PAL: Program-aided Language Models

Luyu Gao [*1] **Aman Madaan** [*1] **Shuyan Zhou** [*1] **Uri Alon** [1] **Pengfei Liu** [1 2]
Yiming Yang [1] **Jamie Callan** [1] **Graham Neubig** [1 2]
{luyug,amadaan,shuyanzh,ualon,pliu3,yiming,callan,gneubig}@cs.cmu.edu

Abstract

Large language models (LLMs) have demonstrated an impressive ability to perform arithmetic and symbolic reasoning tasks, when provided with a few examples at test time ("few-shot prompting"). Much of this success can be attributed to prompting methods such as "chain-of-thought", which employ LLMs for both *understanding* the problem description by decomposing it into steps, as well as *solving* each step of the problem. While LLMs seem to be adept at this sort of step-by-step decomposition, LLMs often make logical and arithmetic mistakes in the solution part, even when the problem is decomposed correctly. In this paper, we present Program-Aided Language models (PAL): a novel approach that uses the LLM to read natural language problems and generate *programs* as the intermediate reasoning steps, but offloads the *solution* step to a runtime such as a Python interpreter. With PAL, decomposing the natural language problem into runnable steps remains the only learning task for the LLM, while solving is delegated to the interpreter. We demonstrate this synergy between a neural LLM and a symbolic interpreter across 13 mathematical, symbolic, and algorithmic reasoning tasks from BIG-Bench Hard and others. In all these *natural language* reasoning tasks, generating code using an LLM and reasoning using a Python interpreter leads to more accurate results than much larger models. For example, PAL using CODEX achieves state-of-the-art few-shot accuracy on GSM8K, surpassing PaLM-540B which uses chain-of-thought by absolute 15% top-1. [1]

[*]Equal contribution [1]Language Technologies Institute, Carnegie Mellon University, USA [2]Inspired Cognition, USA. Correspondence to: Luyu Gao, Aman Madaan, Shuyan Zhou <{luyug,amadaan,shuyanzh}@cs.cmu.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

[1]Code and data at http://reasonwithpal.com.

1. Introduction

Until as recently as three years ago, reasoning was considered to be one of the most significant challenges that large language models (LLMs) had not yet overcome (Marcus, 2018; 2020; Garcez & Lamb, 2020). Recently, LLMs have shown impressive success on a wide range of reasoning tasks, including commonsense (Wei et al., 2021; Sanh et al., 2021; Madaan et al., 2022), mathematical (Lewkowycz et al., 2022; Wu et al., 2022; Mishra et al., 2022), and symbolic reasoning (Yao et al., 2022; Ahn et al., 2022), using few-shot prompting (Brown et al., 2020).

This process has been accelerated by methods that require LLMs to generate their explicit reasoning steps, such as "chain-of-thought" (Wei et al., 2022), "scratchpads" (Nye et al., 2021a), and "least-to-most" (Zhou et al., 2022) prompting. In particular, the widely used chain-of-thought (CoT) method presents the model with the explicit intermediate steps that are required to reach the final answer. Then, the model is expected to apply a similar decomposition to the actual test example, and consecutively reach an accurate final answer (Ling et al., 2017; Amini et al., 2019). Nevertheless, while LLMs can decompose natural language problems into steps and perform *simple* arithmetic operations, their performance falls dramatically when dealing with complex arithmetic (Hendrycks et al., 2021; Madaan & Yazdanbakhsh, 2022) or large numbers (Geva et al., 2020; Nogueira et al., 2021; Qian et al., 2022). In fact, even when fine-tuning a PaLM-based model on 164B tokens of explicit mathematical content, its two most common failures are reportedly "incorrect reasoning" and "incorrect calculation" (Lewkowycz et al., 2022).

In this paper, we propose Program-Aided Language model (PAL): a novel approach that uses an LLM to read natural language problems and generate *programs* as reasoning steps, but offloads the *solution* step to a Python interpreter, as illustrated in Figure 1. This offloading leverages an LLM that can decompose a natural language problem into programmatic steps, which is fortunately available using contemporary state-of-the-art LLMs that are pre-trained on both natural language and programming languages (Brown et al., 2020; Chen et al., 2021a; Chowdhery et al., 2022). While

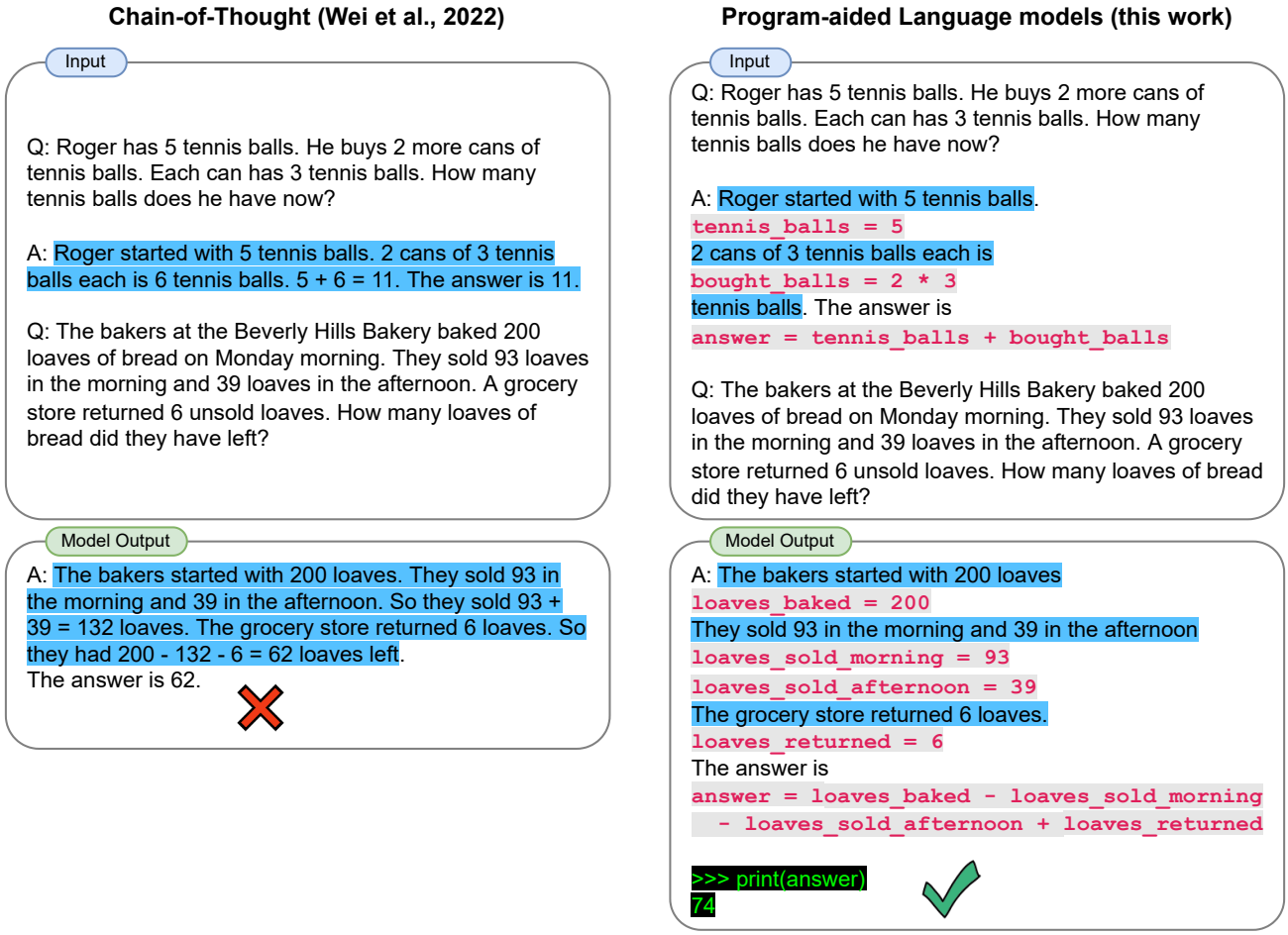


Figure 1: A diagram illustrating PAL: Given a mathematical reasoning question, Chain-of-thought (left) generates intermediate reasoning steps of free-form text. In contrast, Program-aided Language models (PAL, right) generate intermediate steps *and* Python code. This shifts the role of *running* the reasoning steps from the language model to the Python interpreter. The final answer is obtained by running the generated reasoning chain. Chain-of-thought reasoning is highlighted in blue; PAL steps are `highlighted in gray and pink`; the Python interpreter run is highlighted in black and green.

natural language understanding and decomposition *require* LLMs, solving and reasoning can be done with the external solver. This bridges an important gap in chain-of-thought-like methods, where reasoning chains can be correct but produce an incorrect answer.

We demonstrate the effectiveness of PAL across **13** arithmetic and symbolic reasoning tasks. In all these tasks, PAL using Codex (Chen et al., 2021a) outperforms much larger models such as PaLM-540B using chain-of-thought prompting. For example, on the popular GSM8K benchmark, PAL achieves state-of-the-art accuracy, surpassing PaLM-540B with chain-of-thought by absolute 15% top-1 accuracy. When the questions contain large numbers, a dataset we call GSM-HARD, PAL outperforms CoT by an absolute 40%. We believe that this seamless synergy between a neural LLM and a symbolic interpreter is an essential step towards general and robust AI reasoners.

2. Background: Few-shot Prompting

Few-shot prompting leverages the strength of large-language models to solve a task with a set of k examples that are provided as part of the test-time input (Brown et al., 2020; Liu et al., 2021; Chowdhery et al., 2022), where k is usually a number in the low single digits. These input-output examples $\{(x_i, y_i)\}_{i=1}^{k}$ are concatenated in a prompt $p \equiv \langle x_1 \cdot y_1 \rangle \,\|\, \langle x_2 \cdot y_2 \rangle \,\|\, \ldots \,\|\, \langle x_k \cdot y_k \rangle$. where "$\cdot$" denotes the concatenation of an input and output, and "$\|$" indicate the concatenation of different examples. During inference, a test instance x_{test} is appended to the prompt, and $p \,\|\, x_{test}$ is passed to the model which attempts to complete $p \,\|\, x_{test}$, and thereby generate an answer y_{test}. Note that such few-

shot prompting does not modify the underlying LLM.

Wei et al. (2022) additionally augment each in-context example with *chain of thought* (CoT) intermediate steps. Specifically, each in-context example in the CoT setup is a triplet $\langle x_i, t_i, y_i \rangle$, where x_i and y_i are input-output pair as before, and t_i is a natural language description of the steps that are needed to arrive at the output y_i from the input x_i. See Figure 1 for an example. With the additional "thoughts" t_i, the prompt is set to $p \equiv \langle x_1 \cdot t_1 \cdot y_1 \rangle \| \langle x_2 \cdot t_2 \cdot y_2 \rangle \| \ldots \| \langle x_k \cdot t_k \cdot y_k \rangle$.

During inference, the new question x_{test} is appended to the prompt as before and supplied to the LLM. Crucially, the model is tasked with generating *both* the thought t_{test} and the final answer y_{test}. This approach of prompting the model to first generate a reasoning process t_{test} improves the accuracy of the answer y_{test} across a wide range of tasks (Wang et al., 2022a; Wei et al., 2022; Zhou et al., 2022; Wang et al., 2022b).

3. Program-aided Language Models

In a Program-aided Language model, we propose to generate the thoughts t for a given natural language problem x as interleaved natural language (NL) and programming language (PL) statements. Since we delegate the solution step to an interpreter, we do not provide the final answers to the examples in our prompt. That is, every in-context example in PAL is a *pair* $\langle x_i, t_i \rangle$, where $t_j = [s_1, s_2, \ldots, s_N]$ with each $s_i \in \text{NL} \cup \text{PL}$, a sequence of tokens in either NL or PL. The complete prompt is thus $p \equiv \langle x_1 \cdot t_1 \rangle \| \langle x_2 \cdot t_2 \rangle \| \ldots \| \langle x_k \cdot t_k \rangle$.

Given a test instance x_{test}, we append it to the prompt, and $p \| x_{test}$ is fed to the LM. We let the LM generate a prediction t_{test}, which contains both the intermediate steps *and* their corresponding programmatic statements.

```
A: Roger started with 5 tennis balls.
tennis_balls = 5
2 cans of 3 tennis balls each is
bought_balls = 2 * 3
tennis balls. The answer is
answer = tennis_balls + bought_balls
```

Figure 2: A close-up of a single example from a PAL prompt. Chain-of-thought reasoning is highlighted in blue, and PAL programmatic steps are highlighted in gray and pink.

Example A close-up of the example from Figure 1 is shown in Figure 2. While chain-of-thought only decomposes the solution in the prompt into natural language steps such as Roger started with 5 tennis balls and 2 cans of 3 tennis balls each is 6, in PAL we also augment each such NL step with its corresponding programmatic statement such as `tennis_balls = 5` and `bought_balls = 2 * 3`. This way, the model learns to generate a *program* that will provide the answer for the test question, instead of relying on LLM to perform the calculation correctly.

We prompt the language model to generate NL intermediate steps using comment syntax (e.g. "`# ...`" in Python) such they will be ignored by the interpreter. We pass the generated program t_{test} to its corresponding solver, we run it, and obtain the final run result y_{test}. In this work we use a standard Python interpreter, but this can be any solver, interpreter or a compiler.

Crafting prompts for PAL In our experiments, we leveraged the prompts of existing work whenever available, and otherwise randomly selected the same number (3-6) of examples as previous work for creating a fixed prompt for every benchmark. In all cases, we augmented the free-form text prompts into PAL-styled prompts, leveraging programming constructs such as `for` loops and dictionaries when needed. Generally, writing PAL prompts is easy, and does not require more effort than writing the initial CoT prompts.

We also ensure that variable names in the prompt meaningfully reflect their roles. For example, a variable that describes the *number of apples in the basket* should have a name such as `num_apples_in_basket`. This keeps the generated code linked to the entities in the question. In Section 6, we show that such meaningful variable names are critical to the downstream performance. Notably, it is also possible to incrementally run the PL segments and feed the execution results back to the LLM to generate the following blocks. For simplicity, in our experiments, we used a single, post-hoc, execution.

This work focuses on CoT-style reasoning chain, but in Appendix J we show that PAL also improves Least-to-Most (Zhou et al., 2022) prompts, which introduce reasoning chains that decompose a question into sub-questions.

4. Experimental Setup

Data and in-context examples We experiment with three broad classes of reasoning tasks: (1) mathematical problems (§4.1) from a wide range of datasets including GSM8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), ASDIV (Miao et al., 2020), and MAWPS (Koncel-Kedziorski et al., 2016); (2) symbolic reasoning (§4.2) from BIG-Bench Hard (Suzgun et al., 2022); (3) and algorithmic problems (§4.3) from BIG-Bench Hard as well. Details of all datasets are shown in Appendix I. For all of the experiments for

Q: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?

```
money_initial = 23
bagels = 5
bagel_cost = 3
money_spent = bagels * bagel_cost
money_left = money_initial - money_spent
answer = money_left
```

Figure 3: Example prompt for the mathematical reasoning tasks, from the GSM8K benchmark.

Q: On the table, you see a bunch of objects arranged in a row: a purple paperclip, a pink stress ball, a brown keychain, a green scrunchiephone charger, a mauve fidget spinner, and a burgundy pen. What is the color of the object directly to the right of the stress ball?

```
...
stress_ball_idx = None
for i, object in enumerate(objects):
    if object[0] == 'stress ball':
        stress_ball_idx = i
        break
# Find the directly right object
direct_right = objects[stress_ball_idx+1]
# Check the directly right object's color
answer = direct_right[1]
```

Figure 4: An example for a PAL prompt in the COLORED OBJECTS task. For space considerations, we omit the code that creates the list `objects`.

which COT prompts were available, we used the same in-context examples as used by previous work. Otherwise, we randomly sampled a fixed set of in-context examples, and used the same set for PAL and COT.

Baselines We consider three prompting strategies: DIRECT prompting – the standard prompting approach using pairs of questions and immediate answers (e.g., Answer: 11) as in Brown et al. (2020); chain-of-thought (COT) prompting (Wei et al., 2022); and our PAL prompting. We performed greedy decoding from the language model using a temperature of 0. Unless stated otherwise, we used CODEX (`code-davinci-002`) as our backend LLM for both PAL, DIRECT, and COT. In datasets where results for additional base LMs, such as PaLM-540B, were available from previous work, we included them as COT $_{\text{PaLM-540B}}$.

4.1. Mathematical Reasoning

We evaluate PAL on eight mathematical word problem datasets. Each question in these tasks is an algebra word problem at grade-school level. An example for a question and PAL example prompt is shown in Figure 3. We found that using explicit NL intermediate steps does not further benefit these math reasoning tasks, hence we kept only the meaningful variable names in the prompt.

4.2. Symbolic Reasoning

We applied PAL to three symbolic reasoning tasks from BIG-Bench Hard (Suzgun et al., 2022), which involve reasoning about objects and concepts: (1) COLORED OBJECTS requires answering questions about colored objects on a surface. This task requires keeping track of relative positions, absolute positions, and the color of each object. Figure 4 shows an example for a question and example PAL prompt. (2) PENGUINS describes a table of penguins and some additional information in natural language, and the task is to answer a question about the attributes of the penguins, for example, "*how many penguins are less than 8 years old?*". While both PENGUINS and COLORED OBJECT tasks require tracking objects, PENGUINS describes *dynamics* as well, since the penguins in the problem can be added or removed. Figure 17 in Appendix K.2 shows an example for a question, a chain-of-thought prompt, and PAL prompt. (3) DATE is a date understanding task that involves inferring dates from natural language descriptions, performing addition and subtraction of relative periods of time, and having some global knowledge such as "how many days are there in February", and performing the computation accordingly. Appendix K.3 shows example prompts.

Q: I have a chair, two potatoes, a cauliflower, a lettuce head, two tables, a cabbage, two onions, and three fridges. How many vegetables do I have?

```
# note: I'm not counting the chair, tables,
    or fridges
vegetables_to_count = {
    'potato': 2,
    'cauliflower': 1,
    'lettuce head': 1,
    'cabbage': 1,
    'onion': 2
}
answer = sum(vegetables_to_count.values())
```

Figure 5: An example for a PAL prompt in the OBJECT COUNTING task. The base LM is expected to convert the input into a dictionary where keys are entities and values are their quantities, while filtering out non-vegetable entities. Finally, the answer is the sum of the dictionary values.

	GSM8K	GSM-HARD	SVAMP	ASDIV	SINGLEEQ	SINGLEOP	ADDSUB	MULTIARITH
DIRECT $_{\text{Codex}}$	19.7	5.0	69.9	74.0	86.8	93.1	90.9	44.0
COT $_{\text{UL2-20B}}$	4.1	-	12.6	16.9	-	-	18.2	10.7
COT $_{\text{LaMDA-137B}}$	17.1	-	39.9	49.0	-	-	52.9	51.8
COT $_{\text{Codex}}$	65.6	23.1	74.8	76.9	89.1	91.9	86.0	95.9
COT $_{\text{PaLM-540B}}$	56.9	-	79.0	73.9	92.3	94.1	91.9	94.7
COT $_{\text{Minerva 540B}}$	58.8	-	-	-	-	-	-	-
PAL	**72.0**	**61.2**	**79.4**	**79.6**	**96.1**	**94.6**	**92.5**	**99.2**

Table 1: Problem solve rate (%) on mathematical reasoning datasets. The highest number on each task is in **bold**. The results for DIRECT and PaLM-540B are from Wei et al. (2022), the results for LaMDA and UL2 are from Wang et al. (2022b), and the results for Minerva are from Lewkowycz et al. (2022). We ran PAL on each benchmark 3 times and report the average; the standard deviation is provided in Table 9.

4.3. Algorithmic Tasks

Finally, we compare PAL and COT on algorithmic reasoning. These are tasks where a human programmer can write a deterministic program with prior knowledge of the question. We experiment with two algorithmic tasks: OBJECT COUNTING and REPEAT COPY. OBJECT COUNTING involves answering questions about the number of objects belonging to a certain type. For example, as shown in Figure 5: *"I have a chair, **two potatoes**, **a cauliflower**, **a lettuce head**, two tables, ... How many vegetables do I have?"*). REPEAT COPY requires generating a sequence of words according to instructions. For example, as shown in Appendix K.6: *"Repeat the word duck four times, but halfway through also say quack"*).

5. Results

5.1. Math Results

Table 1 shows the following results: across all tasks, PAL using Codex sets a new few-shot state-of-the-art top-1 decoding across all datasets, outperforming $\text{COT}_{\text{Codex}}$, $\text{COT}_{\text{PaLM-540B}}$, and $\text{COT}_{\text{Minerva 540B}}$ which was fine-tuned on explicit mathematical content.

Interestingly, COT also benefits from Codex over PaLM-540B in some of the datasets such as ASDIV, but performs worse than PaLM-540B in others such as SVAMP. Yet, using PAL further improves the solve rate across all datasets.

GSM-HARD LLMs can perform simple calculations with *small* numbers. However, Madaan & Yazdanbakhsh (2022) found that 50% of the numbers in the popular GSM8K dataset of math reasoning problems are *integers between 0 and 8*. This raises the question of whether LLMs can generalize to larger and non-integer numbers? We constructed a harder version of GSM8K, which we call GSM-HARD, by replacing the numbers in the questions of GSM8K with larger numbers. Specifically, one of the numbers in a question was replaced with a random integer of up to 7 digits. More details regarding the this new dataset are provided in I.1. On GSM-HARD (Table 1), the accuracy of DIRECT drops dramatically from 19.7% to 5.0% (a relative drop of 74%), the accuracy of COT drops from 65.6% to 20.1% (a relative drop of almost 70%), while PAL remains stable at 61.5%,

	COLORED OBJECT	PENGUINS	DATE	REPEAT COPY	OBJECT COUNTING
DIRECT Codex	75.7	71.1	49.9	81.3	37.6
COT LaMDA-137B	-	-	26.8	-	-
COT PaLM-540B	-	65.1	65.3	-	-
COT Codex	86.3	79.2	64.8	68.8	73.0
PAL Codex	**95.1**	**93.3**	**76.2**	**90.6**	**96.7**

Table 2: Solve rate on three symbolic reasoning datasets and two algorithmic datasets, In all datasets, PAL achieves a much higher accuracy than chain-of-thought. Results with closed models LaMDA-137B and PaLM-540B are included if available to public (Wei et al., 2022; Suzgun et al., 2022).

dropping by only 14.3%. The results of COT on GSM-HARD did not improve even when we replaced *its prompts* with prompts that include large numbers (Appendix B).

This shows how PAL provides not only better results on the standard benchmarks, but is also much more *robust*. In fact, since PAL offloads the computation to the Python interpreter, any complex computation can be performed accurately given the correctly generated program.

Large Numbers or Incorrect Reasoning? Are the failures on GSM-HARD primarily due to the inability of LLMs to do arithmetic, or do the large numbers in the question "confuse" the LM which generates irrational intermediate steps? To investigate this, we evaluated the outputs generated by COT for the two versions of the same question (with and without large numbers). We find that in 16 out of 25 cases we analyzed, COT generates nearly identical natural language "thoughts", indicating that the primary failure mode is the inability to perform arithmetic accurately. Sample outputs are provided in the Appendix, Table 13.

	GSM8K `majority@40`
COT UL2-20B	7.3
COT LaMDA-137B	27.7
COT Codex	78.0
COT PaLM-540B	74.4
COT Minerva 540B	78.5
PAL Codex	**80.4**

Table 3: Problem solve rate (%) on GSM8K using `majority@40` (Wang et al., 2022b)

Multi-sample Generation As found by Wang et al. (2022b), chain-of-thought-style methods can be further improved by sampling $k > 1$ outputs, and selecting the final answer using majority voting. We thus repeated the greedy-decoding experiments using nucleus sampling (Holtzman et al., 2019) with $p = 0.95$ and $k = 40$ as in Lewkowycz et al. (2022) and temperature of 0.7. As shown in Table 3, this further increases the accuracy of PAL from 72.0% to 80.4% on GSM8K, obtaining 1.9% higher accuracy than Minerva-540B using the same number of samples.

5.2. Symbolic Reasoning & Algorithmic Tasks Results

Results for symbolic reasoning and algorithmic tasks are shown in Table 2. In COLORED OBJECTS, PAL improves over the strong COT by 8.8%, and by 19.4% over the standard direct prompting. In PENGUINS, PAL provides a gain of absolute 14.1% over COT. In DATE, PAL further provides 11.4% gain over both COT Codex, PaLM-540B, and LaMDA-137B.

The two rightmost columns of Table 2 show that PAL is close to solving OBJECT COUNTING, reaching 96.7% and improving over COT by absolute 23.7%. Similarly, PAL vastly outperforms COT by absolute 21.8% on REPEAT COPY. Surprisingly, DIRECT prompting performs better than COT on REPEAT COPY. Yet, PAL improves over DIRECT by 9.3% in REPEAT COPY.

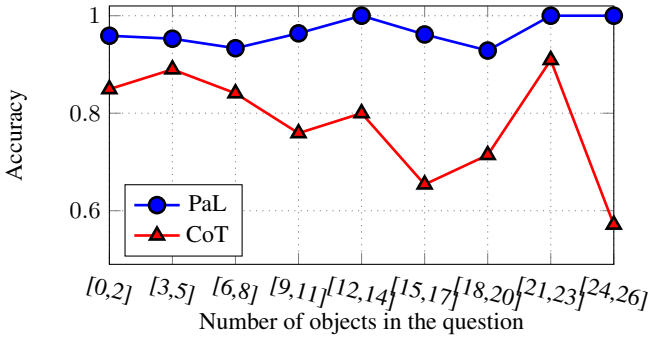


Figure 6: The solve rate on COLORED OBJECTS with respect to the number of objects included in the test question.

Is PAL sensitive to the complexity of the question? We examined how the performance of PAL and COT changes as the complexity of the input question grows, measured as the number of objects in the question of COLORED OBJECTS. As shown in Figure 6, PAL outperforms COT across all input lengths. As the number of objects in the question increases, COT's accuracy is unstable and drops, while PAL

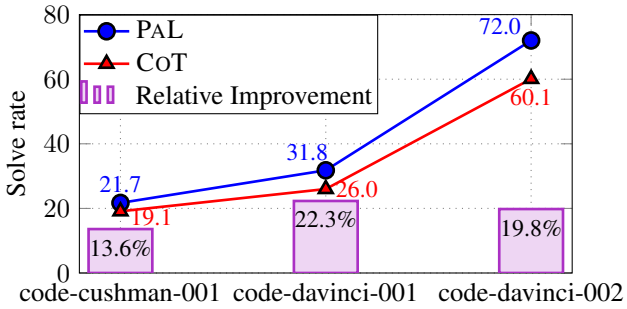


Figure 7: PAL with different models on GSM8K: though the absolute accuracies with `code-cushman-001` and `code-davinci-001` are lower than `code-davinci-002`, the relative improvement of PAL over CoT is consistent across models.

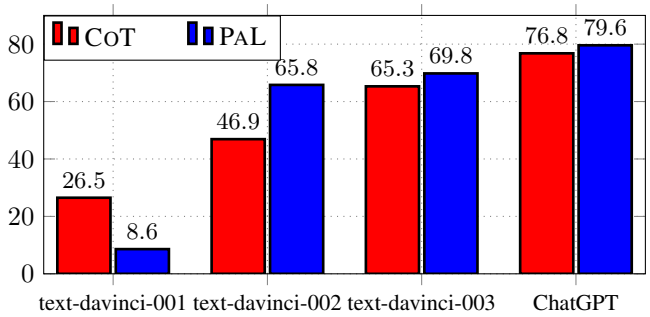


Figure 8: PAL with NL LMs on GSM8K: though CoT outperforms PAL with `text-davinci-001`, once the base LM is sufficiently strong, PAL is beneficial with `text-davinci-002` and `text-davinci-003` as well. That is, PAL is not limited to code-LMs only.

remains consistently close to 100%. More analysis on the token-level predictions can be found in Appendix H.

6. Analysis

Does PAL work with weaker LMs? In all our experiments in Section 5, PAL used the `code-davinci-002` model; but can PAL work with weaker models of code? We compared PAL with CoT when both prompting approaches use the same weaker base LMs `code-cushman-001` and `code-davinci-001`. As shown in Figure 7, even though the absolute accuracies of `code-cushman-001` and `code-davinci-001` are lower, the relative improvement of PAL over CoT remains consistent across models. This shows that PAL can work with weaker models, while its benefit scales elegantly to stronger models as well.

Is PAL limited to Code-LMs? We also experimented with PAL using the *text-davinci* series. Figure 8 shows the following interesting results: when the base LM's "code modeling ability" is weak (using `text-davinci-001`), CoT performs better than PAL. However, once the LM's code modeling ability is sufficiently high (using `text-davinci-002` and `text-davinci-003`), PAL outperforms CoT, and PAL `text-davinci-003` performs almost as PAL `code-davinci-002`. This shows that PAL is not limited to LMs of code, but it can work with LMs that were mainly trained for natural language, if they have a sufficiently high coding ability.

The base *ChatGPT* (`gpt-3.5-turbo`) appears to be stronger than the base `text-davinci-003` on GSM8K. However as shown in Figure 8, PAL *further improves* even the strong ChatGPT model by 2.8% absolute.

Is PAL better because of the Python prompt or because of the interpreter? We experimented with generating Python code, while requiring the neural LM to "execute" it as well, without using an interpreter, following Nye et al. (2021a); Madaan et al. (2022). We created prompts that are similar to PAL's, except that they *do include* the final answer. This resulted in a 23.2 solve rate on GSM8K, much lower than PAL (72.0), and only 4.5 points higher than DIRECT. These results reinforce our hypothesis that the main benefit of PAL comes from the synergy with the interpreter, and not only from having a better prompt. Additional details are provided in Appendix B. For additional discussion on code-prompts compared to textual-prompts, see Appendix H.

Do variable names matter? In all our experiments, we used meaningful variable names in the PAL prompts, to ease the model's grounding of variables to the entities they represent. For the Python interpreter, however, variable names are meaningless. To measure the importance of meaningful variable names, we experimented with two prompts variants:

1. $\text{PAL}_{-\text{comment}}$ – the PAL prompt without intermediate NL comments.
2. $\text{PAL}_{-\text{comment}}^{-\text{var}}$ – the PAL prompt without intermediate NL comments *and* with variable names substituted with random characters.

The results are shown in Figure 9. In COLORED OBJECTED and DATE, removing intermediate NL comments but keeping meaningful variable names ($\text{PAL}_{-\text{comment}}$) – slightly reduces the accuracy compared to the full PAL prompt, but it still achieves higher accuracy than the baselines CoT. Removing variable names as well ($\text{PAL}_{-\text{comment}}^{-\text{var}}$) further decreases accuracy, and performs worse than CoT. Since variable names have an important part in code quality (Gellenbeck & Cook, 1991; Takang et al., 1996), meaningful variable names are only expected to ease reasoning for

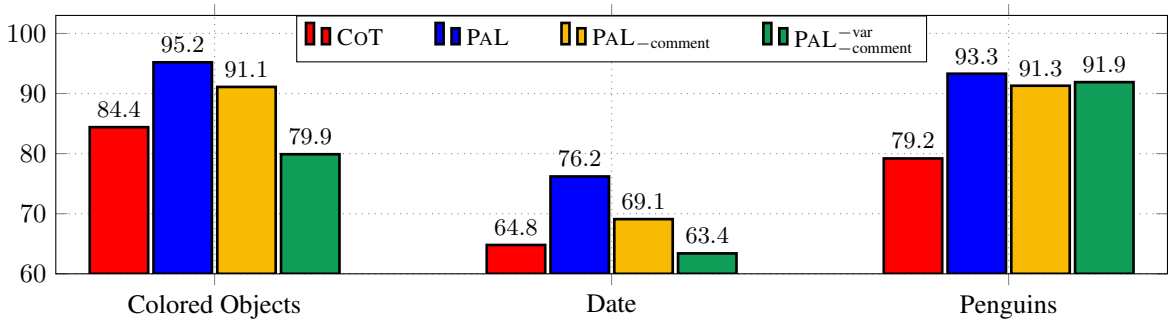


Figure 9: Ablation study of PAL prompt formats. We consider the original PAL prompt, it with natural language comments removed (PAL$_{-\text{comment}}$), and further variable names replaced with random character (PAL$^{-\text{var}}_{-\text{comment}}$). As a reference, we also show the COT performance (blue).

Codex, which was trained on mostly meaningful names, as was also found by Madaan et al. (2022).

Is the generated code correct? *All* generated Python code was syntactically correct. Less than 1% of the examples raised an exception at runtime; among these examples, the most common error was trying to use a variable that was not defined before (`NameError`).

7. Related Work

Prompting Few-shot prompting (Brown et al., 2020) has been shown to be an effective approach for a variety of tasks (Liu et al., 2021) ranging from text- (Gehrmann et al., 2021; Reif et al., 2021; Wei et al., 2021; Sanh et al., 2021) to code-generation (Chen et al., 2021b). Methods such as chain-of-thought prompting (COT) have further unlocked a variety of reasoning tasks, boosting the performance of models on a variety of benchmarks. Nevertheless, all previous approaches suffer from inaccuracy in arithmetic calculation and incorrect reasoning (Lewkowycz et al., 2022; Hendrycks et al., 2021; Madaan & Yazdanbakhsh, 2022). PAL avoids these problems by offloading the calculation and some of the reasoning to a Python interpreter, which is correct by construction, given the right program. Further, not only that PAL can improve the standard chain-of-thought, it can improve least-to-most prompting (Zhou et al., 2022) as well, as we show in Appendix J.

LMs with external tools Several prior works have equipped neural models with specialized modules to create effective cascades (Dohan et al., 2022). For example, Cobbe et al. (2021) employ a calculator for arithmetic operations as a post hoc processing, and Demeter & Downey (2020) add specialized modules for generating cities and dates. Unlike these works, PAL generates code for a Python interpreter, which is general enough to handle both arithmetic calculations and dates, without specialized modules and ad-hoc fixes. Chowdhery et al. (2022) and Wei et al. (2022) have also experimented with external calculators; however, the calculator had improved Codex by only 2.3% (absolute) on GSM8K and improved PaLM-540B by 1.7%, while PAL improves Codex by 6.4% on the same benchmark (Section 5.1). Similarly to our work, Chowdhery et al. (2022) have also experimented with generating Python code for solving the GSM8K benchmark, but their experiments resulted in *lower* accuracy than the standard PaLM-540B that uses chain-of-thought. Pi et al. (2022) pretrain the model on execution results of random expressions on a calculator, instead of using the solver at test time as well. While their model can hypothetically perform arithmetic better than other pretrained LMs, their results on the SVAMP benchmark are much lower: 57.4% using a T5-11B model, while PAL achieves 79.4% on the same benchmark without any specialized pretraining.

Shortly after a preprint of our work was submitted to arXiv, another related work on "program of thought prompting" (Chen et al., 2022) was also submitted to arXiv. Their method is conceptually similar to ours, but Chen et al. (2022) (1) only demonstrates efficacy on mathematical problems, whereas we demonstrate gains on symbolic and algorithmic benchmarks as well, and (2) chose benchmark-specific prompt examples, while we used the same prompt examples as previous work, to disentangled the benefit of our approach from the benefit of the choice of examples.

Semantic parsing Our work can also be seen as a very general form of semantic parsing, where instead of parsing into strict domain-specific languages, the model generates free-form Python code. Some works constrain the decoder using a Context-Free Grammar (CFG) to generate a domain-specific meaning representation (Shin & Van Durme, 2021) or a canonical utterance, which can be converted to a Lisp-like meaning representation (Shin et al., 2021). Nye et al. (2021b) first generate candidate sentences using a language model; then, another language model (GPT-3) is used to

derive logical constraints entailed by each candidate; these constraints are then cross-checked with a predefined list of facts to rank the candidates. In contrast, PAL does not require any constraining or domain-specific representations other than Python code. Further, LMs that were pretrained on Python are abundant compared to other domain-specific languages, making Python code a much more preferable representation. Andor et al. (2019) generate task-specific arithmetic operations for reading comprehension tasks; Gupta et al. (2019) design neural modules such as `count` to deal with arithmetic operations. PAL generalizes these works by generating general Python programs, without the need for defining specialized modules. The closest work to ours technically may be Binder (Cheng et al., 2022), but it addressed mostly answering questions about tables using SQL and SQL-like Python.

8. Conclusion

We introduce PAL, a new approach for natural language reasoning, using *programs* as intermediate reasoning steps. Differently from existing LM-based reasoning approaches, the main idea is to offload solving and calculating to an external Python interpreter, instead of using the LLM for both understanding the problem *and* solving. This results in a final answer that is guaranteed to be accurate, given the correctly predicted programmatic steps. We demonstrate this seamless synergy between an LLM and a Python interpreter across 13 tasks from BIG-Bench Hard and other benchmarks. In all these benchmarks, PAL outperforms larger LLMs such as PaLM-540B which use the popular "chain-of-thought" method and sets new state-of-the-art accuracy on all of them. We believe that these results unlock exciting directions for future neuro-symbolic AI reasoners. To this end, we make all our code, data and prompts available at `http://reasonwithpal.com`.

9. Acknowledgement

We thanks the anonymous reviewers for their useful comments and suggestions. This work is supported by a gift from Amazon AI and a contract from the DARPA KAIROS program under agreement number FA8750-19-2-0200. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, Amazon or the U.S. Government.

References

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R. J., Jeffrey, K., Jesmonth, S., Joshi, N. J., Julian, R., Kalashnikov, D., Kuang, Y., Lee, K.-H., Levine, S., Lu, Y., Luu, L., Parada, C., Pastor, P., Quiambao, J., Rao, K., Rettinghouse, J., Reyes, D., Sermanet, P., Sievers, N., Tan, C., Toshev, A., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Xu, S., Yan, M., and Zeng, A. Do as I Can, not as I Say: Grounding Language in Robotic Affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Amini, A., Gabriel, S., Lin, S., Koncel-Kedziorski, R., Choi, Y., and Hajishirzi, H. MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms. In *ACL*, 2019.

Andor, D., He, L., Lee, K., and Pitler, E. Giving bert a calculator: Finding operations and arguments with reading comprehension. *arXiv preprint arXiv:1909.00109*, 2019.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners. In *NeurIPS*, 2020.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021a.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021b.

Chen, W., Ma, X., Wang, X., and Cohen, W. W. Program of thoughts prompting: Disentangling computation from

reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Cheng, Z., Xie, T., Shi, P., Li, C., Nadkarni, R., Hu, Y., Xiong, C., Radev, D., Ostendorf, M., Zettlemoyer, L., Smith, N. A., and Yu, T. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*, 2022.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. PaLM: Scaling Language Modeling with Pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Cobbe, K., Kosaraju, V., Bavarian, M., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

Demeter, D. and Downey, D. Just add functions: A neural-symbolic language model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7634–7642, 2020.

Dohan, D., Xu, W., Lewkowycz, A., Austin, J., Bieber, D., Lopes, R. G., Wu, Y., Michalewski, H., Saurous, R. A., Sohl-Dickstein, J., et al. Language model cascades. *arXiv preprint arXiv:2207.10342*, 2022.

Garcez, A. d. and Lamb, L. C. Neurosymbolic ai: the 3rd wave. *arXiv preprint arXiv:2012.05876*, 2020.

Gehrmann, S., Adewumi, T., Aggarwal, K., Ammanamanchi, P. S., Anuoluwapo, A., Bosselut, A., Chandu, K. R., Clinciu, M., Das, D., Dhole, K. D., Du, W., Durmus, E., Dušek, O., Emezue, C., Gangal, V., Garbacea, C., Hashimoto, T., Hou, Y., Jernite, Y., Jhamtani, H., Ji, Y., Jolly, S., Kale, M., Kumar, D., Ladhak, F., Madaan, A., Maddela, M., Mahajan, K., Mahamood, S., Majumder, B. P., Martins, P. H., McMillan-Major, A., Mille, S., van Miltenburg, E., Nadeem, M., Narayan, S., Nikolaev, V., Niyongabo, R. A., Osei, S., Parikh, A., Perez-Beltrachini, L., Rao, N. R., Raunak, V., Rodriguez, J. D., Santhanam, S., Sedoc, J., Sellam, T., Shaikh, S., Shimorina, A., Cabezudo, M. A. S., Strobelt, H., Subramani, N., Xu, W., Yang, D., Yerukola, A., and Zhou, J. The GEM Benchmark: Natural Language Generation, its Evaluation and Metrics. *arXiv preprint arXiv:2102.01672*, 2021.

Gellenbeck, E. M. and Cook, C. R. An investigation of procedure and variable names as beacons during program comprehension. In *Empirical studies of programmers: Fourth workshop*, pp. 65–81. Ablex Publishing, Norwood, NJ, 1991.

Geva, M., Gupta, A., and Berant, J. Injecting numerical reasoning skills into language models. In *ACL*, 2020.

Gupta, N., Lin, K., Roth, D., Singh, S., and Gardner, M. Neural module networks for reasoning over text. *arXiv preprint arXiv:1912.04971*, 2019.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The Curious Case of Neural Text Degeneration. In *ICLR*, 2019.

Koncel-Kedziorski, R., Roy, S., Amini, A., Kushman, N., and Hajishirzi, H. Mawps: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1152–1157, 2016.

Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., Wu, Y., Neyshabur, B., Gur-Ari, G., and Misra, V. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022.

Ling, W., Yogatama, D., Dyer, C., and Blunsom, P. Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems. *arXiv preprint arXiv:1705.04146*, 2017.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *arXiv preprint arXiv:2107.13586*, 2021.

Madaan, A. and Yazdanbakhsh, A. Text and patterns: For effective chain of thought, it takes two to tango. *arXiv preprint arXiv:2209.07686*, 2022.

Madaan, A., Zhou, S., Alon, U., Yang, Y., and Neubig, G. Language models of code are few-shot commonsense learners. *arXiv preprint arXiv:2210.07128*, 2022.

Marcus, G. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

Marcus, G. The next decade in ai: four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177*, 2020.

Miao, S.-y., Liang, C.-C., and Su, K.-Y. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 975–984, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.92. URL https://aclanthology.org/2020.acl-main.92.

Mishra, S., Finlayson, M., Lu, P., Tang, L., Welleck, S., Baral, C., Rajpurohit, T., Tafjord, O., Sabharwal, A., Clark, P., and Kalyan, A. Lila: A unified benchmark for mathematical reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022.

Nogueira, R., Jiang, Z., and Lin, J. Investigating the limitations of transformers with simple arithmetic tasks. *arXiv preprint arXiv:2102.13019*, 2021.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., Sutton, C., and Odena, A. Show your Work: Scratchpads for Intermediate Computation with Language Models. *arXiv preprint arXiv:2112.00114*, 2021a.

Nye, M., Tessler, M., Tenenbaum, J., and Lake, B. M. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. *Advances in Neural Information Processing Systems*, 34: 25192–25204, 2021b.

Patel, A., Bhattamishra, S., and Goyal, N. Are NLP Models Really Able to Solve Simple Math Word Problems? *arXiv preprint arXiv:2103.07191*, 2021.

Pi, X., Liu, Q., Chen, B., Ziyadi, M., Lin, Z., Gao, Y., Fu, Q., Lou, J.-G., and Chen, W. Reasoning like program executors. *arXiv preprint arXiv:2201.11473*, 2022.

Qian, J., Wang, H., Li, Z., Li, S., and Yan, X. Limitations of language models in arithmetic and symbolic induction. *arXiv preprint arXiv:2208.05051*, 2022.

Reif, E., Ippolito, D., Yuan, A., Coenen, A., Callison-Burch, C., and Wei, J. A Recipe for Arbitrary Text Style Transfer with Large Language Models. *arXiv preprint arXiv:2109.03910*, 2021.

Sanh, V., Webson, A., Raffel, C., Bach, S. H., Sutawika, L., Alyafeai, Z., Chaffin, A., Stiegler, A., Scao, T. L., Raja, A., Dey, M., Bari, M. S., Xu, C., Thakker, U., Sharma, S. S., Szczechla, E., Kim, T., Chhablani, G., Nayak, N., Datta, D., Chang, J., Jiang, M. T.-J., Wang, H., Manica, M., Shen, S., Yong, Z. X., Pandey, H., Bawden, R., Wang, T., Neeraj, T., Rozen, J., Sharma, A., Santilli, A., Fevry, T., Fries, J. A., Teehan, R., Biderman, S., Gao, L., Bers, T., Wolf, T., and Rush, A. M. Multitask Prompted Training Enables Zero-Shot Task Generalization, 2021.

Shin, R. and Van Durme, B. Few-shot semantic parsing with language models trained on code. *arXiv preprint arXiv:2112.08696*, 2021.

Shin, R., Lin, C. H., Thomson, S., Chen, C., Roy, S., Platanios, E. A., Pauls, A., Klein, D., Eisner, J., and Van Durme, B. Constrained language models yield few-shot semantic parsers. *arXiv preprint arXiv:2104.08768*, 2021.

Suzgun, M., Scales, N., Scharli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E., Zhou, D., and Wei, J. Challenging big-bench tasks and whether chain-of-thought can solve them. *ArXiv*, abs/2210.09261, 2022.

Takang, A. A., Grubb, P. A., and Macredie, R. D. The effects of comments and identifier names on program comprehensibility: an experimental investigation. *J. Prog. Lang.*, 4(3):143–167, 1996.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., and Zhou, D. Rationale-Augmented Ensembles in Language Models. *arXiv preprints arXiv:2207.00747*, 2022a.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., and Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv preprint arXiv:2203.11171*, 2022b.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned Language Models are Zero-shot Learners. *arXiv preprint arXiv:2109.01652*, 2021.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint arXiv:2201.11903*, 2022.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with Large Language Models. *arXiv preprint arXiv:2205.12615*, 2022.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Bousquet, O., Le, Q., and Chi, E. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. *arXiv preprint arXiv:2205.10625*, 2022.

Part I

Appendix

Table of Contents

A. Alternative Prompts without Meaningful Variable Names

```
a = 23
b = 5
c = 3
d = b * c
e = a - d
print(e)
```

(a) Structured explanation with uninformative variable names (PAL - `var`)

```
# Olivia has $23
a = 23
# number of bagels bought
b = 5
# price of each bagel
c = 3
# total price of bagels
d = b * c
# money left
e = a - d
print(e)
```

(b) Structured explanation with uninformative variable names, but useful comments (PAL - `var` + `comms`)

```
money_initial = 23
bagels = 5
bagel_cost = 3
money_spent = bagels * bagel_cost
money_left = money_initial - money_spent
result = money_left
print(result)
```

(c) PAL prompts

Figure 10: **Role of text in PAL**: three different reasoning steps for the question *Olivia has $23. She bought five bagels for $3 each. How much money does she have left?* Uninformative variable names (left), Uninformative variable names with useful comments (left), and PAL. Including text description

Setting	CoT	PAL - `var`	PAL - `var` + `comms`	PAL
Solve Rate	63.1	59.0	69.0	71.8

Table 4: Role of text: including text either as informative variable names (PAL) or comments is important (PAL - `var` + `comms`). Uninformative variable names PAL - `var` cause a drastic drop in performance, indicating that just structure is not sufficient. The corresponding prompts are shown in Figure 10.

For mathematical problems, since our standard prompts do not use much comment, we start by creating alternative prompts where the informative variable names are replaced with single-letters (Figure 10). The results in Table 4 shows a considerable performance drop: from an average of 71.8% to 59%. Note that the ablation where structured outputs are completely removed in favor of purely text explanations is precisely the CoT setting, which achieves a solve rate of 63%. These results underscore the importance of text but more importantly show that combining both text and procedural statements leads to higher performance gains—either is sub-optimal.

B. Additional analysis on Arithmetic Reasoning

GSM-hard with hard prompts The GSM-HARD experiments used prompts that were sampled from the GSM8K training set. Will CoT be helped by using larger numbers in the prompts as well? To investigate this, we create prompts where the

numbers are changed to larger numbers, matching the distribution of numbers in GSM-HARD. The results in Table 5 shows that even with a prompt that matches the numbers, there are only modest gains in performance. These results show that the gains achieved by using code-based reasoning chains may not be achieved simply by using better few-shot examples for CoT.

	Regular Prompt	Prompt with Larger Numbers
CoT	23.3	23.8

Table 5: GSM-hard results, when the prompts also had examples of larger numbers.

Succinct code The programs used in few-shot examples by PaL are multi-step, and show a step-by-step breakdown of the reasoning process. Is this breakdown necessary? Alternatively, can we return a single line expression (see Figure 11b) to calculate the result? Results in Table 6 (4^{th} row) shows that is not the case. With single-line expressions, the performance of PaL falls to the level of direct prompting.

Generating the answer directly PaL first generates a reasoning chain in the form of a Python program, and passes the generated program to a runtime to obtain an answer. Is PaL better *only* because of the program-style intermediate reasoning chains, or are the improvements derived from offloading execution to the Python runtime? To investigate this, we experiment with a variant that forces the LLM to generate the answer after generating the reasoning chain (Figure 11e). This setting compels the LLM to condition on the generated code-based reasoning to generate an answer, simulating the runtime. The results in Table 6 (5^{th} row) show that the solve rate drops to near Direct levels. This reinforces our hypothesis that while current LLMs can be excellent at specifying a high-level plan to solve a task—they are still incapable of executing them.

Ablation	**Solve Rate**
Direct (no intermediate reasoning)	19.7
CoT	65.6
PaL	72.0
Succinct Code	47.8
LLM Simulating Runtime	23.2

Table 6: Solve rates for ablations

C. Effect of Using Language Models of Code

In our experiments, we focused on evaluating the performance of a language model for code. We aimed to investigate whether the additional performance boost observed in our results was due to the use of models like Codex, or whether our formulation was useful even for text-based models. To this end, we conducted additional experiments using text-based language models. Our findings indicate that the PAL approach is not restricted to working solely with Codex, but can also be applied to natural language (NL) models, as long as the model is sufficiently strong. Specifically, our results showed that in the text-davinci-001 model, the use of the CoT approach resulted in better performance.

Model	CoT	PaL
text-davinci-001	26.5	8.6
text-davinci-002	46.9	65.8
text-davinci-003	65.3	69.8

Table 7: Performance on GSM8K with different language models of text

D. Experiments with ChatGPT

We compare CoT and PaL on GSM8K with ChatGPT (`gpt-turbo-3.5`), a dialogue-tuned language model. We follow the official guideline[2] to present few-shot examples as the `example_user` and the `example_assistant`. The results are shown in Table 8. We find that PAL achieves stronger performance of 79.6, outperforming CoT by 2.8%.

Model	CoT	PaL
ChatGPT	76.8	79.6

Table 8: Performance on GSM8K with ChatGPT

E. Analyzing the Effect of Increasing Number of Samples on PAL

In Section 5.1, we show that PAL outperforms strong baselines both for a single sample and by drawing 40 samples and using majority voting. Figure 12 illustrates the trends for cases when the number of samples drawn are between 1 and 40, and the interpolation estimates demonstrate that PAL remains competitive throughout the number of samples.

[2] https://github.com/openai/openai-cookbook

```
def solution():
"""Shawn has five toys. For Christmas, he got two toys each from his
 ;  mom and dad. How many toys does he have now?"""
    toys_initial = 5
    mom_toys = 2
    dad_toys = 2
    total_received = mom_toys + dad_toys
    total_toys = toys_initial + total_received
    result = total_toys
    return result
```

(a) Original Example

```
def solution():
    return 5 + 2 + 2
```

(b) Succinct Code

```
def solution():
"""Shawn has 10312864 toys. For Christmas, he got 13267894 toys each
    from his mom and dad. How many toys does he have now?"""
    toys_initial = 10312864
    mom_toys = 13267894
    dad_toys = 13267894
    total_received = mom_toys + dad_toys
    total_toys = toys_initial + total_received
    result = total_toys
    return result
```

(c) Hard Examples in Prompt (PAL)

```
Example(
    question="Shawn has 10312864 toys. For Christmas, he got 13267894
        toys each from his mom and dad. How many toys does he have
        now?",
    thought="Shawn started with 10312864 toys. If he got 13267894 toys
        each from his mom and dad, then that is 26535788 more toys.
        10312864 + 26535788 = 36848652.",
    answer="36848652",
),
```

(d) Hard Examples in Prompt (CoT)

```
def solution():
"""Shawn has five toys. For Christmas, he got two toys each from his
 ;  mom and dad. How many toys does he have now?"""
    toys_initial = 5
    mom_toys = 2
    dad_toys = 2
    total_received = mom_toys + dad_toys
    total_toys = toys_initial + total_received
    result = total_toys
    return result
    ans = 9
```

(e) Generating Answers Directly

Figure 11: Ablations of the original example solution for the few-shot prompting experiment.

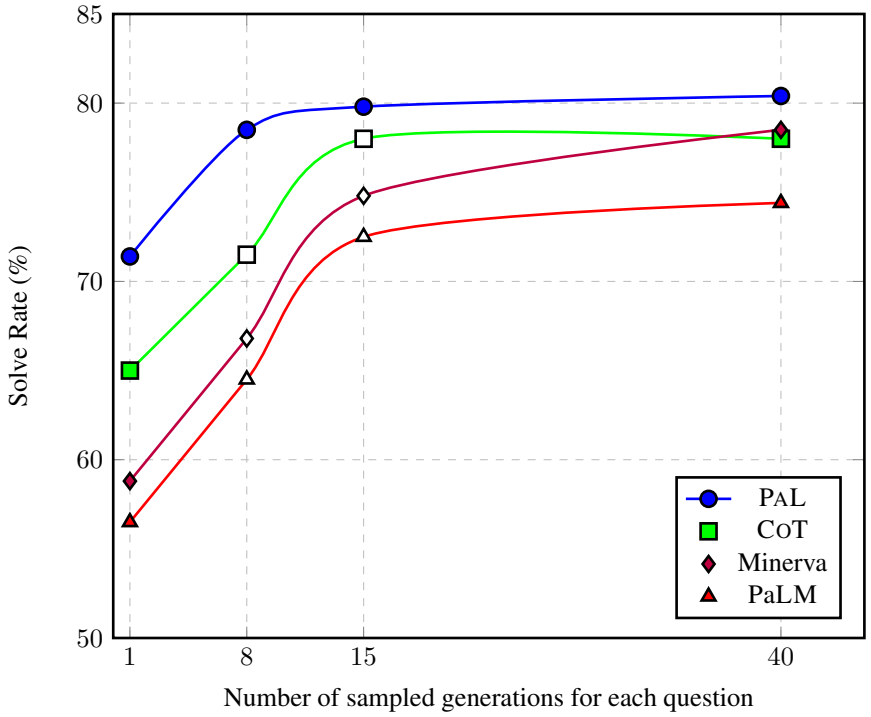


Figure 12: Comparison of solve rates between PAL and baselines as the number of samples increases from 1 to 40. Note that the solve rates for the baselines (PaLM, CoT, Minerva) are obtained through logistic interpolation of solve rates at 1 and 40

F. Standard Deviations Across Multiple Order of Prompts

For each math reasoning task, we run inference using three random orderings of the prompts. As shown in Table 9, the standard deviation between the results obtained from the three different seeds is minimal.

	CoT		PaL	
	Average	Standard Deviation	Average	Standard Deviation
GSM8K	65.6	1.10	72.0	0.16
SVAMP	74.8	0.19	79.4	0.20
ASDIV	76.9	0.65	79.6	0.14
GSM-HARD	23.3	0.49	61.2	0.91
MAWPS-SingleEq	89.1	0.54	96.1	0.30
MAWPS-SingleOp	91.9	0.55	94.6	0.36
MAWPS-AddSub	86.0	0.62	92.5	0.34
MAWPS-MultiArith	95.9	0.51	99.2	0.48

Table 9: Standard deviations for three runs for the math reasoning datasets.

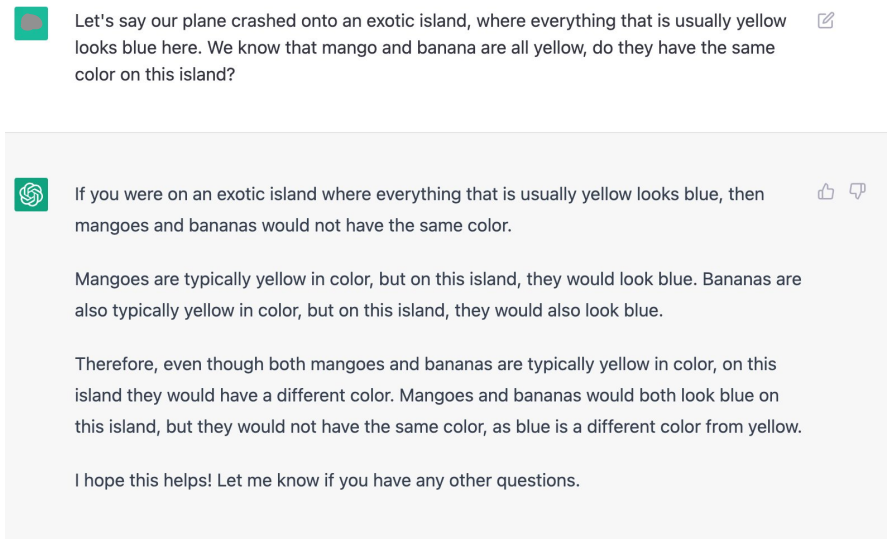
G. PaL Beyond Benchmarks

We argue that symbolic reasoning is a crucial component in solving a wide range of tasks. In this section, we demonstrate examples of tasks that may not initially appear to require using programs as intermediate reasoning steps, but can be improved through the use of PaL-style reasoning. We demonstrate these examples using the ChatGPT tool.[3] In contrast to the in-context-learning methods we used in the main paper, here we instruct ChatGPT to perform program-aided reasoning through one of the user utterances.

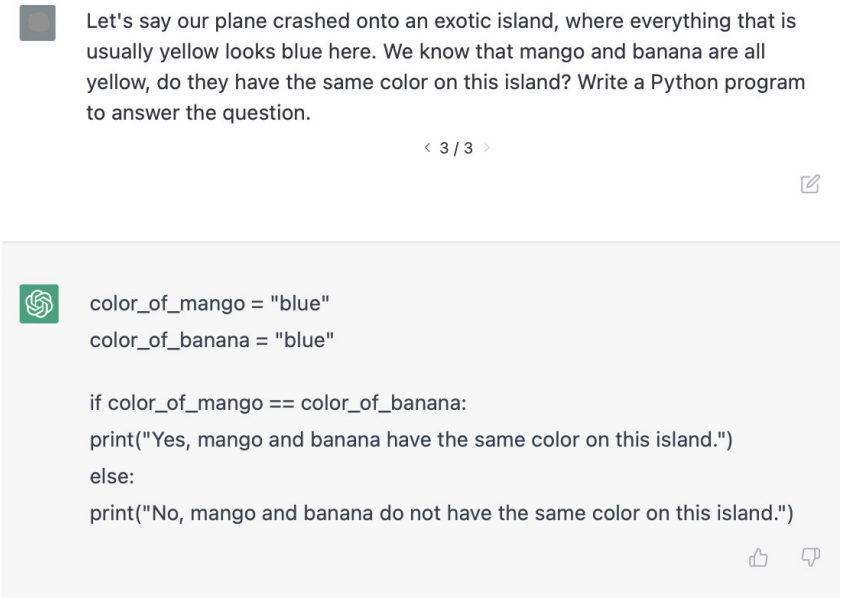
In Figure 13, in CoT-style reasoning, while the reasoning chain is correct, the final answer is wrong. In contrast, PaL-style reasoning could not only accurately extract the color of objects from the question but also produce the correct lines of code to branch to different situations that yield their corresponding correct answers.

A more intriguing example is letting an LLM count the number of letters in the word "intriguing". In Figure 14a, while the step-by-step explanation appears reasonable by splitting the letters by spaces, ChatGPT does not change the answer after this explicit reasoning and insists on the wrong answer. Explicitly instructing the model to perform step-by-step reasoning *before* answering the question still yields the wrong answer. In contrast, PaL-style reasoning only takes a few lines of code, and the execution does produce the correct answer, in this case. These examples indicate that PaL can benefit even an ostensibly powerful model like ChatGPT.

[3] chat.openai.com

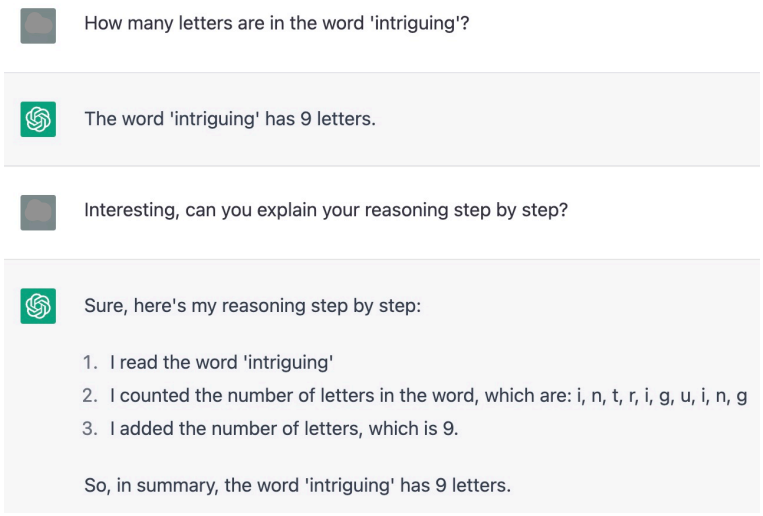


(a) In CoT style reasoning, the correct intermediate reasoning chain leads to wrong answers.

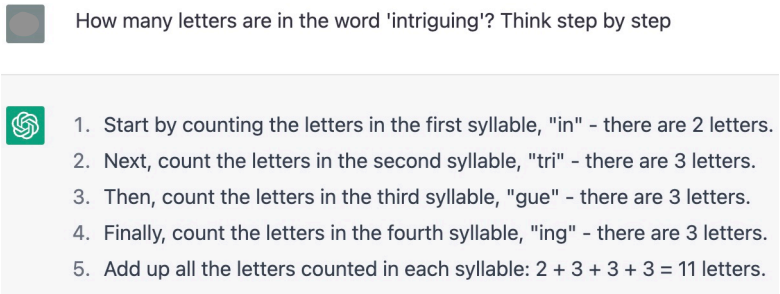


(b) In PAL, the execution of the code will produce the correct answer.

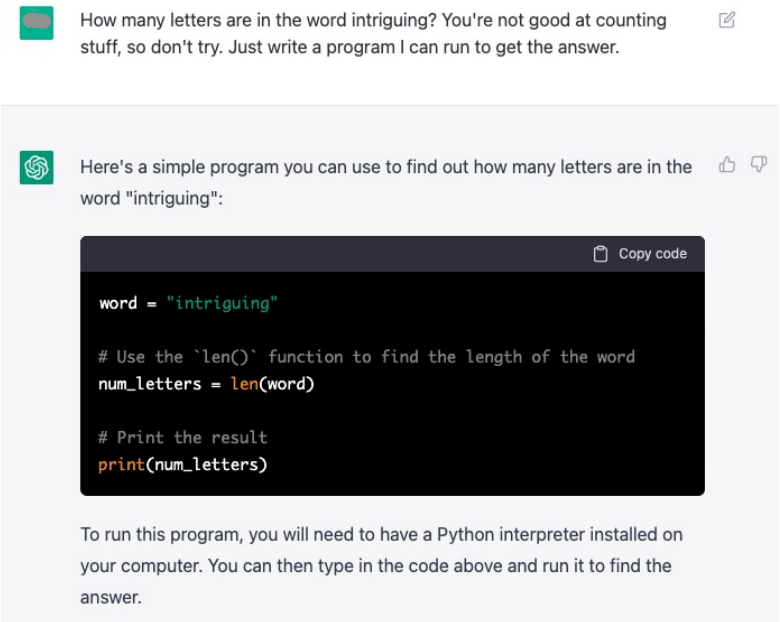
Figure 13: ChatGPT with PAL and CoT to answer a user-posted question



(a) Step-by-step reasoning struggle on counting the number of letters in the word "intriguing" which has ten letters.



(b) Explicitly instructing ChatGPT to reason step-by-step before generating answer still leads to the wrong answer.



(c) PAL takes a few lines of code and the execution could result in the correct answer.

Figure 14: ChatGPT with PAL and COT to answer a user-posted question

H. Closer Look into Token-level Behaviors of Different Mechanisms

Beyond empirical results, we make initial attempts to gain a deeper understanding of the behavior of LLMs with different reasoning mechanisms by looking into the token-level log-likelihood of reasoning chains produced by CoT and PAL. We randomly selected 20 questions from the COLORED OBJECTS dataset, along with their corresponding CoT and PAL solutions. We then manually compared the two mechanisms by focusing on tokens with a low log-likelihood.

Our analysis reveals that CoT often has lower confidence in tokens related to numbers and quantitative information, the grounded position of spatial adjectives (*e.g.,* right-most), properties such as the color of objects, and nouns that refer to the objects. Specifically, we found that this occurred in seven, six, two, and six examples out of the 20 we examined. In contrast, PAL uses list manipulations, such as `len(objects)`, and accesses objects and their associated properties through list indexing (*e.g.,* `object[3][0]`). We found that the LLM is typically confident in producing these programs. Furthermore, we observed that while CoT requires different expressions for the same concept in different contexts, PAL almost always uses the same expression, which is presumably more robust. For example, when there are five objects, CoT predicts "the right-most thing is the fifth item on the list", and "the right-most thing is the third item on the list" when the number of objects is three. Occasionally, CoT also predicts "the right-most thing is last item on the list" which does not provide more concrete information. On the contrary, PAL confidently predicts `objects[-1]` consistently. The more consistent and uniform use of expressions in PAL can be attributed to the explicit and defined nature of programming languages, which allows for clear and accurate expressions.

I. Datasets

In the following tables (Table 10,Table 11, Table 12), we presents statistics and examples for the datasets we considered.

Dataset	N	Example
Reasoning about Colored Objects	2000	On the table, you see a bunch of objects arranged in a row: a purple paperclip, a pink stress ball, a brown keychain, a green scrunchiephone charger, a mauve fidget spinner, and a burgundy pen. What is the color of the object directly to the right of the stress ball?
Penguins in a Table	149	Here is a table where the first line is a header and each subsequent line is a penguin: name, age, height (cm), weight (kg) Louis, 7, 50, 11 Bernard, 5, 80, 13 Vincent, 9, 60, 11 Gwen, 8, 70, 15 For example: the age of Louis is 7, the weight of Gwen is 15 kg, the height of Bernard is 80 cm. We now add a penguin to the table: James, 12, 90, 12 How many penguins are less than 8 years old?
Date Understanding	369	2015 is coming in 36 hours. What is the date one week from today in MM/DD/YYYY?

Table 10: Reasoning datasets about everyday objects and concepts.

Dataset	N	Example
Object Counting	1000	I have a chair, two potatoes, a cauliflower, a lettuce head, two tables, a cabbage, two onions, and three fridges. How many vegetables do I have?
Repeat Copy	32	Repeat the word duck four times, but halfway through also say quack.

Table 11: Reasoning datasets about algorithmic problems.

Dataset	N	Example
GSM8K (Cobbe et al., 2021)	1319	Olivia has $23. She bought five bagels for $3 each. How much money does she have left?
SVAMP (Patel et al., 2021)	1000	Each pack of dvds costs 76 dollars. If there is a discount of 25 dollars on each pack. How much do you have to pay to buy each pack?
ASDIV (Miao et al., 2020)	2096	Ellen has six more balls than Marin. Marin has nine balls. How many balls does Ellen have?
SINGLEOP (Koncel-Kedziorski et al., 2016)	562	If there are 7 bottle caps in a box and Linda puts 7 more bottle caps inside, how many bottle caps are in the box?
SINGLEEQ (Koncel-Kedziorski et al., 2016)	508	Benny bought a soft drink for 2 dollars and 5 candy bars. He spent a total of 27 dollars. How much did each candy bar cost?
ADDSUB (Koncel-Kedziorski et al., 2016)	395	There were 6 roses in the vase. Mary cut some roses from her flower garden. There are now 16 roses in the vase. How many roses did she cut?
MULTIARITH (Koncel-Kedziorski et al., 2016)	600	The school cafeteria ordered 42 red apples and 7 green apples for students lunches. But, if only 9 students wanted fruit, how many extra did the cafeteria end up with?

Table 12: Examples from math reasoning datasets used in this work. Table from Wei et al. (2022). Datasets vary in number of words per question, with 47 words per question in GSM8K, 21 words in SINGLEOP, and $\sim$30 words in the other datasets.

Question	CoT: GSM	CoT: GSMHard	Identical?
90 single use contacts come in 1 box and will last Pete 45 days. Each box is $100.00 and currently 10% off. If he buys 2 boxes of contact, how much will each pair of contacts cost?	90 contacts come in 1 box. Each box is $100.00. So 90 contacts cost $100.00. If he buys 2 boxes, then he will have 2 x 90 = 180 contacts. 180 contacts cost 2 x $100.00 = $200.00. So each pair of contacts costs $200.00 / 180 = $1.11. The answer is $1.11.	90 single use contacts come in 1 box. So 1 box will last Pete 45 days. Each box is $5451440 and currently 10% off. So each box is $5451440 * 0.9 = $4906296. If he buys 2 boxes of contact, he will have 2 * 90 = 180 contacts. So each pair of contacts will cost $4906296 / 180 = $2725849. The answer is $2725849	Yes
A chef bought 4 bags of onions. Each bag weighs 50 pounds. A pound of onions cost $1.50. How much did the chef spend?	The chef bought 4 bags of onions. Each bag weighs 50 pounds. So the total weight is 4 * 50 = 200 pounds. A pound of onions cost $1.50. So 200 pounds of onions cost 200 * 1.50 = 300 dollars. The answer is 300.	The chef bought 4 bags of onions. Each bag weighs 50 pounds. So the total weight is 4 * 50 = 200 pounds. A pound of onions cost $409201. So 200 pounds of onions cost 200 * 409201 = $81840200. The answer is $81840200.	Yes
A bag has a 5% discount. If it is marked $140, how much will you pay after the discount?	The discount is 5%. So you will pay 95% of the price. 95% of 140 is 133. The answer is 133.	The discount is 5%. So you will pay 95% of the price. 95% of $3921772.0722 is $3730654.0682. The answer is $3730654.0682	Yes
A basket of green food costs $25 and a basket of red food costs $18. If you buy 3 baskets of green food and red food, how much will you have to pay in total if you get $2 off for each basket of red food?	3 baskets of green food will cost 3 * 25 = 75 dollars. 3 baskets of red food will cost 3 * 18 = 54 dollars. But you get $2 off for each basket of red food. So you get 3 * 2 = 6 dollars off. So you pay 75 + 54 - 6 = 123 dollars. The answer is 123.	A basket of green food costs 9412763 dollars. A basket of red food costs 18 dollars. If you buy 3 baskets of green food and red food, you will have to pay 9412763 * 3 + 18 * 3 = 28238280 dollars. If you get 2 dollars off for each basket of red food, you will have to pay 28238280 - 18 * 2 = 28238244 dollars. The answer is 28238244	No
A bumper car rink has 12 red cars. They have 2 fewer green cars than they have red cars. They have 3 times the number of blue cars as they have green cars. The rink also has yellow cars. If the rink has 75 cars in total how many yellow cars do they have?	There are 12 red cars. There are 2 fewer green cars than red cars. So there are 12 - 2 = 10 green cars. There are 3 times as many blue cars as green cars. So there are 3 * 10 = 30 blue cars. There are 12 + 10 + 30 = 52 cars that are not yellow. So there are 75 - 52 = 23 yellow cars. The answer is 23.	The rink has 9318738 red cars. They have 2 fewer green cars than red cars. So they have 9318738 - 2 = 9318736 green cars. They have 3 times the number of blue cars as green cars. So they have 3 * 9318736 = 27956208 blue cars. The total number of cars is 75. So the number of yellow cars is 75 - 9318738 - 9318736 - 27956208 = -55,828,829. The answer is -55,828,829	Yes

Table 13: Analysis of generations from CODEX. Manual analysis of 25 randomly generated thoughts reveals that 16 out of 25 thoughts were identical, whereas others were close paraphrases.

I.1. Creating GSM-HARD

While replacing numbers in the question is easy using pattern matching, a more challenging aspect is recalculating the correct answer. GSM8K evaluation set contains 1319 samples, which is prohibitively expensive to perform manual re-calculation. Instead, we leverage PAL to assist obtaining the correct answers. For 71% of the examples where PAL is correct on GSM8K, we utilize the generated program and replace the initial value with the larger values. For example, if we create a harder version of the problem in Figure 3 by replacing $23 dollars with $15687 dollars, we correspondingly replace `money_initial=23` to `money_initial=15678`. Running the program could automatically produce the correct answer of the harder question. Notably, this annotation process assumes that a program that produces a correct answer to a GSM8K question indicates the correctness of the program itself. While this is not guaranteed due to possible spurious correlations, we manually checked 25 programs and found all of them are correct. For the incorrect 29% of the cases, we run PAL again and perform nucleus sampling (Holtzman et al., 2019) with temperature 0.7, and repeat the above process if any correct solution is found. Finally, the authors manually annotate the correct answer for 50 remaining cases that PAL was not able to solve after 100 iterations.

Note that the GSM-HARD benchmark was created automatically, it sometimes contains negative target values or target values that do not adhere to commonsense (*e.g., John bought 3.5 apples*). Unfortunately, we do not have the resources to manually annotate all examples, so our assumption is that there is a penalty of 5%-10% drop in performance for all models and prompting approaches that are evaluated on this benchmark. Since this penalty is similar to all approaches, we believe that the relative comparison between different approaches is the right thing to measure.

I.2. GSM-HARD Analysis

Table 13 shows thoughts generated with CoT on GSM8K and GSM-HARD. A manual analysis reveals that a majority of the generated thoughts (16/25) were identical for GSM8K and GSM-HARD, indicating that larger numbers primarily diminish performance due to failure of LLM to do arithmetic..

J. Generalization of PAL to Least-to-Most Prompting

```
Q: Four years ago, Kody was only half as old as Mohamed. If Mohamed is currently
↪  twice 30 years old, how old is Kody?
A: To answer the question "How old is Kody?", we need to know: "How old is Mohamed?",
↪  "How old was Mohamed four years ago?", "How old was Kody four years ago?".
```

(a) Least-to-Most Math Reducing Prompt

```
Four years ago, Kody was only half as old as Mohamed. If Mohamed is currently twice
↪  30 years old, how old is Kody?

Q: How old was Mohamed four years ago?
A: We were told that Mohamed is currently twice 30 years old, so he is currently 30 *
↪  2 = 60 years old. That means that four years ago he must have been 60 - 4 = 56
↪  years old. The answer is 56.

Q: How old is Kody?
A: Four years ago, Kody was half as old as Mohamed, so Kody must have been 56 / 2 =
↪  28 years old then. Since Kody was 28 years old four years ago, she must now be 28
↪  + 4 = 32 years old. The answer is 32.
```

(b) Least-to-Most Math Solving Prompt

```
# Four years ago, Kody was only half as old as Mohamed. If Mohamed is currently twice
    30 years old, how old is Kody?

# How old was Mohamed four years ago?
mohamed_age_current = 30 * 2
mohamed_age_4_years_ago = mohamed_age_current - 4

# Final Question: How old is Kody?
kody_age_4_years_ago = mohamed_age_4_years_ago / 2
kody_age_current = kody_age_4_years_ago + 4
answer = kody_age_current
```

(c) PAL Math Solving Prompt

Figure 15: Prompts for Math data sets.

Previous experiments focus on the COT technique. This section examines if PAL generalizes to other prompt types. We consider a strong alternative prompting strategy LEAST-TO-MOST (Zhou et al., 2022). LEAST-TO-MOST solves problems in two stages, problem-reducing and problem-solving. Problem reducing stage turns the problem into sub-problems, and the solving stage solves them sequentially. It keeps two prompts, each for an individual stage. To patch LEAST-TO-MOST prompts with PAL, we adopt a simple and straightforward approach: we note that problem reduction requires logically thinking in NL while solving requires the precision that PL offers. We therefore keep the original reducing prompts while only turning solution segments in the solving scripts in PL. We show an example reducing prompt, original solving prompt, and PAL solving prompt in Figure 15. Note that one unique property of PAL solving can naturally use previous questions' answers as the symbol values are shared. In comparison, the original solving script needs to explicitly re-cite answers from previous answers.

Dataset (500 examples)	LEAST-TO-MOST	LEAST-TO-MOST + PAL
GSM8K	67.2	**72.8**
SVAMP	75.2	**78.2**

Table 14: Results on GSM8K and SVAMP with LEAST-TO-MOST and LEAST-TO-MOST with PAL solving prompt.

For our analysis, we consider the Math data sets GSM8K, and SVAMP as Zhou et al. (2022) found Least-to-Most helps solve complex math problems. We patch the GSM8K prompt from the Zhou et al. (2022) into PAL. Note that the other tasks in

Zhou et al. (2022), like "concatenating last letters" from several words, require simple routines and are trivially solvable by PAL. We experiment with subsets of 500 examples and record results in Table 14. Here we see PAL can take advantage of the problem decomposition offered by the LEAST-TO-MOST reducing and further leverage the arithmetic capability in the Python runtime to achieve additional performance gains.

K. Prompts

We show here example PAL prompts we used for each data set. We show one example for each of the few-shot prompts. The fulls prompt can be found in our released code.

K.1. Reasoning about Colored Objects

```
# Q: On the table, you see a bunch of objects arranged in a row: a purple paperclip,
    a pink stress ball, a brown keychain, a green scrunchiephone charger, a mauve
    fidget spinner, and a burgundy pen. What is the color of the object directly to
    the right of the stress ball?
# Put objects into a list to record ordering
objects = []
objects += [('paperclip', 'purple')] * 1
objects += [('stress ball', 'pink')] * 1
objects += [('keychain', 'brown')] * 1
objects += [('scrunchiephone charger', 'green')] * 1
objects += [('fidget spinner', 'mauve')] * 1
objects += [('pen', 'burgundy')] * 1
# Find the index of the stress ball
stress_ball_idx = None
for i, object in enumerate(objects):
    if object[0] == 'stress ball':
        stress_ball_idx = i
        break
# Find the directly right object
direct_right = objects[stress_ball_idx+1]
# Check the directly right object's color
direct_right_color = direct_right[1]
answer = direct_right_color
```

K.2. Penguins in a Table

```
"""Q: Here is a table where the first line is a header and each subsequent line is a
    penguin: name, age, height (cm), weight (kg)  Louis, 7, 50, 11 Bernard, 5, 80, 13
    Vincent, 9, 60, 11 Gwen, 8, 70, 15 For example: the age of Louis is 7, the weight
    of Gwen is 15 kg, the height of Bernard is 80 cm.  We now add a penguin to the
    table: James, 12, 90, 12
How many penguins are less than 8 years old?
"""
# Put the penguins into a list.
penguins = []
penguins.append(('Louis', 7, 50, 11))
penguins.append(('Bernard', 5, 80, 13))
penguins.append(('Vincent', 9, 60, 11))
penguins.append(('Gwen', 8, 70, 15))
# Add penguin James.
penguins.append(('James', 12, 90, 12))
# Find penguins under 8 years old.
penguins_under_8_years_old = [penguin for penguin in penguins if penguin[1] < 8]
# Count number of perguins under 8.
num_penguin_under_8 = len(penguins_under_8_years_old)
answer = num_penguin_under_8
```

Figure 17

K.3. Date Understanding

```
# Q: 2015 is coming in 36 hours. What is the date one week from today in MM/DD/YYYY?
# If 2015 is coming in 36 hours, then today is 36 hours before.
today = datetime(2015, 1, 1) - relativedelta(hours=36)
# One week from today,
one_week_from_today = today + relativedelta(weeks=1)
# The answer formatted with %m/%d/%Y is
one_week_from_today.strftime('%m/%d/%Y')
```

K.4. Math

```
#Q: Olivia has \$23. She bought five bagels for \$3 each. How much money does she have
    left?
money_initial = 23
bagels = 5
bagel_cost = 3
money_spent = bagels * bagel_cost
money_left = money_initial - money_spent
print(money_left)

#Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost
    2 more. How many golf balls did he have at the end of wednesday?
golf_balls_initial = 58
golf_balls_lost_tuesday = 23
golf_balls_lost_wednesday = 2
golf_balls_left = golf_balls_initial - golf_balls_lost_tuesday -
    golf_balls_lost_wednesday
print(golf_balls_left)

#Q: There were nine computers in the server room. Five more computers were installed
    each day, from monday to thursday. How many computers are now in the server room?

computers_initial = 9
computers_per_day = 5
num_days = 4  # 4 days between monday and thursday
computers_added = computers_per_day * num_days
computers_total = computers_initial + computers_added
print(computers_total)


#Q: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in
    the parking lot?
cars_initial = 3
cars_arrived = 2
total_cars = cars_initial + cars_arrived
print(total_cars)

#Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do
    they have left in total?

leah_chocolates = 32
sister_chocolates = 42
total_chocolates = leah_chocolates + sister_chocolates
chocolates_eaten = 35
chocolates_left = total_chocolates - chocolates_eaten
print(chocolates_left)
```

Figure 19: Prompt used for mathematical reasoning (1/2)

```
#Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops.
    How many lollipops did Jason give to Denny?
jason_lollipops_initial = 20
jason_lollipops_after = 12
denny_lollipops = jason_lollipops_initial - jason_lollipops_after
print(denny_lollipops)



#Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today.
    After they are done, there will be 21 trees. How many trees did the grove workers
    plant today?
trees_initial = 15
trees_after = 21
trees_added = trees_after - trees_initial
print(trees_added)


#Q: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How
    many toys does he have now?

toys_initial = 5
mom_toys = 2
dad_toys = 2
total_received = mom_toys + dad_toys
total_toys = toys_initial + total_received
print(total_toys)
```

Figure 20: Prompt used for mathematical reasoning (2/2)

K.5. Object Counting

```
# Q: I have a chair, two potatoes, a cauliflower, a lettuce head, two tables, a
    cabbage, two onions, and three fridges. How many vegetables do I have?

# note: I'm not counting the chair, tables, or fridges
vegetables_to_count = {
    'potato': 2,
    'cauliflower': 1,
    'lettuce head': 1,
    'cabbage': 1,
    'onion': 2
}
print(sum(vegetables_to_count.values()))


# Q: I have a drum, a flute, a clarinet, a violin, four accordions, a piano, a
    trombone, and a trumpet. How many musical instruments do I have?

musical_instruments_to_count = {
    'drum': 1,
    'flute': 1,
    'clarinet': 1,
    'violin': 1,
    'accordion': 4,
    'piano': 1,
    'trombone': 1,
    'trumpet': 1
}
print(sum(musical_instruments_to_count.values()))



# Q: I have a chair, two ovens, and three tables. How many objects do I have?

objects_to_count = {
    'chair': 1,
    'oven': 2,
    'table': 3
}
print(sum(objects_to_count.values()))
```

Figure 21: Prompt used for OBJECT COUNTING.

K.6. Repeat Copy

```
# Q: Repeat the word duck four times, but halfway through also say quack

result = []
for i in range(1, 5):
    result.append("duck")
    if i == 2:
        result.append("quack")
print(" ".join(result))


# Q: Print boolean eleven times, but after the 3rd and 8th also say correct

result = []
for i in range(1, 12):
    result.append("boolean")
    if i == 3 or i == 8:
        result.append("correct")
print(" ".join(result))

# Q: say java twice and data once, and then repeat all of this three times.

result = []
tmp = ["java", "java", "data"]
for i in range(3):
    result.extend(tmp)
print(" ".join(result))


# Q: ask a group of insects in what family? four times. after the fourth time say The
    happy family

result = []
tmp = []
for i in range(1, 5):
    tmp.append("a group of insects in what family?")
tmp.append("The happy family")
result.extend(tmp)
print(" ".join(result))
```

Figure 22: Prompt used for REPEAT COPY.

L. Success and Failure Modes in Symbolic Tasks

L.1. Colored Objects

```
# Find non-gold items to the right of the pencil
non_gold = [object for object in objects[i+1:] if object[1] != 'gold']
```

(a) Snippet of PAL doing a filter operation.

```
# Remove all pink objects
non_pink = [object for object in objects if object[1] != 'pink']

# Count number of crayons
crayons = [object for object in non_pink if object[0] == 'crayon']
answer = len(crayons)
```

(b) Snippet of PAL composing multiple operations.

Figure 23: Example model generation snippets on Reasoning about Color Object.

By manually examining the the model outputs, we observe that PAL often performs better on questions that involve *counting* objects that satisfy one or several conditions. For example, COT fails in the following example: *"On the desk, you see a bunch of items arranged in a row: a gold textbook, a purple puzzle, a teal necklace, and a silver pencil. How many non-gold items do you see to the right of the pencil?"*. With pure NL reasoning, a LLM can easily lose track of the objects and output the wrong answer of *"The number of non-gold items to the right of the pencil is two. So the answer is two."*

In comparison, PAL is able to accurately construct the object lists with correct order and attributes. Further, it can precisely leverage the simple yet complete PL syntax: it composes routines with functional operators from elementary builtin operation/operators. Figure 4 lists the last a few lines of the solution to the question described above generated by PAL, which perform filtering over objects. PAL can further compose such operations across multiple reasoning steps as shown in Figure 23b, where it first filters out all pink objects and then scans the remaining to count crayons. Other reasoning such as "how many objects are left after filtering" can be easily tackled precisely with an runtime.

L.2. Penguins in a Table

```
...This question focuses on age. We know the following: Louis is 7 years old, Bernard
↪  is 5 years old, Vincent is 9 years old, and Gwen is 8 years old.Now, we add James
↪  to this table: James is 12 years old. We then delete the penguin named Bernard
↪  from the table.The penguins that are less than 8 years old are Louis and Bernard.
↪  There are 2 penguins less than 8 years old. So the answer is 2.
```

```
...
# Delete penguin Bernard.
penguins = [penguin for penguin in penguins if penguin[0] != 'Bernard']
# Find penguins under 8 years old.
penguins_under_8_years_old = [penguin for penguin in penguins if penguin[1] < 8]
# Count number of perguins under 8.
num_penguin_under_8 = len(penguins_under_8_years_old)
answer = num_penguin_under_8
```

Figure 24: Generated Snippets for one question (*...We then delete the penguin named Bernard from the table. How many penguins are less than 8 years old?*) in Penguins in a Table

.

In this task, we found a typical failure mode of the language is its inability to stably tracks states in its past generation. Figure 24 lists the generations of COT and PAL to the question that contains the removal of a penguin (*"...We then delete the penguin named Bernard from the table. How many penguins are less than 8 years old?"*). Although COT picks up the critical information that "penguins that are less then 8 years old are Louis and Bernard", and that "Bernard is deleted". It still fails to aggregate the information properly and infer that there is one penguin less then 8 left in the end. In comparison, PAL expresses this dynamic through manipulating a `penguins` list by filtering out the penguin whose name is "Bernard",

and maintaining a `penguins_under_8_years_old` list. It offloads tracking exact the values (in this case, the length of the list) to the deterministic Python runtime.

L.3. Date Understanding

```
Q: Jane was born on the last day of Feburary in 2001. Today is her 16-year-old
↪  birthday. What is the date 24 hours later in MM/DD/YYYY?

A: The last day of February is the 28th, so Jane was born on 02/28/2001. Today is her
↪  16-year old birthday, so today is 02/28/2017. So 24 hours later is 02/29/2017. So
↪  the answer is 02/29/2017.
```

```
# Q: Jane was born on the last day of Feburary in 2001. Today is her 16-year-old
    birthday. What is the date 24 hours later in MM/DD/YYYY?
# If Jane was born on the last day of Feburary in 2001 and today is her 16-year-old
    birthday, then today is 16 years later.
today = datetime(2001, 2, 28) + relativedelta(years=16)
# 24 hours later,
later = today + relativedelta(hours=24)
# The answer formatted with %m/%d/%Y is
later.strftime('%m/%d/%Y')
```

Figure 25: Example model generation on Date Understanding.

We found this especially common when the time deltas are across the month boundary. We show an example in Figure 25. Here with CoT prompting, the LLM expresses the knowledge of the 28-day-long February, yet it still outputs *02/29/2017* as the final answer. With PAL, the actual calendar is accurate as a program handles the operation.