INVERSELY ELICITING NUMERICAL REASONING IN LANGUAGE MODELS VIA SOLVING LINEAR SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Numerical reasoning over natural language has been a long-standing goal for the research community. However, recent language models have proven difficult to reliably generalize to a broad range of numbers, although they have shown proficiency in reasoning over common and simple numbers. In this paper, we propose a novel method to elicit and exploit the numerical reasoning knowledge hidden in pre-trained language models using simple anchor numbers. Concretely, we first leverage simple numbers as anchors to probe the implicitly inferred arithmetic expressions from language models, and then explicitly apply the expressions on complex numbers to get corresponding answers. To inversely elicit arithmetic expressions, we transform and formulate the task as an analytically solvable linear system. 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 (Devlin et al., 2018; Brown et al., 2020b; Chowdhery et al., 2022), and recent works even explore to use LMs as a general-purpose interface for diverse modalities (Hao et al., 2022; Xie et al., 2022; He et al., 2022). But when it comes to reasoning about numbers, the crucial parts of text, tables, and knowledge bases, the performance of LMs slumps. A key challenge of numerical reasoning for now is number calculation. Even rational numbers, a small subset of real numbers, readily constitute an infinite space that cannot be completely covered by pre-training corpora, hence posing a significant obstacle to LMs. Recent works have shown strong context understanding capabilities of LMs in numerical reasoning datasets (Dua et al., 2019; Cobbe et al., 2021), but LMs are still far from being robust on implicit numerical calculation: as numbers grow bigger and more complex, LMs are more likely to fail, e.g., $8,534.5 + 17.85$; and even for small number additions, e.g., $512+128$ and $513+129$, LMs are not stable enough to produce the correct answer consistently. Similar observations are also reported by Razeghi et al. (2022), showing that end-to-end LMs easily fail to calculate numbers that rarely appear in pre-training corpora.

Fortunately, by reverse thinking, we have a positive perspective: with the exact same context, LMs are significantly more accurate and stable on simple numbers - typically small integers that appear frequently in the pre-training corpora - than complex numbers, indicating that LMs have a strong capability of applying arithmetic results to simple numbers after pre-training. 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 complex numbers*. Specifically, as Figure 1 illustrates, when detecting complex numbers ($10,477$ and $7,459$) that are challenging for LMs, to first replace them by anchor numbers(10 and 7, etc) and use LMs to output answers (3, etc) that are more much accurate than complex numbers, then inversely elicit the hidden arithmetic relationship ($x_1 - x_2$) implicitly inferred by LMs through anchor inputs/outputs (10,7,3, etc), and finally explicitly doing the arithmetic using the initial complex numbers ($10,477-7,459$) to produce the precise 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.

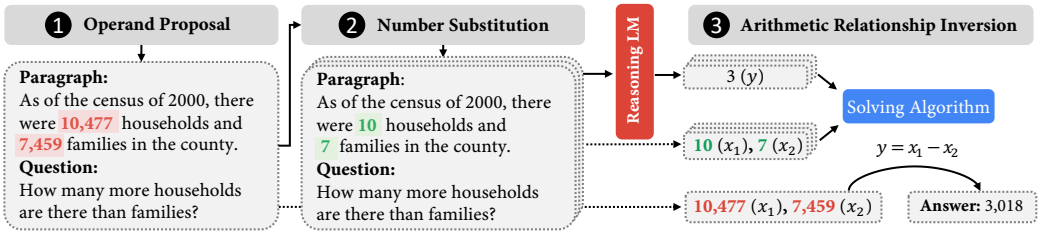

Figure 1: The illustration of our proposed framework, which elicits numerical reasoning in language models via **So**lving **Li**near **S**ystems (SOLIS).

This paper introduces a new idea of eliciting and exploiting the numerical reasoning knowledge hidden in pre-trained LMs through probing with simple anchor numbers. Importantly, our framework does not need any additional model training or labeled data, because it simply works during the test-time inference phase, and it is portable to all existing fine-tuned/few-shot/zero-shot LMs with decoders. Thus it is significantly different from existing neural symbolic methods that need continuous training (Liang et al., 2016) and program synthesis from examples that need specific and human-provided input-output pairs for each example in the inference phase (Gulwani, 2011).

To inversely elicit arithmetic relationships in LMs through anchor numbers, we propose SOLIS, a novel method to transform and formulate this problem to a linear system that can be straightforwardly solved in an analytic way. Alternative search-based and heuristic-based methods are further devised to promote robustness for noisy linear systems. Experimental results show significant and consistent gains over various language models and diverse zero-shot, few-shot and fine-tuning settings on several representative numerical reasoning datasets.

2 PRELIMINARY STUDY

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 (Roy & Roth, 2015) 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 (Brown et al., 2020a). 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[1] to get all the predictions, and evaluate the performance accordingly.

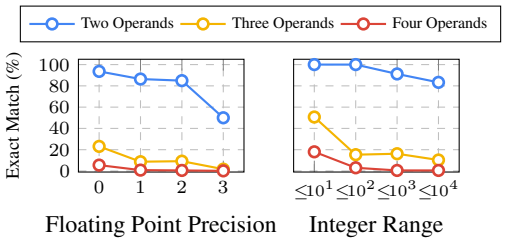

Figure 2: Performance with different floating point precision (left) and integer range (right).

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

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

precision of operands increases; (ii) the prediction accuracy also drops dramatically as the arithmetic relationship getting more complex, i.e., number of operands increases. In Appendix A, we also present the performance with our SOLIS framework, which is more robust to influence of floating point precision and integer range.

3 NUMERICAL REASONING VIA SOLVING LINEAR SYSTEMS

The preliminary study demonstrates that the current language models are vulnerable to complex numbers. For example, they have no chance to guess the answer to the sum of two floating point numbers with three decimal places. However, the language model can perform reliably 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.

3.1 METHOD OVERVIEW

As mentioned above, our method can be integrated into language models in a plug-and-play manner at test time. For the sake of clarification, in the following we refer to LMs that can steadily perform numerical reasoning as reasoning LMs. They can be either LMs obtained by fine-tuning on datasets involving numerical reasoning, or LMs that perform numerical reasoning via in-context learning.

As shown in Figure 1, our method generally 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[2] 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 as observed data, we can inversely derive the internal reasoning flow from the reasoning LM, i.e. the arithmetic expression between the operands (e.g., $y = x_1 - x_2$). By applying the expression on the original numbers, the answer to the original paragraph can be obtained.

3.2 OPERAND PROPOSAL

There are often many numbers involved in a paragraph, and it is quite challenging to model the arithmetic relationships among all these numbers simultaneously. Consequently, it is important during the operand proposal step to trim the prospective operands to a manageable size. A straightforward strategy would be to select only the numbers pertinent to the answer as candidate operands, which is not trivial in practice since there is no intermediate supervision on the relevance between each number and the answer.

To address the issue, we provide a novel technique that employs number perturbation and the reasoning LM to measure the relevance systematically. It is largely inspired by prior works that leverage an image classifier to quantify the relevance of pixels with image categories (Samek et al., 2017) and its application on natural language tasks (Liu et al., 2021). In their 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., $98.5 \rightarrow 98.6$) and evaluate whether the model prediction changes correspondingly. Despite the fact that the reasoning LM hardly perform accurate calculations over numbers, we observe that LMs have strong context understanding capabilities about numbers and are sensitive to slight changes in the numbers used to forecast answer. More details about the operand proposal mechanism can be found in Appendix B.

[2]We use the terms *number* and *operand* interchangeably.

3.3 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.

3.4 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 four most prevalent operators in numerical reasoning (Dua et al., 2019). 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 answer, 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, which will be elaborated in Section 4. Finally, the answer can be reached in a trustworthy and interpretable manner by applying the derived expression to the original numbers.

4 SOLVING ALGORITHM

In this section, we introduce three algorithms that can derive expressions merely from anchor numbers and anchor answers, namely *analytical-based*, *search-based* and *heuristic-based* algorithm.

4.1 FORMULATION

Formally, 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 goal 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:

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

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.

4.2 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 $x_0, \ldots, x_n$ by itself, the coefficient matrix $\mathbf{P}$ may not be full-ranked. To address this, 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 method is theoretically complete to deduce arithmetic expressions in our pre-defined problem space. But in practice, LMs may produce incorrect results even for anchor numbers, especially when given a complex expression, so as to violate the analytic method which needs purely correct anchor answers. To best tolerate them, we then propose search-based and heuristic-based methods to better solve a noisy linear system. Gladly, the analytic method theoretical supports other methods in aspects such as guiding the number of anchors to sample to ensure a unique expression.

4.3 SEARCH-BASED ALGORITHM

The search-based algorithm exhaustively explores the search space and finds out the most preferable arithmetic expression in the space. 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 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 $\epsilon(\hat{f}, f_i) = \sum_j \epsilon_{ij} = \sum_j \text{abs}(\hat{y}_j - y_{ij})$, 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. Specifically, when the number of exact matching exceeds a pre-defined $c_{threshold}$, we pick the expression f_i with the highest $\boldsymbol{c}_i$; otherwise, we pick the expression f_i with the lowest ϵ_i. The search process is sketched in Algorithm 1.

Algorithm 1 SEARCH

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

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

This method is robust for probably incorrect predictions, i.e., when model does not have sufficient number of exact matching, it is still capable to return the most nearest expression by selecting the one with the minimum error. However, the search-based method can be challenged by exponentially explosive search space when the number of operands surges, and it's not efficient to search constant numbers that has a wide and even infinite range, neither.

4.4 HEURISTIC-BASED ALGORITHM

In this section, we introduce a heuristic-based algorithm, simulated annealing, which is efficient and does not need to search for the whole problem space, though it may produce sub-optimal results given a limited number of exploration steps. We follow the formulation introduced in Section 4.1 and proposed a optimization target $\mathcal{L}_H$ to measure the L1 loss of $\boldsymbol{Pa}$. The pipeline includes: (1) randomly initialize $\boldsymbol{a}$ with values {-1, 0, 1} and calculate initial $\mathcal{L}_H$; (2) randomly select i from 0 to k and perturb a_i by adding or subtracting a constant number (we use 1 here); (3) calculate new $\mathcal{L}_H$,

and adopt the perturbation with a large probability if $\mathcal{L}_H$ decreases and with a low probability if it increases, balanced by a pre-defined temperature T, which decreases over steps; (4) return $\boldsymbol{a}$ if the number of steps is enough or $\mathcal{L}_H$ equals to zero, otherwise repeat from step 1. Note that, we restrict coefficients in $\boldsymbol{a}$ to be integers for simplicity, so different from the analytical method restricting $\sum_{i=0}^{k} a_i = 1$, we ensure only one of the coefficients of y-related polynomial basis $\{y, x_1y, \ldots, x_1x_2 \ldots x_ny\}$ to be non-zero (with a static value 1) and at least two coefficients in $\boldsymbol{a}$ are non-zero during the whole initialization and perturbation process to avoid some infeasible local optimal.

In summary, Table 1 shows the strong and weak points of these algorithms. In the problem space introduced in Section 3.4 within at most four operands, the search-based method does not have scalability issues, so it achieves best performance in our experiments because it's robust to LMs' predictions and can retrieve optimal expression through exhaustive search except rare constants.

Table 1: Comparison of solving algorithms.

	Optimum	Robustness	Scalability
Analytical	✓	✗	✓
Search	✓	✓	✗
Heuristic	✗	✓	✓

5 EXPERIMENTS

In this section, we integrate SOLIS with various language models as backbones and evaluate the effectiveness of SOLIS on two well-known numerical reasoning benchmarks.

5.1 EXPERIMENTAL SETUP

Datasets We perform experiments on DROP (Dua et al., 2019), AddSub and MultiArith, of which the latter two are widely used subsets from MAWPS (Roy & Roth, 2015). DROP is a reading comprehension benchmark that focuses on numerical reasoning and has a variety of answer types, including *span*, *number*, and *date*. The experimental results of DROP are evaluated with the official evaluation metrics Exact Match (EM) and F1. As for MAWPS, it consists of math word problems which also require numerical reasoning ability. The subset AddSub features relatively easier numerical reasoning, whereas MultiArith necessitates multi-step numerical calculations. The EM metric is used to evaluate the results of AddSub and MultiArith. More details can be found in Appendix C.

Backbone and Baselines The **fine-tuning** evaluation is conducted on DROP, where we adopt two kinds of fine-tuned LMs as backbones, including (i) Vanilla LMs: BART (Lewis et al., 2020) and T5 (Raffel et al., 2020), (ii) Reasoning LMs: GENBERT Geva et al. (2020), TAPEX (Liu et al., 2022) and POET (Pi et al., 2022). We compare the performance of our method with previous specialized models designed for DROP, such as NumNet (Ran et al., 2019), NeRd (Chen et al., 2020b), MTMSN (Hu et al., 2019) and QDGAT (Chen et al., 2020a). All models are fine-tuned on the DROP train set, and the best validation set performance is reported. The **few-shot** and **zero-shot** evaluations are done using AddSub and MultiArith, where we adopt GPT-3 code-davinci-002 (Brown et al., 2020a) with different prompting as our backbones: Few-shot Chain-of-Thought Prompting (Few-shot Chain) (Wei et al., 2022b) and Zero-shot Chain-of-Thought Prompting (Zero-shot Chain) (Kojima et al., 2022). These models perform numerical reasoning by in-context learning, and the few-shot demonstrations are the 8 samples released by Wei et al. (2022b).

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 (Wallace et al., 2019; Pi et al., 2022). Second, we employ the reverse decoding technique, which proves to be a successful design to mimic arithmetic carry (Geva et al., 2020). 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.

5.2 EXPERIMENTAL RESULTS

Since our work focuses on addressing arithmetic problems, we first evaluate suggested solving algorithms via their performance on the DROP subset whose answers are numbers (i.e., numeric subset).

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

LM	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)
	Heuristic	52.8 (+22.4)	71.7 (+5.3)
POET-SQL	–	66.8	78.4
	Analytical	73.3 (+6.5)	80.0 (+1.6)
	Search	76.9 (+10.1)	81.4 (+3.0)
	Heuristic	73.0 (+6.2)	80.5 (+2.1)

Table 3: Fine-tuning evaluation on the validation set of DROP dataset.

Models	EM(%)	F1(%)
Specialized Models		
NumNet (Ran et al., 2019)	64.9	68.3
MTMSN (Hu et al., 2019)	76.7	80.5
NeRd (Chen et al., 2020b)	78.6	81.9
QDGAT (Chen et al., 2020a)	84.1	87.1
Vanilla LMs		
BART (Lewis et al., 2020)	67.4	70.6
w. SOLIS	72.9 (+5.5)	76.1 (+5.5)
T5 (Raffel et al., 2020)	61.0	64.6
w. SOLIS	69.9 (+8.9)	73.5 (+8.9)
Reasoning LMs		
GENBERT (Geva et al., 2020)	68.8	72.3
w. SOLIS	70.5 (+1.7)	74.4 (+2.1)
TAPEX (Liu et al., 2022)	76.3	79.3
w. SOLIS	78.5 (+2.2)	81.6 (+2.3)
POET-SQL (Pi et al., 2022)	76.9	80.0
w. SOLIS	78.2 (+1.3)	82.0 (+2.0)

Meanwhile, we select cases in which the answer is greater than 1000, identify them as "hard" cases, and additionally report the average performance on them. As shown in Table 2, all of our proposed algorithms significantly improve the performance of LMs, especially in hard cases. For example, the search-based algorithm boosts BART with an absolute 30.4% improvement on hard cases. The full results of the performance comparison can be found in Appendix D. Notably, since the search-based algorithm is the most effective, we apply it as the default algorithm in SOLIS.

Table 3 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. Taking the T5 model as an example, it could be boosted by a maximum of 8.9% with SOLIS. Even for POET-SQL which are already pre-trained for numerical reasoning, our method yields a 2.0% F1 improvement, pushing the best LM performance to 82.0% F1. Table 4 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 performance of chain-of-thought prompting.

Table 4: Few-shot evaluation on the AddSub and MultiArith dataset.

Language Model	Setting	AddSub	MultiArith
PaLM (540B)	Few-shot Vanilla (Chowdhery et al., 2022)	–	42.2
	Few-shot Chain (Wei et al., 2022b)	91.9	94.7
GPT-3 (175B)	Zero-shot Chain (Kojima et al., 2022)	66.6	63.8
	w. SOLIS	89.4 (+22.8)	80.0 (+16.2)
	Few-shot Chain (Wei et al., 2022b)	88.4	96.7
	w. SOLIS	90.9 (+2.5)	98.7 (+2.0)

Table 5: Case study on derived expressions using POET-SQL *w.* SOLIS on DROP. Listed are, the intention, the example question with intention trigger words (i.e., the **colorful** spans) and the derived expression, and the proportion of each intention.

Question Intention	Example Question with [Derived Expression]	Proportion
Addition	How many **total** yards of touchdown passes were there? $[\mathtt{y} = \mathtt{x}_1 + \mathtt{x}_2 + \mathtt{x}_3]$	8.92%
Diff Constant	How many **in percent** in the county from the census of 2000 **weren't** English? $[\mathtt{y} = 100 - \mathtt{x}]$	36.49%
Subtraction	How many **more** percentages of people were germans **compared to** irish? $[\mathtt{y} = \mathtt{x}_1 - \mathtt{x}_2]$	54.25%
Composition	How many **more** Albanian citizens were there **compared to** Bulgarian and Georgia citizens **combined ?** $[\mathtt{y} = \mathtt{x}_0 - (\mathtt{x}_1 + \mathtt{x}_2)]$	0.34%

6 MODEL ANALYSIS

Arithmetic Relationship Inversion 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 and categorize them into four types in Table 5. 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.

Solving Algorithm Robustness The possibility that the anchor answers provided by reasoning LMs are inaccurate presents a challenge for the solving algorithms. To measure the robustness of our solving algorithms, we roughly decrease the probability that anchor answers are correct by decreasing the number of few-shot demonstrations in Figure 3. As shown, even though the backbone LM performance drops to 60.0%, the improvement of SOLIS is still as high as to 5.1%, suggesting its robustness.

Number Substitution To study the impact of different factors during the number substitution stage, we conduct experiments on MathExp in Figure 4. As demonstrated, expanding the range of anchor numbers results in a minor performance drop, showing that the reasoning LM is more familiar with small integers. Furthermore, increasing the size of anchor number groups gives a large improvement on the performance, especially when there are four operands.

Limitation Discussion The first limitation of our framework is that we cannot support expressions that cannot be solved with linear systems. For example, with respect to the question "How many yards was Donovan McNabb's longest rushing TD?", the expected expression $[\mathtt{y} = \max_{\mathtt{i}}(\mathtt{x}_{\mathtt{i}})]$ is not supported by SOLIS. Second, the framework is less efficient when there are many operands. On the one hand, the group of anchor numbers would be quite huge, making the algorithm's runtime unacceptable. For example, when expanding to 5 operands, number substitution must be performed at least 50 times. On the other hand, for the search-based algorithm, the search space will increase exponentially, making the algorithm impracticable. Last, we assume a certain level of numeracy understanding of the reasoning LM. Therefore, if the reasoning LM is unable to comprehend the numeracy relationship, our method would not work well.

7 RELATED WORK

Numerical Understanding and Reasoning via Specialized Models Since our work focuses on numerical reasoning, it is related to previous works on numerical understanding, which has been found to be important for deep learning models (Spithourakis & Riedel, 2018; Wallace et al., 2019; Naik et al., 2019; Zhang et al., 2020; Sundararaman et al., 2020; Thawani et al., 2021). In general, previous work on numerical understanding aims to develop better numeracy embeddings that accurately reflect their properties, so they typically evaluate numeracy embeddings on synthetic tasks. For example, Wallace et al. (2019) proposes List Maximum, an evaluation task by predicting the index of the maximal number in a list of five numeracy embeddings. When it comes to downstream

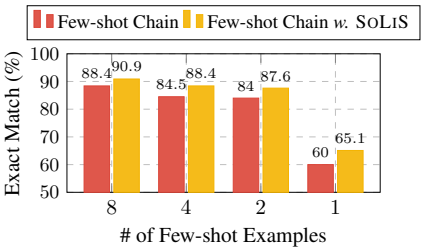

Figure 3: Experimental results of Few-shot Chain with or without SoLiS on AddSub as the number of few-shot examples decreases.

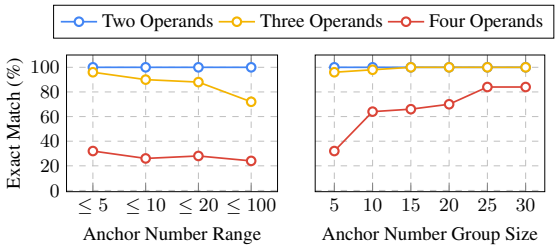

Figure 4: The experimental results of SoLiS on MathExp with different choices of anchor number range (**left**) and anchor number groups (**right**).

applications, the majority of previous works (Jiang et al., 2020; Duan et al., 2021) are studied on classification or regression tasks. Differently, numerical reasoning involves flexible answers that go beyond classification (e.g., requiring complex calculation), which is our focus. As for numercial reasoning, previous works generally design trainable specialized modules and equip LMs with them to tackle different kinds of numerical reasoning problems (e.g., counting). While these methods work well on specific datasets (Dua et al., 2019; Andor et al., 2019; Hu et al., 2019; Ding et al., 2019), they are hardly suited across different datasets and backbone LMs (Chen et al., 2020b). Differently, since our method does not require additional model training, it is applicable to almost all models, even those that only provide an inference interface (e.g., GPT-3). As for methods that first generate programs or logic forms, it is quite laborious to define domain-specific language and collect corresponding training data (Berant et al., 2013). Unlike them, our methods does not require extra annotated programs. Instead, our method allows for the program discovery from examples via solving linear systems.

Numerical Reasoning via Pre-training This line of work always focuses on the pre-training of language models with corpus which involves reasoning. The corpus can be reasoning-oriented natural language texts from Internet (Deng et al., 2021; Lewkowycz et al., 2022), human-designed templates filled by different data sources (Geva et al., 2020; Yoran et al., 2022), or programs with rich reasoning semantics (Liu et al., 2022; Pi et al., 2022). Although this kind of pre-training allows language models to perform better reasoning, they still require considerable computation budgets during pre-training and may still be challenged by complex numbers. In contrast, our method is efficient since it can be integrated into existing models without further training or pre-training.

Numerical Reasoning in Giant Language Models Recent works demonstrate that with proper prompting, giant language models (e.g., GPT-3) perform much better than smaller ones on several reasoning tasks (Wei et al., 2022b;a; Kojima et al., 2022; Li et al., 2022; Zhou et al., 2022; Wang et al., 2022). For example, with the chain-of-thought prompting, the few-shot PaLM model (Chowdhery et al., 2022) can beat the previous best fine-tuned model on math word problems. However, their conclusions do not generalize to non-giant language models. Different from them, our method can be simultaneously applied to language models ranging from millions (e.g., BART) to billions (e.g., GPT-3). Moreover, our work is orthogonal to these giant LMs and can be complementary to each other. For example, Section 5 shows that our approach can further boost the numerical reasoning capability of GPT-3 with chain-of-thought prompting.

8 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 usually 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 complex numbers to perform numerical reasoning. With modeling the expression derivation as solving linear systems, we propose three kinds of algorithms to achieve SoLiS with noisy signals. 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 under zero-shot, few-shot, and fine-tuning scenarios. Our work provides a new perspective towards tackling numerical reasoning, which can be potentially applied to more language models and numerical reasoning tasks.

REFERENCES

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. Giving bert a calculator: Finding operations and arguments with reading comprehension. 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)*, pp. 5947–5952, 2019.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1533–1544, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

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 (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020a.

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, 2020b.

Kunlong Chen, Weidi Xu, Xingyi Cheng, Zou Xiaochuan, Yuyu Zhang, Le Song, Taifeng Wang, Yuan Qi, and Wei Chu. Question directed graph attention network for numerical reasoning over text. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6759–6768, 2020a.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V Le. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *ICLR*, 2020b.

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

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

Xiang Deng, Yu Su, Alyssa Lees, You Wu, Cong Yu, and Huan Sun. ReasonBERT: Pre-trained to reason with distant supervision. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6112–6127, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.494.

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.

Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive graph for multi-hop reading comprehension at scale. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2694–2703, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1259.

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

Hanyu Duan, Yi Yang, and Kar Yan Tam. Learning Numeracy: A Simple Yet Effective Number Embedding Approach Using Knowledge Graph. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 2597–2602, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.221. URL https://aclanthology.org/2021.findings-emnlp.221.

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*, pp. 946–958, 2020.

Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46(1):317–330, 2011.

Yaru Hao, Haoyu Song, Li Dong, Shaohan Huang, Zewen Chi, Wenhui Wang, Shuming Ma, and Furu Wei. Language models are general-purpose interfaces. *arXiv preprint arXiv:2206.06336*, 2022.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16000–16009, 2022.

Minghao Hu, Yuxing Peng, Zhen Huang, and Dongsheng Li. A multi-type multi-span network for reading comprehension that requires discrete reasoning. 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)*, pp. 1596–1606, 2019.

Chengyue Jiang, Zhonglin Nian, Kaihao Guo, Shanbo Chu, Yinggong Zhao, Libin Shen, and Kewei Tu. Learning Numeral Embedding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 2586–2599, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.235. URL https://aclanthology.org/2020.findings-emnlp.235.

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.

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*, pp. 7871–7880, 2020.

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

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

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *arXiv preprint arXiv:1611.00020*, 2016.

Qian Liu, Dejian Yang, Jiahui Zhang, Jiaqi Guo, Bin Zhou, and Jian-Guang Lou. Awakening latent grounding from pretrained language models for semantic parsing. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 1174–1189, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.100.

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.

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.

Aakanksha Naik, Abhilasha Ravichander, Carolyn Rose, and Eduard Hovy. Exploring Numeracy in Word Embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3374–3380, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1329. URL https://aclanthology.org/P19-1329.

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.

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.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. Numnet: Machine reading comprehension with numerical reasoning. 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)*, pp. 2474–2484, 2019.

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.

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

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.

Georgios Spithourakis and Sebastian Riedel. Numeracy for Language Models: Evaluating and Improving their Ability to Predict Numbers. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia, July 2018. Association for Computational Linguistics.

Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. Methods for Numeracy-Preserving Word Embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4742–4753, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.384. URL https://aclanthology.org/2020.emnlp-main.384.

Avijit Thawani, Jay Pujara, and Filip Ilievski. Numeracy enhances the Literacy of Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6960–6967, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.557. URL https://aclanthology.org/2021.emnlp-main.557.

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)*, pp. 5307–5315, 2019.

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

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *ArXiv*, abs/2206.07682, 2022a.

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*, 2022b.

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*, pp. 38–45, 2020.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang, et al. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*, 2022.

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)*, pp. 6016–6031, 2022.

Xikun Zhang, Deepak Ramachandran, Ian Tenney, Yanai Elazar, and Dan Roth. Do Language Embeddings capture Scales? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4889–4896, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.439. URL https://aclanthology.org/2020.findings-emnlp.439.

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

A Preliminary Study Details

Here we present the model performance on MathExp of GPT-3 with different solving algorithms in Figure 5 and Figure 6. We can conclude that: (1) both algorithms are not sensitive with either the 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.

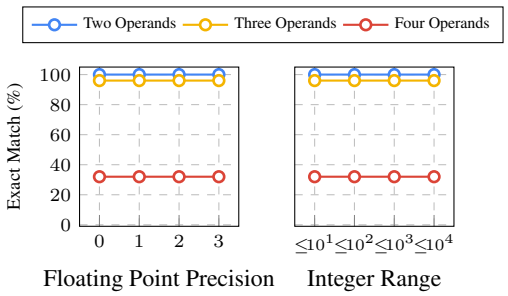

Figure 5: Performance over different floating point precision (left) and integer range (right) on MathExp of GPT-3 *w.* search-based algorithm.

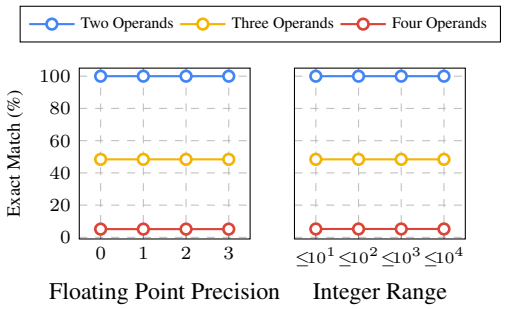

Figure 6: Performance over different floating point precision (left) and integer range (right) on MathExp of GPT-3 *w.* analytical-based algorithm.

B Operand Proposal Details

In Section 3.2, 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 p, then the substituted numbers can be represented as $x + k \cdot 10^{\mathrm{p}}$, where $k \in \{-5, -4, -3, -2, -1, 1, 2, 3, 4, 5\}$.

C Experiments

C.1 Experimental Setup

Table 6: Statistics of DROP dataset

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

Table 7: Statistics of MAWPS dataset

Subset	# Questions
AddSub	395
MultiArith	600

For BART, we implement the fine-tuning methods using the Huggingface transformers library (Wolf et al., 2020) on 4 V100 16GB GPUs. We use $\texttt{BART}_{\text{LARGE}}$(Lewis et al., 2020) 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 $\texttt{T5}_{\text{LARGE}}$ (Raffel et al., 2020) as our backbone.

Hyperparameter Selection For fine-tuning evaluation, we apply Adam (Loshchilov & Hutter, 2019) optimizer. The fine-tuning epochs are set as 50. For BART models (i.e., BART and PoET-SQL), we follow previous works (Pi et al., 2022) 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 zero-shot / few-shot evaluation, we employ the 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.

C.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 8.

Table 8: 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 (Pi et al., 2022)	66.2	69.2
BART*	67.4	70.6
T5 (Yoran et al., 2022)	–	64.6
T5*	61.0	64.6
PoET-SQL (Pi et al., 2022)	77.7	80.6
PoET-SQL*	76.9	80.0

C.3 Combination with Majority Voting (Wang et al., 2022)

Wang et al. (2022) proposed *self-consistency*, which samples different reasoning paths, and then pick the most consistent final answer by majority voting. Such majority voting method proves to improve LLMs' performance over various reasoning benchmarks. We have also conducted experiments combining majority voting and SoLiS, to check if there is further improvement. We present the performance comparison in Table 9.

Table 9: Combination with Majority Voting (Wang et al., 2022)

Language Model	Setting	MultiArith
GPT-3 (175B)	Zero-shot Chain (Kojima et al., 2022)	63.8
	w. majority voting	73.5 (+9.7)
	w. majority voting +SoLiS	83.5 (+19.7)

D MORE RESULTS ON DROP

We present the performance breakdown of F1 on dev set of DROP in Table 10 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 11 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 12, which again illustrates the unreliability of language models.

Table 10: 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 11: Few-shot evaluation 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 12: 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