# Solving Math Word Problems by Combining Language Models With Symbolic Solvers

**Joy He-Yueya, Gabriel Poesia, Rose E. Wang, Noah D. Goodman**
Stanford University
{heyueya, poesia, rewang, ngoodman}@stanford.edu

## Abstract

Automatically generating high-quality step-by-step solutions to math word problems has many applications in education. Recently, combining large language models (LLMs) with external tools to perform complex reasoning and calculation has emerged as a promising direction for solving math word problems, but prior approaches such as Program-Aided Language model (PAL) are biased towards simple procedural problems and less effective for problems that require declarative reasoning. We propose an approach that combines an LLM that can incrementally formalize word problems as a set of variables and equations with an external symbolic solver that can solve the equations. Our approach achieves comparable accuracy to the original PAL on the GSM8K benchmark of math word problems and outperforms PAL by an absolute 20% on ALGEBRA, a new dataset of more challenging word problems extracted from Algebra textbooks. Our work highlights the benefits of using declarative and incremental representations when interfacing with an external tool for solving complex math word problems. Our data and prompts are publicly available at https://github.com/joyheyueya/declarative-math-word-problem.

## 1 Introduction

Learning to solve mathematical word problems is an important skill but can be challenging for students. [5, 13]. A tool that can automatically generate step-by-step solutions to such problems has the potential to provide personalized support for students working through word problems [14, 6] and help educators with curriculum development [12].

Using few-shot prompting over large language models (LLMs) has recently emerged as a promising approach for solving math word problems [15, 17, 7]. The chain-of-thought (COT) [15] prompting method presents explicit intermediate reasoning steps to the LLM to further enhance its reasoning capability. However, LLMs often struggle with performing arithmetic operations [8, 9, 15]. To address this, [15] uses an external calculator to evaluate the arithmetic operations in the generated reasoning steps. Program-Aided Language model (PAL) [7] extends this idea by generating Python programs as reasoning steps, offloading all calculations to a Python interpreter. Although programs offer a direct representation of *procedures*, they require special devices to represent more abstract mathematical *declarations*. For example, a statement like $a = b + 1$ can be directly interpreted as a variable assignment in Python if $b$ is known, but not if $b$ is unknown. Nonetheless, the equation remains a valid mathematical expression even when $b$ is unknown, suggesting that we instead want to allow models to perform mathematical *declarations* beyond those that yield a procedure (for a full example, see the problem in Figure 1).

In this work, we present an approach that combines an LLM, which can incrementally formalize word problems as a set of variables and equations, with an external symbolic solver that can solve the equations. Our approach achieves comparable performance to the original PAL on the GSM8K

37th Conference on Neural Information Processing Systems (NeurIPS 2023) Workshop on MATH-AI.

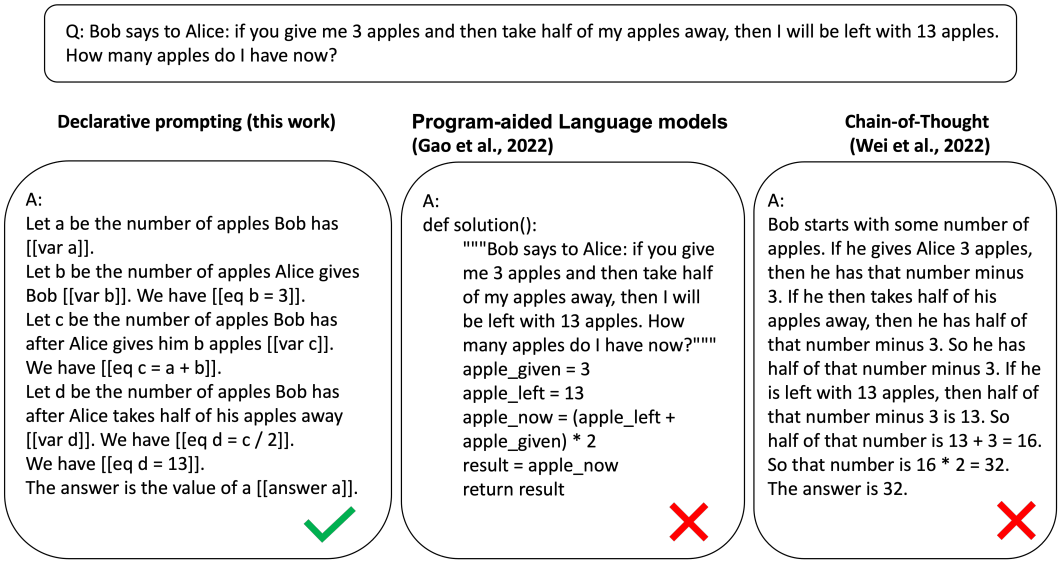


Figure 1: Declarative solutions are typically more intuitive to write than procedural solutions for challenging algebra word problems. PAL and CoT try to generate procedural solutions that describe a set of plans for achieving the goal, which are incorrect in this case. The DECLARATIVE prompting generates a correct solution that describes the properties of the goal, which is generally more appropriate for hard problems with no obvious procedural solutions.

[4] benchmark of math word problems. To evaluate current approaches on more challenging word problems, we introduce ALGEBRA, a dataset of 222 word problems collected from open access Algebra textbooks. We show that our approach outperforms PAL by an absolute $20\%$ on ALGEBRA. Our work highlights the effectiveness of incrementally generating declarative formalizations when interfacing with an external tool for solving complex math word problems.

## 2 Related work

Recent studies have explored the use of few-shot prompting over LLMs for solving math word problems [15, 17, 7]. The chain-of-thought [15] prompting method presents explicit intermediate reasoning steps to the LLM to improve its reasoning capability. Since LLMs often make arithmetic errors [8, 9, 15], several prior works [15, 3] have experimented with using an external calculator to carry out the operations generated by LLMs. This generally improves final performance by less than $5\%$ on GSM8K. Program-Aided Language model [7] extends to more complex arithmetic by generating Python programs as reasoning steps and using a Python interpreter to perform the calculations. However, generating Python programs carries a strong bias toward procedural calculations and does not work well for word problems that do not have a straightforward procedural solution.

## 3 Our Approach: Equipping an LLM With an External Symbolic Solver

Our approach for solving a math word problem consists of two steps: *(1) declarative and incremental formalization using an LLM* and *(2) solving equations using a symbolic solver*.

### 3.1 Declarative and incremental formalization using an LLM

To solve a math word problem, we first use an LLM to formalize the problem as a set of variables and equations. Recently, using few-shot prompting over LLMs has emerged as an effective approach for natural language understanding and decomposition. Few-shot prompting is a technique that uses LLMs to solve a task by providing the LLMs with a few demonstrations of the task as part of the input at inference time [1]. In this technique, the demonstrations (i.e., examples of input-output pairs) are concatenated into a prompt, which is passed to the model along with the new input to generate

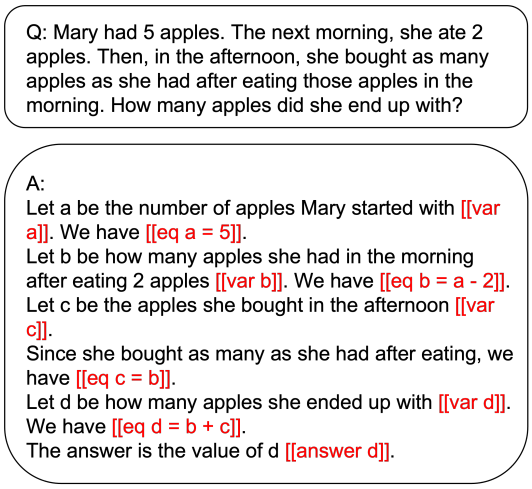


Figure 2: An example of a math word problem and its solution from the DECLARATIVE prompt. Variables and equations are in red.

an output. Formally, a set of $k$ input-output examples $\{(x_i, y_i)\}_{i=1}^{k}$ are concatenated in a prompt $p \equiv (x_1, y_2)||(x_1, y_2)||...||(x_k, y_k)$ where $||$ denotes the concatenation of examples. At inference time, $p||x_{test}$ is passed to the model where $x_{test}$ denotes a new input instance, and the model attempts to complete $p||x_{test}$ by generating the output $y_{test}$.

To formalize word problems using few-shot prompting, we introduce the DECLARATIVE prompt $p \equiv (x_1, y_2)||(x_1, y_2)||...||(x_k, y_k)$ where $x_i$ is the word problem in natural language, and $y_i$ is the step-by-step solution to $x_i$. In the DECLARATIVE prompt, $y_i$ consists of interleaved natural language statements and formal variable or equation declarations in double-square brackets. Our approach aims to generate solutions that formalize word problems based on a set of principles listed in Appendix A. Figure 2 shows an example used in the DECLARATIVE prompt that we created according to these principles. To solve a new word problem, $x_{test}$, we append it to $p$ and pass $p||x_{test}$ to an LLM, which generates $y_{test}$ as the solution for $x_{test}$.

### 3.2 Solving equations using a symbolic solver

The step-by-step solution generated by the LLM using the DECLARATIVE prompt includes the list of variables and equations that describe the word problem but does not provide the final answer (see Figure 2). Instead of relying on the LLM to solve the equations directly, we pass the equations to an external symbolic solver to do the calculation. In this work, we use SymPy [11], a Python library for symbolic computation, to algebraically solve a system of equations extracted from the solution generated by the LLM.

## 4 Experimental Setup

### 4.1 Datasets

We evaluate our approach on two math word problem datasets: GSM8K [4] and a new dataset called ALGEBRA [1]. We use the GSM8K test set, which contains 1319 math word problems at grade-school level. To evaluate our approach on more challenging problems, we curated ALGEBRA, which consists of 222 word problems from two open-access Algebra textbooks: Basic Algebra with Applications ([16]; released under the Creative Commons Attribution-ShareAlike license) and Elementary Algebra 2e ([10]; released under the Creative Commons Attribution license). We took every word problem that has a solution in these textbooks. The resulting dataset includes word problems covering all topics leading up to System of Equations, with the exception of problems related to geometry, graphing, or inequalities.

[1]The ALGEBRA dataset is publically available at https://github.com/joyheyueya/declarative-math-word-problem.

### 4.2 Baselines and variants of the DECLARATIVE prompting

We consider three methods: chain-of-thought (COT) prompting [15], Program-Aided Language model (PAL) [7], and our DECLARATIVE prompting combined with SymPy (DECLARATIVE + SymPy). We created two different prompts for each prompting method. The first prompt (8-shot) uses the same set of eight examples used in prior work [15]. The second prompt (3-shot) uses three examples that we designed to help illustrate step-by-step and declarative thinking and the formalization format we expect.

For our DECLARATIVE prompting method, we experimented with three variants.

1. DECLARATIVE$_{\text{3-shot}}$ + principles + SymPy: adding the list of principles in Table 2 at the beginning of the prompt (see an example in Figure 3a).
2. DECLARATIVE$_{\text{3-shot}}$ + principles: using the LLM to directly calculate the value of the goal variable (see an example in Figure 3b).
3. ONE-STEP DECLARATIVE$_{\text{3-shot}}$ + SymPy: formalizing the word problem in a single step instead of incrementally (see an example in Figure 4).

We use Codex (code-davinci-002) [2] as the LLM for all methods. We use top-1 decoding and a temperature of 0. We set `max_tokens` to be 600.

## 5 Results

| | GSM8K | ALGEBRA |
|---|---|---|
| COT$_{\text{8-shot (original)}}$ | $62.5 \pm 0.16$ | $45.3 \pm 0.56$ |
| COT$_{\text{3-shot (ours)}}$ | $58.9 \pm 0.16$ | $47.9 \pm 1.18$ |
| PAL$_{\text{8-shot (original)}}$ | $70.2 \pm 0.25$ | $51.7 \pm 0.21$ |
| PAL$_{\text{3-shot (ours)}}$ | **73.3** $\pm 0.13$ | $56.2 \pm 0.21$ |
| DECLARATIVE$_{\text{8-shot}}$ + SymPy | 64.7 | - |
| DECLARATIVE$_{\text{3-shot}}$ + SymPy | $66.0 \pm 0.33$ | - |
| DECLARATIVE$_{\text{3-shot}}$ + principles + SymPy | $69.4 \pm 0.65$ | **76.3** $\pm 0.93$ |
| DECLARATIVE$_{\text{3-shot}}$ + principles | $22.4 \pm 0.27$ | - |
| ONE-STEP DECLARATIVE$_{\text{3-shot}}$ + SymPy | $57.5 \pm 0.06$ | - |

Table 1: Problem solve rate (%) on GSM8K and ALGEBRA. We report the average and standard deviation across three runs. The highest number on each dataset is in **bold**. For COT and PAL, we ran both the 8-shot prompt used in the original papers and the 3-shot prompt we created.

On GSM8K (Table 1), our 3-shot prompt leads to a better performance than the original 8-shot prompt for PAL and DECLARATIVE. PAL outperforms DECLARATIVE across both sets of comparable examples, but using our DECLARATIVE prompting method with the 3-shot prompt (DECLARATIVE$_{\text{3-shot}}$ + principles + SymPy) gives a performance equivalent to the original PAL (PAL$_{\text{8-shot (original)}}$).

Interestingly, prepending the list of principles to the DECLARATIVE prompt (DECLARATIVE$_{\text{3-shot}}$ + principles + SymPy) leads to a better performance on GSM8K than DECLARATIVE$_{\text{3-shot}}$ + SymPy. Asking the LLM to solve the equations directly leads to a dramatic drop in accuracy (from $69.4\%$ to $22.4\%$), which highlights the benefit of using an external solver. Additionally, our DECLARATIVE prompting benefits from incremental formalization, as shown by the performance gap between the incremental version (DECLARATIVE$_{\text{3-shot}}$ + principles + SymPy) and the non-incremental variant (ONE-STEP DECLARATIVE$_{\text{3-shot}}$ + SymPy).

On ALGEBRA (Table 1), our approach (DECLARATIVE$_{\text{3-shot}}$ + principles + SymPy) achieves the highest accuracy among all methods, outperforming PAL by an absolute $20\%$. The accuracy of the original COT drops from $62.5\%$ on GSM8K to $45.3\%$ on ALGEBRA, which demonstrates that problems in ALGEBRA are generally harder than those in GSM8K. The main reason that the DECLARATIVE prompting method works better than COT and PAL on ALGEBRA is that it is less intuitive to generate procedural solutions to Algebra problems that require declarative reasoning (see

an example in Figure 1). Although our 3-shot prompt improves the performance of CoT and PAL on ALGEBRA compared to the original 8-shot prompt, our DECLARATIVE method is still much more effective than CoT and PAL.

## 6 Conclusion

We present an approach for automatically generating step-by-step solutions to math word problems by equipping an LLM with an external symbolic solver. Our approach uses an LLM to incrementally formalize word problems as variables and equations and avoids arithmetic errors by using an external symbolic solver that can solve the equations. Our approach achieves comparable accuracy to the original PAL on GSM8K and improves over PAL by an absolute $20\%$ on a new dataset consisting of harder word problems from Algebra textbooks. We demonstrate the effectiveness of using declarative formalization when interfacing with an external tool for solving complex math word problems. Additionally, encouraging incremental formalization is beneficial, especially when using declarative representations. Our approach is particularly useful for math education since many advanced math problems can be divided into separate conceptual pieces, with one piece being declarative and the other involving procedural knowledge.

## References


[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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

[3] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[4] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[5] D. D. Cummins. Children's interpretations of arithmetic word problems. *Cognition and instruction*, 8(3):261–289, 1991.

[6] J. del Olmo-Muñoz, J. A. González-Calero, P. D. Diago, D. Arnau, and M. Arevalillo-Herráez. Intelligent tutoring systems for word problem solving in covid-19 days: could they have been (part of) the solution? *ZDM–Mathematics Education*, pages 1–14, 2022.

[7] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2022.

[8] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[9] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022.

[10] L. Marecek, M. Anthony-Smith, and A. H. Mathis. *Elementary Algebra 2E*. OpenStax, 2020.

[11] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.

[12] O. Polozov, E. O'Rourke, A. M. Smith, L. Zettlemoyer, S. Gulwani, and Z. Popović. Personalized mathematical word problem generation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[13] N. Pongsakdi, A. Kajamies, K. Veermans, K. Lertola, M. Vauras, and E. Lehtinen. What makes mathematical word problem solving challenging? exploring the roles of word problem characteristics, text comprehension, and arithmetic skills. *ZDM*, 52:33–44, 2020.

[14] S. Ritter, J. R. Anderson, K. R. Koedinger, and A. Corbett. Cognitive tutor: Applied research in mathematics education. *Psychonomic bulletin & review*, 14:249–255, 2007.

[15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[16] I. G. Zaigralin. *Basic Algebra with Applications*. Ivan G. Zaigralin, 6 edition, 2018.

[17] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, O. Bousquet, Q. Le, and E. Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

# A Principles for declarative solutions

| Principles for declarative solutions |
|---|
| 1. Each sentence in the solution either introduces a new variable or states a new equation. |
| 2. The last sentence gives the goal: which variable will contain the answer to the problem. |
| 3. Each equation only uses previously introduced variables. |
| 4. Each quantity is only named by one variable. |
| 5. The solution uses all the numbers in the question. |

Table 2: A list of principles we would like the solutions to satisfy.

# B Prompt examples

All the prompts used in this work are publicly available at https://github.com/joyheyueya/declarative-math-word-problem.

```
Let's solve mathematical word problems in a careful,
formal manner. The solution will follow the Peano
format:
1- Each sentence in the solution either introduces a new
variable or states a new equation.
2- The last sentence gives the goal: which variable will
contain the answer to the problem.
3- Each equation only uses previously introduced
variables.
4- Each quantity is only named by one variable.
5- Use all the numbers in the question.

Q: Mario and Luigi together had 10 years of experience
in soccer. Luigi had 3 more than Mario. How many did
Mario have?

Peano solution:
Let a be the number of years Mario had [[var a]].
Let b be the number of years Luigi had [[var b]]. We have
[[eq a + b = 10]]. We also have [[eq b = a + 3]].
The answer is the value of a [[answer a]].
```

(a) Adding principles to the beginning of the DECLARATIVE prompt.

```
Let's solve mathematical word problems in a careful,
formal manner. The solution will follow the Peano
format:
1- Each sentence in the solution either introduces a new
variable or states a new equation.
2- The last sentence gives the goal: which variable will
contain the answer to the problem.
3- Each equation only uses previously introduced
variables.
4- Each quantity is only named by one variable.
5- Use all the numbers in the question.

Q: Mario and Luigi together had 10 years of experience
in soccer. Luigi had 3 more than Mario. How many did
Mario have?

Peano solution:
Let a be the number of years Mario had [[var a]].
Let b be the number of years Luigi had [[var b]]. We have
[[eq a + b = 10]]. We also have [[eq b = a + 3]].
The answer is the value of a [[answer a = 3.5]].
```

(b) Adding principles to the beginning of the DECLARATIVE prompt and calculating the final answer. The final answer is in red.

Figure 3: The difference between "$\text{DECLARATIVE}_{\text{3-shot}}$ + principles + SymPy" and "$\text{DECLARATIVE}_{\text{3-shot}}$ + principles" is that "$\text{DECLARATIVE}_{\text{3-shot}}$ + principles + SymPy" passes the equations to SymPy to solve, but "$\text{DECLARATIVE}_{\text{3-shot}}$ + principles" asks the LLM to solve the equations directly.

```
Q: Mario and Luigi together had 10 years of experience
in soccer. Luigi had 3 more than Mario. How many did
Mario have?

A:
[[eq a = (10 - 3) / 2]]
[[answer a]]
```

Figure 4: An example of formalizing a math word problem in a single equation.