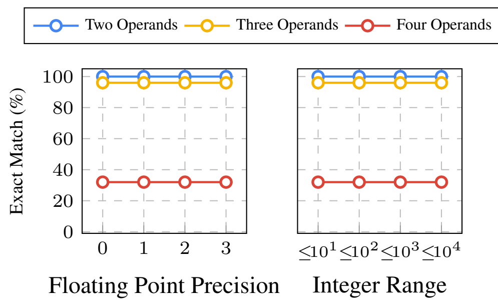

Figure 3: Performance over different floating point precision (left) and integer range (right) on MathExp of GPT-3 *w.* search-based algorithm.

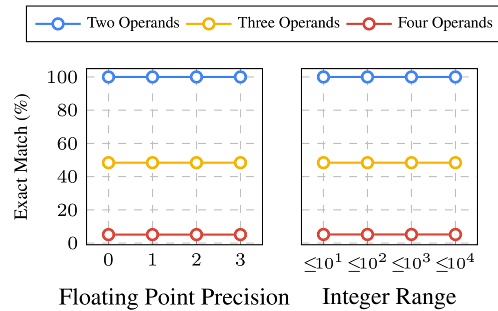

Figure 4: Performance over different floating point precision (left) and integer range (right) on MathExp of GPT-3 *w.* analytical-based algorithm.

We also present the model performance on MathExp of GPT-3 with different solving algorithms in Figure 3 and Figure 4. We can conclude that: (1) both algorithms are not sensitive with either the

[2] https://openai.com/api

floating point precision or the integer range; (2) the search-based algorithm is most robust than the analytical-based algorithm with respect to the number of operands.

B Operand Proposal Details

In Section 2, we mention that the textual context on a realistic dataset may be noisy, i.e., contains irrelevant numbers, thus we need to locate the operand number first. We substitute 10 times for each number appearing in the paragraph, if the output gives ≥ 3 different prediction numbers out of 10, we decide the current tested number is involved to the answer. Moreover, we substitute numbers following a template: suppose the original number x is with precision $\mathtt{p}$, then the substituted numbers can be represented as $x + k \cdot 10^{\mathtt{p}}$, where $k \in \{-5, -4, -3, -2, -1, 1, 2, 3, 4, 5\}$.

C Search-based Algorithm

Algorithm 1 SEARCH

Input: parameters $\mathbf{X}, \hat{\boldsymbol{y}}, \mathbb{F}, c_{threshold}$
Output: Most preferable expression $\widetilde{f}$

1: **while** $j < m$ **do**
2:     **for** $f_i \in \mathbb{F}$ **do**
3:         $y^*_{ij} \leftarrow f_i(X_j)$
4:         $c_i \leftarrow c_i + \mathbf{1}(y^*_j == \hat{y}_{ij})$
5:         $\epsilon_i \leftarrow \epsilon_i + |y^*_j - \hat{y}_{ij}|$
6:     **end for**
7:     $j \leftarrow j + 1$
8: **end while**
9: $i^*_c \leftarrow \arg\max \boldsymbol{c},\ i^*_\epsilon \leftarrow \arg\min \epsilon$
10: **if** $c_{i^*} \geq c_{threshold}$ **then** $\widetilde{f} \leftarrow f_{i^*_c}$
11: **else** $\widetilde{f} \leftarrow f_{i^*_\epsilon}$
12: **end if**

D Experiments

D.1 Experimental Setup

Table 5: Statistics of DROP dataset

Dataset	Train		Dev	
	# Questions	# Docs	# Questions	# Docs
DROP	$77,409$	$5,565$	$9,536$	582

Table 6: Statistics of MAWPS dataset

Subset	# Questions
AddSub	395
MultiArith	600

For BART, we implement the fine-tuning methods using the Huggingface transformers library [21] on 4 V100 16GB GPUs. We use $\texttt{BART}_{\text{LARGE}}$[9] as our backbone. We use same-scale reasoning-pretrained POET-SQL and TAPEX models in experiments. For T5, we implement its fine-tuning on the Huggingface transformers library on A100 GPUs. We use $\texttt{T5}_{\text{LARGE}}$ [15] as our backbone.

D.2 Experimental Details on DROP

Fine-tuning Details For all fine-tuning methods, we select the default max token length for each model. We set the max token length of generation as 96. To save training time, we set early stop mechanism: we evaluate the EM and F1 score per 500 or 1000 steps, if the performance does not increase in the latest 20 evaluations, we stop the training and save the best checkpoint.

On DROP, we pre-pend the question to the given paragraph. For multi-span answer, we insert ";" between each span and make up the final answer. For $\texttt{T5}_{\text{LARGE}}$, we also insert "</s>" token between the question and the given paragraph. Since most LMs' checkpoints on DROP is currently not off-the-shelf, we re-implement them and compare to the results reported in previous works. We present the comparison results in Table 7.

Table 7: Performance Comparison on DROP between reported results in previous works and our re-implementation. Results marked with * represent our re-implementation results.

Models	EM (%)	F1 (%)
BART [14]	66.2	69.2
BART*	67.4	70.6
T5 [22]	–	64.6
T5*	61.0	64.6
POET-SQL [14]	77.7	80.6
POET-SQL*	76.9	80.0

D.3 Hyperparameter Selection

For fine-tuning, we apply Adam [11] optimizer. The fine-tuning epochs are set as 50. For BART models (i.e., BART and POET-SQL), we follow previous works [14] to set the batch size as 128 and the learning rate as 3×10^{-5}. For T5, we decrease the batch size to 32 due to the computational budget. The early stop technique is used to save training time. For GPT-3 API, we keep the temperature as default setting 0, and set the maximum output tokens to 128. As for anchor number groups: the group size is $6/8/10$ corresponding to corresponding to $2/3/4$ operands on DROP; the group size is 4 on AddSub, and 10 on MultiArith because MultiArith requires more compositional operations.

D.4 Design Choices on DROP

Following previous work, we apply two general-purpose numerical designs on the DROP dataset. First, we employ the character-level rather than subword-level number representation, which proves to be more effective [19, 14]. Second, we employ the reverse decoding technique, which proves to be a successful design to mimic arithmetic carry [7]. Meanwhile, as mentioned above, the search-based algorithm has difficulties in covering expressions including constants. Considering the constant 100 is frequently used for percentage calculations (e.g., "How many percent of the national population does not live in Bangkok?"), we add it to be one candidate in DROP.

E More Results on DROP

We present the performance breakdown of F1 on dev set of DROP in Table 8 Apart from fine-tuning models on DROP dataset, we also use GPT-3 to conduct a study on few-shot learning. We pre-pend 10 random training samples in train set, and run all cases where answer type equals to "number". We also apply our search-based algorithm on GPT-3. To save API calling time, we only substitute the number for one time. Table 9 presents the F1 score comparison.

We also summarize common calculation error cases in our tested language models and present some of them for case study in Table 10, which again illustrates the unreliability of language models.

Table 8: Breakdown of model F1 score by answer types on the dev set of DROP.

Models	Number	Span	Spans	Date	Total
BART	66.3	80.3	66.0	56.7	70.6
w. SOLIS	75.2	80.5	66.7	55.7	76.1
T5	55.5	81.6	73.0	53.5	64.6
w. SOLIS	69.8	81.8	73.9	53.5	73.5
TAPEX	77.8	84.3	72.9	62.8	79.3
w. SOLIS	81.4	84.4	73.0	61.7	81.6
POET-SQL	78.4	84.6	76.6	63.4	80.0
w. SOLIS	81.4	84.9	76.9	62.6	82.0

Table 9: Performance of GPT-3 *w.* SOLIS on the DROP numeric subset.

Language Model	Algorithm	F1(%) on Hard	F1(%) on Total
GPT-3 (175B)	-	42.5	64.7
	Search	59.9 (+17.4)	68.7 (+4.9)

Table 10: Common calculation error cases on DROP dataset.

Error Type	Example	Prediction	Label
Carry Error	. . . the size of the black-white IQ gap in the United States decreased from **16.33** to **9.94** IQ points. . . . **Q:** How many IQ points did the black-white IQ gap decrease in the United States in a 2013 analysis of the National Assessment of Educational Progress?	6.49	6.39
Missing High Digit	. . . The Department of Tourism recorded **26,861,095** Thai and **11,361,808** foreign visitors to Bangkok in 2010. . . . **Q:** How many more Thai visitors did Bangkok have in 2010 compared to other foreign visitors?	499287	15499287
Extra Integer digit	. . . Rayner nailed a **23**-yard field goal . . . Rayner got a **54**-yarder and a **46**-yarder to end the half . . . **Q:** How many total yards of field goals did Dave Rayner have?	111113	123
Extra Float Number Digits	. . . have estimated the IQ means of 17-year-old black, white, and Hispanic students to range respectively from **90.45-94.15** . . . **Q:** How many points difference is the IQ range in 17-year-old black students?	3.75	3.7
Insufficient Precision	. . . The Diocese of Karelia has **22,000** church members in **12** parishes. . . . **Q:** How many church members approximately are in each one of the 12 parishes?	1833	1833.33