MATHSENSEI: MATHEMATICAL REASONING WITH A TOOL-AUGMENTED LARGE LANGUAGE MODEL

Debrup Das *[1], Debopriyo Banerjee[2], Somak Aditya[1], and Ashish Kulkarni[2]

[1]IIT Kharagpur
debrupdas@iitkgp.ac.in, saditya@cse.iitkgp.ac.in
[2]Rakuten Group, Inc. Rakuten Institute of Technology India
{debopriyo.banerjee,ashish.kulkarni}@rakuten.com

ABSTRACT

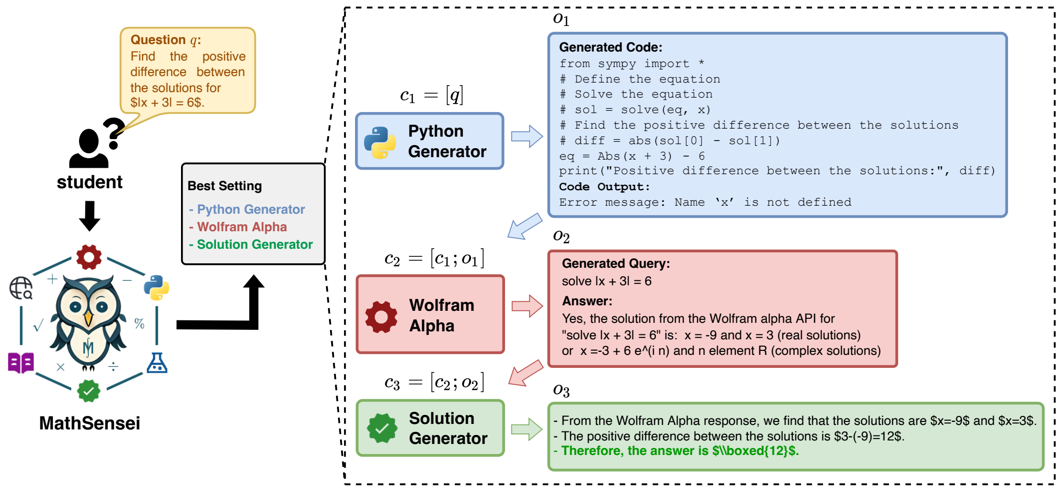
Large Language Models, augmented with specialized tools or modules, commonly referred to as *TALMs*, show superior reasoning abilities over generic LLMs, across different knowledge intensive Question Answering (QA) tasks. However, their efficacy on complex mathematical reasoning benchmarks, has remained largely unexplored. Moreover, existing research lacks the study of complementary benefits offered by diverse tool-sets towards solving mathematical problems. In this work, we present a TALM-based framework - MATHSENSEI, which is powered by a knowledge retriever (LLM or Bing Web Search), program generator + executor (Python), and symbolic problem solver (Wolfram-Alpha). We perform extensive ablations with various tool combinations, across multiple math sub-disciplines of different datasets. Our experiments also comprise evaluation of well-known planning algorithms such as REACT and Plan-And-Solve. MATHSENSEI outperforms GPT-3.5-TURBO with chain-of-thought (CoT) by 13.5 % on the MATH dataset. We observe that *TALM*s are beneficial for progressively increasing complexity of problems (such as AQuA, MMLU-Math, and higher level complex questions in MATH), and show minimal benefits over simpler math word problems (such as GSM-8k). The code and data are available at https://github.com/Debrup-61/MathSensei.

1 INTRODUCTION

The family of Large Language Models (LLMs), ranging from proprietary gpt-3.5-turbo, GPT-4 to open-source models, such as Llama 2, are considered as a one-stop solution for solving a broad spectrum of NLP tasks (Brown et al., 2020; Radford et al., 2019; Chowdhery et al., 2022; OpenAI, 2023). However, generic LLMs consistently fail to solve problems that require extensive mathematical, commonsense, abductive, and multi-hop reasoning (Lu et al., 2023b; Cobbe et al., 2021; Huang & Chang, 2023). For enhancing the reasoning abilities of LLMs, the research community has been exploring various techniques, such as - (1) **intelligent prompting variations**, such as chain of thought (Wei et al., 2022), program of thought (Chen et al., 2022), tree of thoughts (Yao et al., 2023), and self-refinement (Madaan et al., 2023), (2) **multi-model interaction frameworks**, such as Multi-agent Debate (Du et al., 2023; Liang et al., 2023) and Round-Table Conference (Chen et al., 2023b), (3) **tool-augmented LLMs** (TALMs) powered by external symbolic tools, APIs, and libraries.

Recent advancements in *TALM* frameworks, such as Chameleon (Lu et al., 2023a), OlaGPT (Xie et al., 2023), ART (Paranjape et al., 2023), and SocraticAI (Yang & Narasimhan, 2023) have explored the effectiveness of incorporating external tools for solving knowledge-intensive reasoning tasks and simple mathematical problems (such as arithmetic and algebra). However, the effectiveness of *TALM* framework is yet to be validated on diverse mathematical domains (e.g., PreAlgebra, Calculus, Geometry, Intermediate Algebra, Probability) at varying levels of difficulty. In addition,

*Work done during internship at Rakuten India Enterprise Private Limited.



Generated Code:

```
from sympy import *
# Define the equation
# Solve the equation
# sol = solve(eq, x)
# Find the positive difference between the solutions
# diff = abs(sol[0] - sol[1])
eq = Abs(x + 3) - 6
print("Positive difference between the solutions:", diff)
```

Code Output:

```
Error message: Name 'x' is not defined
```

Generated Query:
solve |x + 3| = 6

Answer:
Yes, the solution from the Wolfram alpha API for "solve |x + 3| = 6" is: x = -9 and x = 3 (real solutions) or x =-3 + 6 e^(i n) and n element R (complex solutions)

- From the Wolfram Alpha response, we find that the solutions are \$x=-9\$ and \$x=3\$.
- The positive difference between the solutions is \$3-(-9)=12\$.
- **Therefore, the answer is \$\\boxed{12}\$.**

Figure 1: An end-to-end workflow of MATHSENSEI on the compositional setting from the MATH dataset. The final answer is highlighted in green font (c_i and o_i are defined in Section 2).

existing research works have not delved into individual tool study, both in isolation and in combinations with other tools. This motivated us to explore the potential of *TALM* frameworks in the context of complex mathematical reasoning tasks. In this paper, we propose and develop MATHSENSEI, a *TALM*-based framework, comprising a distinct set of tools (also referred to as modules), combined in a sequential fashion (Fig. 1). These modules include LLM-based components, such as - knowledge retriever (KR), python code generator (PG), code refiner (CR), and solution generator (SG); and APIs, such as - Bing-Web-Search-API (BS) and Wolfram-Alpha-API (WA). After conducting extensive ablations, varying the set and order of modules in our framework, we observe that complex mathematical problems, spanning different subdomains can be benefited by specific types, combinations, and *order* of these modules. Our best configuration of MATHSENSEI, PG + WA + SG (+ +) achieves an impressive performance accuracy of 47.6 % on the MATH (Hendrycks et al., 2021b) dataset, surpassing GPT-3.5-TURBO() with Chain-of-Thought (CoT) prompting by 13.5% (Chen et al., 2023a). The same setting shows a performance gain of +11.6% over GPT-4 (with CoT prompting) on Intermediate Algebra problems. For Precalculus, GPT-4 (with CoT prompting) has an accuracy of 26.7%, which gets improved to 28.9% by our WA + PG + SG (+ +) setting. Improvements on AQuA-RAT (Ling et al., 2017) and MMLU-Math (Hendrycks et al., 2021a) are lower, 2.4% and 3.3% respectively, showing that the efficacy decreases as requirement of external knowledge decreases. This further highlights the need for effective planning strategies. We evaluate two advanced planning techniques (Plan-And-Solve (Lu et al., 2023a) and REACT (Yao et al., 2022)) with MATHSENSEI and do not observe benefit of using them over our best configuration. Our contributions are as follows: (a) we comprehensively evaluate the effectiveness of TALM frameworks across multiple mathematical datasets, such as GSM-8K, AQUA-RAT, MATH, MMLU-Math, encompassing diverse mathematical problem types and tasks, (b) through systematic ablations by varying the set and order of modules in our framework, we observe that complex mathematical problems spanning different domains (such as, algebra, calculus, number theory, and probability from the MATH dataset) can be benefited by certain types, combinations, and order of these modules, and (c) we quantify the performance of state-of-the-art planning techniques, such as Plan-And-Solve and REACT coupled with tool-augmented LLMs on the MATH dataset.

2 METHOD

Problem Formulation. Given an input mathematical query $q \in Q$, the objective is to provide the final correct answer a by executing the set of relevant modules. Let $[m_1, \ldots, m_t]$, be the ordered sequence of chosen modules for answering q, and $[o_1, \ldots, o_t]$ be the output sequence of the t modules. Let, s_i, f_i, and c_i denote the instruction, in-context example(s), and context, respectively, that we use for module m_i. The input prompt p_i, corresponding to module m_i is defined as:

$$p_i = \langle s_i; f_i; c_i \rangle \tag{1}$$

where context c_i is defined as:

$$c_i = \begin{cases} [q], \text{ if } i = 1; \\ [c_{i-1}; o_{i-1}], \text{ for } i = 2, \ldots, t \end{cases} \tag{2}$$

Here, $x; y$ denotes concatenation of x and y.

Datasets. We perform evaluations on MATH (Hendrycks et al., 2021b), GSM-8K (Cobbe et al., 2021), AQUA-RAT (Ling et al., 2017), and MMLU-Math (Hendrycks et al., 2021a), with special focus on MATH, since it contains a diverse set of subject types (*Precalculus, Prealgebra, Algebra, Geometry, Intermediate Algebra, Counting and Probability, and Number Theory*) and levels of difficulty(1-5). GSM-8K consists of basic arithmetic operations with linguistically diverse math word problems while AQUA-RAT is a MCQ format task spanning algebra. MMLU-Math has problems in *abstract algebra, elementary mathematics, high-school mathematics, college mathematics*, and *formal logic*.

Modules. For our experiments, we use a diverse set of modules: (1) LLM-based Knowledge Retriever (KR), (2) Bing Web Search (BS), (3) Wolfram Alpha (WA), (4) Python Generator + Executor (PG) using *Sympy* library, (5) Code Refiner (CR), and (6) LLM-based Solution Generator (SG). Overview of the modules are presented in A.2. We also provide illustrative examples of each module in A.3. We assign GPT-3.5-TURBO (🤖) as the default LLM in all LLM-based modules unless mentioned otherwise. This is mainly because it is more accessible and cheaper, compared to *GPT-4*. For querying a search-engine, we use `Bing-Web-Search-API`. Please refer to Table 24 for more details about the online resources that we use in this work.

3 Experiments and Results

Advantages of Bing Search over LLM-based knowledge retrieval. As shown in Table 4, the Bing Search (BS 🌐) module significantly outperforms the KR module. The extra knowledge retrieved by the KR module is useful only for problems in Algebra, PreAlgebra, and Probability domains. Moreover, the overall accuracy drops steadily as we change KR's LLM from GPT-3.5-TURBO to other variants (Table 7). This indicates that generic LLMs are not equipped with mathematical concepts of other domains. After analyzing different LLM variants for the KR module, we find that the knowledge retrieved by weaker LLMs heavily degrades performance of the downstream SG module. This motivated us to explore the impact of search engine-based knowledge retrieval. In Table 5, we observe that BS + SG setting is a clear winner over the SG setting, when GPT-3.5-TURBO is used for generating the `BING-Web-Search-API` query and getting final solution from SG. This holds true even when the LLM used in both modules is varied between TEXT-DAVINCI-003 (+4.6%) and GPT-3.5-TURBO (+4.2%). Thus, augmenting LLMs with knowledge (relevant to a mathematical question) retrieved from the web proves to be more beneficial in improving problem solving capabilities, compared to LLM-generated knowledge. Please refer to Appendix A.6 and A.7 for more details.

Program generation (PG) with Sympy. Recent research works such as PAL (Gao et al., 2023) and Program of Thoughts (PoT) (Chen et al., 2022) have showcased the benefits of using python code representations to solve mathematical problems, by offloading execution of generated code to an external python interpreter. We observe that the PG + SG setting using the `sympy` library can improve upon the performance accuracy of SG on the MATH dataset by a substantial margin of 10.1%. We find that a majority of problems in MATH require complex computations such as solving equations, representation of complex mathematical objects such as vectors, solving problems in Geometry, some of which are hurdles for the Solution generator module since text representations alone fail to capture such complexities. Libraries such as `sympy` in our implementation, on the other hand, has support for symbolically representing such objects using well-defined functions, classes, methods, and sub-packages, which greatly improves the performance. We present additional insights into the strengths and limitations of the PG module in Appendix A.9.1 and A.9.2, respectively.

QUESTION: Determine the number of ways to arrange the letters of the word NINE. GOLD ANSWER: $\boxed{12}$			
SG (●):	**PG+SG (●+●)**	**BS+SG (●+●)**	**WA+SG (●+●)**

There are 4 distinct letters in the word NINE. The number of ways to arrange 4 distinct letters is $4! = 24$. Therefore, the answer is $\boxed{24}$.	`from sympy import *`
	 `word = 'NINE'`
	 `numperm=factorial(len(word));`
	 `print(numperm);`
	 ...
	 The word NINE has 4 letters.The number of ways to arrange 4 distinct letters is $4! = 24$. Therefore, the answer is $\boxed{24}$.
		Retrieved Example: In the word "BANANA", there are 6 letters but only 3 distinct letters . . . repeated letters.
		 Retrieved Formula: nPr = n!/(n1! n2! ... nr!) . . . Therefore, the answer is 4!/ (2! 1! 1!) = $\boxed{12}$.
			We call the Permutations function on the word NINE to find all possible arrangements of the letters. The answer from WolframAlpha is EINN, ENIN, . . . , NINE. Therefore, the answer is $\boxed{12}$.

SG module misses the repeated characters in NINE.	*PG module ignores the repeated characters.*	*BS module retrieves similar examples and relevant formula from the Web.*	*WA module generates the correct query: Permutations[NINE] to solve the given question and final SG answer is grounded on WA output.*

Table 1: Qualitative Analysis of the Responses generated by different Settings for a given MATH example.

Wolfram Alpha outperforms PG on specific sub-disciplines. We compare the paradigm of program-guided solving (having Sympy) with the use of symbolic engines utilizing knowledge bases (Wolfram Alpha). We find from Table 5 that WA + SG outperforms the SG approach by 8.1%, when both WA and SG are powered by GPT-3.5-TURBO. Additionally, there is a consistent improvement in accuracy over SG (close to 11 %) when TEXT-DAVINCI-003 is used as the LLM instead of GPT-3.5-TURBO. This showcasses the meaningful positive impact of augmenting Wolfram Alpha's capabilities with chain of thought (CoT) style reasoning of LLMs. Even more interestingly, WA+SG closely follows PG+SG performance on MATH, even outperforming it on specific sub-disciplines such as *Intermediate Algebra* by a significant margin (10.4%). We find that the majority of errors in Intermediate Algebra arise from Python code execution errors (Table 18), clearly showcasing the inability of python code to represent complex math objects in this domain. In contrast, the WA module effectively interacts with the API using Wolfram code language to address these issues, resulting in substantial enhancements (example in Table 2). As shown in Fig. 6, the maximum utility of the WA module lies in subdomains of *Intermediate Algebra* and *Algebra*. We present a further deep dive into the strengths and weaknesses of WA in Appendix A.8.2 and A.8.3, respectively.

Effect of Multiple Module Combinations. We experiment with various module combinations on MATH, reported in Table 4. Our findings on the MATH dataset reveal that distinct module combinations exhibit specialized efficacy in addressing specific categories of mathematical problems. The PG+WA+SG setting outperforms PG+SG and WA+SG by 3% (*p-value = 0.013*) and 5% (*p-value = 0.0*) respectively, showcasing the potential for effective utilization of complementary skills and strengths offered by both modules. We used a *Z-test* with a left-tailed alternative hypothesis for the above. The maximum gain for the PG+WA+SG setting over PG+SG and WA+SG are achieved in the sub-domains of Number Theory (40.5 %, 37.8% to 49.1%) and Intermediate Algebra (21.1%, 31.5% to 35.0%). We observe that WA+PG+SG is the best performing setting on problems in Algebra and Pre-calculus, which shows that the order of the modules used also has a substantial impact on the performance. The PG+SG and the PG+CR+SG settings are most useful in the subdomains of Pre-algebra and Probability respectively. Table 6 presents an examination of the variations in accuracy among various settings as a function of the problem levels (1-5) in the MATH dataset. Our analysis reveals a consistent improvement of over 10% across all difficulty levels with diverse modular configurations.

We consider an example from the MATH dataset and present a qualitative analysis of the responses generated by different settings in Table 1. We observe that in this example, SG and PG+SG are unable to capture the fine-grained nuances in the input question (repetition of characters in the word "NINE"), leading to reasoning errors. On the other hand, the BS+SG and WA+SG settings avoid committing such errors. Further insights into the distribution of math problems where our best configuration (PG+WA+SG) is most useful, is presented in Appendix A.10. These insights on the complementary benefits provided by each module, motivated us to evaluate state-of-the-art planning techniques with MATHSENSEI, which we present in Appendix A.11.

QUESTION: Let a, b, c be real numbers such that $a + b + c = 1$. Find the minimum value of $2a^2 + 3b^2 + 6c^2$.	
GOLD ANSWER $\boxed{1}$	
PG+SG (+)	**WA+SG (+)**

`from sympy import *`
`a, b, c = symbols('a b c')`
`expr = 2*a**2 + 3*b**2 + 6*c**2`
`constraint = Eq(a + b + c, 1)`
`solved = solve(constraint, c)`
`exprsub = expr.subs(c, solved[0])`
`deriv = diff(exprsub, a)`
`criticalpoint = solve(deriv, a)`
`minvalue = exprsub.subs(a, criticalpoint[0])`
	WA() Query Generator:
	
	Minimize[$2a^2 + 3b^2 + 6c^2, a + b + c == 1$]
	[Wolfram Alpha code provides simple way of representing question.]

Python () Output: **Minimum value:** $3b^2+6(1/4-b/4)^2+2(3/4-3b/4)^2$ *[PG module unable to get min value as an integer]*	WA() Output: Yes, the answer is: min($2a^2 + 3b^2 + 6c^2 \| a + b + c = 1$ = 1) at (a, b, c) = (1/2, 1/3, 1/6)
SG () Output: We can use the method of Lagrange multipliers to solve this problem ...Therefore, the answer is $\boxed{\frac{13}{6}}$ *[SG module does not use PG Output]*	SG() Output We call the Wolfram Alpha API to minimize $2a^2 + 3b^2 + 6c^2$ subject to $a + b + c = 1$. The answer from Wolfram Alpha is 1. Therefore, the answer is $\boxed{1}$ *[SG module uses WA output to give correct answer]*

Table 2: Example Demonstration of advantages of WA() over PG() in Int.Algebra

Effectiveness of MathSensei on MMLU-Math. Results in Table 8 reveal that the BS+PG+SG configuration enhances the accuracy of the SG setting by 3.3%. As the performance gain is low, we further perform a type wise analysis in Table 9. We observe that, other than Formal Logic (FL), adding different modules show substantial improvements in different types, such as 17% in College Math, 11.7% in High School Math, 7.5% in Elementary Math. Our analysis in MMLU-Math further supports the complimentary benefit of the tools used in MATHSENSEI framework for various mathematical types.

Decreased Efficacy on GSM-8K, AQUA datasets. From Table 8, we observe marginal improvements of using multiple modules on AQUA-RAT and GSM-8k, over the standalone SG module. Both datasets contain simpler algebraic and arithmetic word problems. GSM-8K consists of problems requiring simple arithmetic operations such as addition, subtraction, etc. and its complexity stems from linguistic diversity. The addition of the PG module (with Sympy) has a negative effect on the SG module on GSM-8K . By simply removing the *Sympy* library, we achieve accuracy gains of 2-3%. However, the SG module still outperforms PG+SG for GPT-3.5-TURBO. For GEMINI-PRO as well, we observe minimal improvements by incorporating PG on GSM-8K and AQUA as shown in Tables 10 and 11 ,respectively. Our observations align with previous research work such as Program of Thought (PoT) (Chen et al., 2022), which showed program-guided solving to be useful for solving more challenging classes of problems (such as, polynomial equations, combinatorics, symbolic questions) compared to simple arithmetic questions. Similar to GSM-8K, AQUA-RAT primarily focuses on problems that require generic language-based reasoning skills. We find that settings with tools mostly hurt the performance compared to SG. This is attributed to the fact that WA and BS are unnecessary for addressing straightforward problems, and invoking them often introduces noisy and irrelevant information into the context of SG.

4 CONCLUSION

We introduce a tool-augmented Large Language Model (TALM) framework, aka MATHSENSEI, targeted for mathematical reasoning. We utilize tools for web-search based knowledge retrieval, program generation, and API-based problem solving (Wolfram Alpha). We perform extensive ablations over the individual tools, along with varying the order and combinations on complex mathematical reasoning datasets (such as MATH). Our best configuration achieves a 13.5% improvement over GPT-3.5-TURBO (with CoT prompting) on MATH. Our experiments with planning strategies does not improve over our best configuration. We also observe that benefit of mathematical *TALM*s are minimal for simpler math word problems (in GSM-8k) and its benefit progressively increases as the required complexity and knowledge requirement for the problem increases through AQuA, MMLU-Math.

REFERENCES

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Jiaao Chen, Xiaoman Pan, Dian Yu, Kaiqiang Song, Xiaoyang Wang, Dong Yu, and Jianshu Chen. Skills-in-context prompting: Unlocking compositionality in large language models, 2023a.

Justin Chih-Yao Chen, Swarnadeep Saha, and Mohit Bansal. Reconcile: Round-table conference improves reasoning via consensus among diverse llms, 2023b.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning, 2023.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023.

Yiduo Guo, Yaobo Liang, Chenfei Wu, Wenshan Wu, Dongyan Zhao, and Nan Duan. Learning to program with natural language, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021b.

Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey, 2023.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597, 2015. doi: 10.1162/tacl_a_00160. URL https://aclanthology.org/Q15-1042.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging divergent thinking in large language models through multi-agent debate, 2023.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation : Learning to solve and explain algebraic word problems, 2017.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*, 2023a.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. A survey of deep learning for mathematical reasoning, 2023b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023.

Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing english math word problem solvers, 2021.

OpenAI. Gpt-4 technical report, 2023.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

Aarohi Srivastava, Abhinav Rastogi, and et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2023.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification, 2018.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Yuanzhen Xie, Tao Xie, Mingxiong Lin, WenTao Wei, Chenglin Li, Beibei Kong, Lei Chen, Chengxiang Zhuo, Bo Hu, and Zang Li. Olagpt: Empowering llms with human-like problem-solving abilities, 2023.

Runzhe Yang and Karthik Narasimhan. The Socratic Method for Self-Discovery in Large Language Models. https://princeton-nlp.github.io/SocraticAI/, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. Progressive-hint prompting improves reasoning in large language models, 2023.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification, 2023.

Table of Contents

A APPENDIX

A.1 RELATED WORK

Prompting Techniques. Large Language Models (LLMs) employing prompting strategies such as Chain-of-Thought (CoT) (Wei et al., 2022) and Program-of-Thought (POT) (Chen et al., 2022) have demonstrated commendable performance on simple mathematical datasets such as GSM-8K (Cobbe et al., 2021). However, their efficacy diminishes for datasets requiring complex computations and advanced mathematical knowledge. For instance, on the MATH dataset, GPT-4 with CoT prompting exhibits a notably low accuracy of 42%. Several variations of these strategies have been explored to improve accuracy in reasoning tasks. (Madaan et al., 2023) proposed *self-refine* that involves iteratively refining the initial output by utilizing feedback from the same model. (Zhou et al., 2023) employs code-based self-verification, by utilizing python code to check simple constraints that the LLM generated output should satisfy and correcting the output if necessary. Similarly, Progressive-Hint-Prompting (Zheng et al., 2023) involves multiple turns of interactions, using previously generated answers as hints for subsequent turns.

Similar to POT prompting, PAL (Program Aided language models) (Gao et al., 2023) adopts a program-guided solving paradigm. It reads natural language problems, generates programs as intermediate reasoning steps, and delegates the solution step to a runtime environment, such as the Python interpreter. Across 13 natural language reasoning tasks within Big-Bench-Hard (Suzgun et al., 2022), they observe that program-guided solving consistently outperforms significantly larger models. In our Tool-augmented framework (MATHSENSEI), we incorporate several such techniques. We adopt CoT prompting for the text generation modules, and use the methodology by (Gao et al., 2023) to generate python code (using libraries like `sympy`) based on the current context and mathematical question; followed by execution of the code using python interpreter.

While (Gao et al., 2023) focuses on elementary level MWP (Math Word problems) and simple arithmetic datasets such as ASDIV (Miao et al., 2021) and SingleEQ (Koncel-Kedziorski et al., 2015), we explore complex mathematical datasets spanning diverse math problem types (MATH, AQUA (Ling et al., 2017), MMLU-Math). Following *self-refine*, we employ a code refinement module to iteratively rectify syntactical errors in the original generated code, using error messages from the interpreter.

Tool-Augmented LLMs. The emerging trend of tool-augmented LLMs has garnered increasing attention within the research community. Large language models, trained on the objective of next-token prediction, excel at generating tokens based on probabilistic patterns in their training data, making them effective in data-intensive tasks. However, their proficiency falls short in capturing nuanced reasoning or token relationships, particularly in domains like mathematics. Consequently, there are instances or specific question types where it would be advantageous for an LLM to leverage support from specialized tools or modules. For instance, consider a question requiring the solution to the roots of a 4th-degree polynomial. The LLM, upon generating a special token followed by a query, can pause its generation and invoke a mathematics knowledge-base like Wolfram Alpha. Wolfram Alpha, in turn, can utilize its API to process the query and return the answer to the LLM, which can then continue its generation.

Toolformer (Schick et al., 2023) leverages data annotated with such tool calls (using special tokens for tools) and responses to train language models to employ tools as needed in a self-supervised manner. Similarly, the tool-augmented LLM framework CHAMELEON (Lu et al., 2023a) adopts a plug-and-play approach to utilize tools sequentially. In their setup, the sequence of execution of the tools is predetermined based on a target task; the output of each tool is added to the context for subsequent downstream tools in the pipeline. They perform evaluation on multi-modal knowledge-intensive datasets like ScienceQA, TabMWP. Similarly, frameworks such as ART (Paranjape et al., 2023) engage in multi-step reasoning, where each step is linked to a tool call. Utilizing search and code tools, ART tackles various tasks across datasets such as MMLU (Hendrycks et al., 2021a)and BigBench (Srivastava et al., 2023).

Our work adopts the generic backbone of popular tool-augmented LLM frameworks such as Toolformer and CHAMELEON. In comparison to the previous work, we distinguish ourselves by conducting a comprehensive analysis and comparison specific to tools useful for addressing diverse mathematical problems. Notably, CHAMELEON lacks evaluation on mathematical datasets, and

	Modules					
Models/APIs	**KR**	**BS**	**WA**	**PG**	**CR**	**SG**
`Bing-Web-Search-API`	✗	✓	✗	✗	✗	✗
`Wolfram-Alpha-API`	✗	✗	✓	✗	✗	✗
`Llama-2-7B`	✓	✗	✗	✗	✗	✗
`Phind-CodeLlama-34B-V2`	✗	✗	✗	✓	✗	✗
`text-davinci-002`	✓	✗	✗	✗	✗	✗
`text-davinci-003`	✓	✓	✓	✗	✗	✓
`gpt-3.5-turbo`	✓	✓	✓	✓	✓	✓

Table 3: Module Inventory.

ART focuses exclusively on algebra, leading to gaps in the assessment of tool-augmented LLMs. Furthermore, our study incorporates a comparison of planning techniques within tool-augmented LLM frameworks for mathematical reasoning, an aspect not adequately addressed in the current literature. To the best of our knowledge, planning techniques like REACT (Yao et al., 2022) have primarily been tested on knowledge-intensive reasoning datasets such as FEVER (Thorne et al., 2018) and HotpotQA.

A.2 Module Inventory

In this section, we present a brief overview of the tools or modules that we use in our study. We show the list of model/api used for each module in Table 3. A detailed description of the prompts used in each module is presented in A.12.

- **LLM-based Knowledge Retriever (KR) -** For this module, we design a prompt to extract relevant knowledge from a pre-trained LLM (taking any one from the list of models mentioned in Table 3) in the form of concepts, formulas, mathematical expressions, theorems, definitions, and hints on how to solve a corresponding mathematical question. An example prompt and output is shown in Fig. 2.

- **Bing Web Search (BS) -** This module queries the `Bing-Web-Search-API` to extract the most relevant snippets which may contain similar questions and concepts required for solving a mathematical problem. For similar questions search, we directly query the `API` with a mathematical question. In case of concepts search, we first use an LLM (either gpt-3.5-turbo or text-davinci-003) to generate a query corresponding to the input question, and then call the `API` to retrieve relevant concepts (refer to Fig. 3 for an example).

- **Wolfram Alpha (WA) -** This module (comprising multiple components) calls the `Wolfram-Alpha-API` using a query in the Wolfram language, retrieving the mathematical information from this knowledge base and utilizing the capabilities of its computation engine. First we employ an LLM to generate contextualized thoughts. Subsequently, based on the generated thought, the next component formulates a Wolfram code language query (referred to as the *"Final Query"*). On passing this query as input to the Wolfram Alpha API, we get a `JSON` dictionary object. We extract all the useful information from this dictionary (using an LLM-based extractor) and add it to the context of next module. An overview of the WA module is presented in Fig. 4.

- **Python Generator+Executor (PG) -** We use an LLM that takes as input the current context as a part of a well-structured prompt (shown in Appendix Fig. 5). The LLM is explicitly instructed to use the `sympy` library for accessing a set of mathematical operations and data structures required. Based on the prompt, the module generates an (executable) Python code, which on execution returns some output(s) or an error message. We handle syntax errors using two setups:

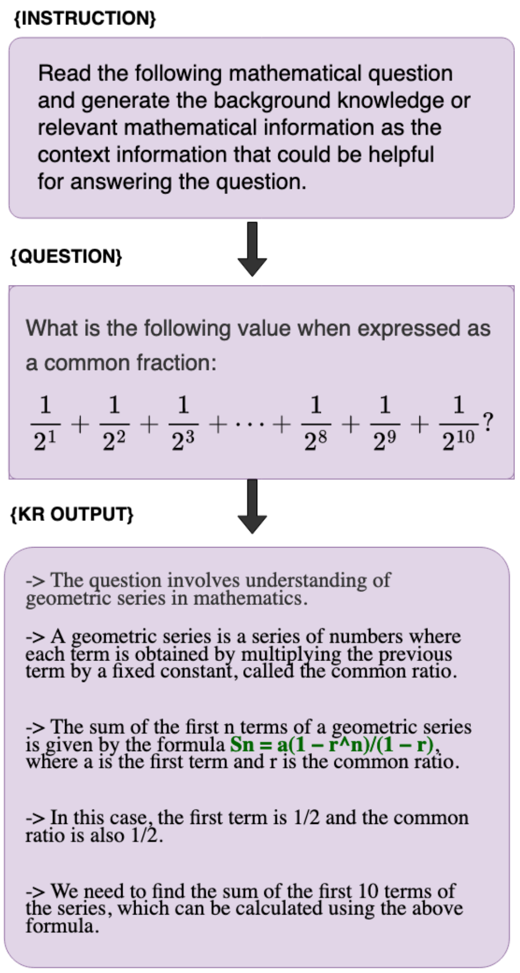
  - **Without refiner**: Here, if generated code produces syntax errors, we omit the output of PG from the context for next module.
  - **Code-Refiner** (CR): Here, we feed the error message along with the incorrect program to a code-fixing LLM which then generates a corrected python code and rationales of fixed errors given as "Errors fixed". We also add the information of common errors from our

qualitative analysis in the system prompt to aid the code refinement process. An output for the code refinement setup from the MATH dataset is presented in Fig. 5.

• **Solution Generator (SG) -** The solution generator is the final module in all settings. It takes the output from the pipeline and compiles a step-by-step solution based on all the context of previous modules. The final step is prompted to produce the answer of the question. It outputs the final answer enclosed within $\\boxed{}$ (similar to the MATH dataset).

A.3 Overview of Modules

A.3.1 Overview of LLM-based knowledge retriever



{INSTRUCTION}

Read the following mathematical question and generate the background knowledge or relevant mathematical information as the context information that could be helpful for answering the question.

{QUESTION}

What is the following value when expressed as a common fraction:

$$\frac{1}{2^1}+\frac{1}{2^2}+\frac{1}{2^3}+\cdots+\frac{1}{2^8}+\frac{1}{2^9}+\frac{1}{2^{10}}?$$

{KR OUTPUT}

-> The question involves understanding of geometric series in mathematics.

-> A geometric series is a series of numbers where each term is obtained by multiplying the previous term by a fixed constant, called the common ratio.

-> The sum of the first n terms of a geometric series is given by the formula **Sn = a(1 – r^n)/(1 – r)**, where a is the first term and r is the common ratio.

-> In this case, the first term is 1/2 and the common ratio is also 1/2.

-> We need to find the sum of the first 10 terms of the series, which can be calculated using the above formula.

Figure 2: Overview of the LLM-based Knowledge Retriever (KR) module.The text in green is a relevant knowledge generated by the LLM.

A.3.2 Overview of Bing Search Module

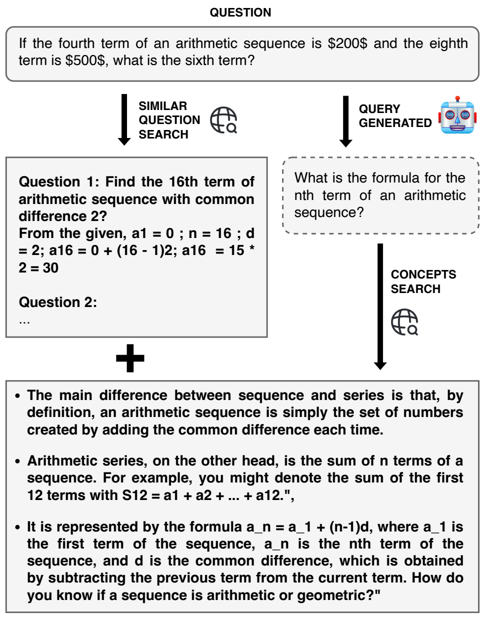

QUESTION
If the fourth term of an arithmetic sequence is 200 and the eighth term is 500, what is the sixth term?
SIMILAR QUESTION SEARCH
QUERY GENERATED
Question 1: Find the 16th term of arithmetic sequence with common difference 2?
From the given, a1 = 0 ; n = 16 ; d = 2; a16 = 0 + (16 - 1)2; a16 = 15 * 2 = 30
Question 2:
...
What is the formula for the nth term of an arithmetic sequence?
CONCEPTS SEARCH
• The main difference between sequence and series is that, by definition, an arithmetic sequence is simply the set of numbers created by adding the common difference each time.
• Arithmetic series, on the other head, is the sum of n terms of a sequence. For example, you might denote the sum of the first 12 terms with S12 = a1 + a2 + ... + a12.",
• It is represented by the formula a_n = a_1 + (n-1)d, where a_1 is the first term of the sequence, a_n is the nth term of the sequence, and d is the common difference, which is obtained by subtracting the previous term from the current term. How do you know if a sequence is arithmetic or geometric?"

Figure 3: Overview of the BS module; We concatenate the similar questions and concepts (from which relevant knowledge is retrieved by GPT-3.5-TURBO and then used as context by a downstream module).

A.3.3 Overview of Wolfram Alpha Module

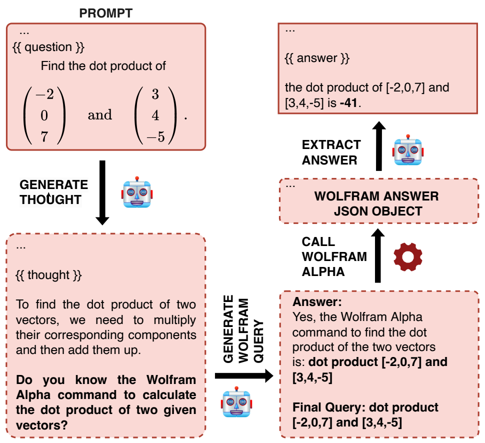
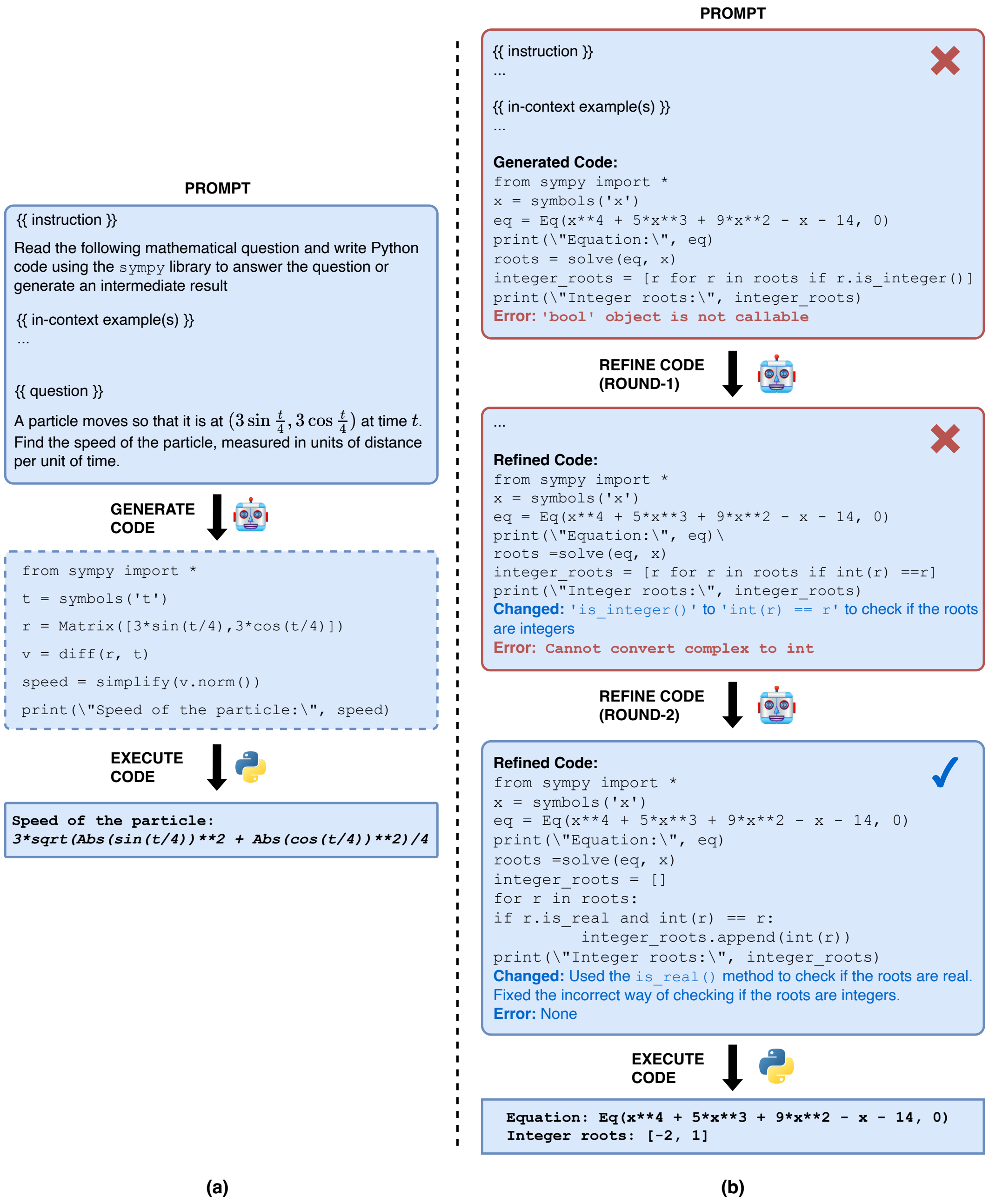

PROMPT
...
{{ question }}
Find the dot product of $\begin{pmatrix} -2 \\ 0 \\ 7 \end{pmatrix}$ and $\begin{pmatrix} 3 \\ 4 \\ -5 \end{pmatrix}$.
GENERATE THOUGHT
...
{{ thought }}
To find the dot product of two vectors, we need to multiply their corresponding components and then add them up.
Do you know the Wolfram Alpha command to calculate the dot product of two given vectors?
GENERATE WOLFRAM QUERY
Answer:
Yes, the Wolfram Alpha command to find the dot product of the two vectors is: dot product [-2,0,7] and [3,4,-5]
Final Query: dot product [-2,0,7] and [3,4,-5]
CALL WOLFRAM ALPHA
...
WOLFRAM ANSWER JSON OBJECT
EXTRACT ANSWER
...
{{ answer }}
the dot product of [-2,0,7] and [3,4,-5] is -41.

Figure 4: Overview of the WA module.

A.3.4 Overview of Python Generation Module



{{ instruction }}

Read the following mathematical question and write Python code using the `sympy` library to answer the question or generate an intermediate result

{{ in-context example(s) }}

...

{{ question }}

A particle moves so that it is at $\left(3 \sin \frac{t}{4}, 3 \cos \frac{t}{4}\right)$ at time t. Find the speed of the particle, measured in units of distance per unit of time.

```
from sympy import *
t = symbols('t')
r = Matrix([3*sin(t/4),3*cos(t/4)])
v = diff(r, t)
speed = simplify(v.norm())
print(\"Speed of the particle:\", speed)
```

```
Speed of the particle:
3*sqrt(Abs(sin(t/4))**2 + Abs(cos(t/4))**2)/4
```

{{ instruction }}

...

{{ in-context example(s) }}

...

Generated Code:

```
from sympy import *
x = symbols('x')
eq = Eq(x**4 + 5*x**3 + 9*x**2 - x - 14, 0)
print(\"Equation:\", eq)
roots = solve(eq, x)
integer_roots = [r for r in roots if r.is_integer()]
print(\"Integer roots:\", integer_roots)
```

Error: `'bool' object is not callable`

...

Refined Code:

```
from sympy import *
x = symbols('x')
eq = Eq(x**4 + 5*x**3 + 9*x**2 - x - 14, 0)
print(\"Equation:\", eq)\
roots =solve(eq, x)
integer_roots = [r for r in roots if int(r) ==r]
print(\"Integer roots:\", integer_roots)
```

Changed: `'is_integer()'` to `'int(r) == r'` to check if the roots are integers

Error: `Cannot convert complex to int`

Refined Code:

```
from sympy import *
x = symbols('x')
eq = Eq(x**4 + 5*x**3 + 9*x**2 - x - 14, 0)
print(\"Equation:\", eq)
roots =solve(eq, x)
integer_roots = []
for r in roots:
if r.is_real and int(r) == r:
         integer_roots.append(int(r))
print(\"Integer roots:\", integer_roots)
```

Changed: Used the `is_real()` method to check if the roots are real. Fixed the incorrect way of checking if the roots are integers.

Error: None

```
Equation: Eq(x**4 + 5*x**3 + 9*x**2 - x - 14, 0)
Integer roots: [-2, 1]
```

Figure 5: Overview of (a) Python Generator Module and (b) Code Refiner Module

A.4 Results Tables

Method	**Alg**	**P.Cal**	**P.Alg**	**Geom**	**Prob**	**N.Th**	**Int.Alg**	**O.Acc**
***Baselines with* `gpt-3.5-turbo` (🤖)**								
CoT-LTP (Guo et al., 2023)	49.6	16.3	52.3	22.5	30.2	29.8	16.9	31.1
ComplexCoT (Fu et al., 2023)	49.1	16.8	53.8	22.3	29.7	33.4	14.6	34.1
ComplexCoT+PHP (Zheng et al., 2023)	51.1	16.1	57.7	25.4	33.7	35.1	17.1	36.5
SKiC (Chen et al., 2023a)	57.9	23.0	62.0	30.1	38.2	35.5	17.8	40.6
***Baselines with* `GPT-4`**								
CoT (Zhou et al., 2023)	70.8	26.7	71.6	36.5	53.1	49.6	23.4	50.4
PHP (Zhou et al., 2023)	74.3	29.8	73.8	41.9	56.3	55.7	26.3	53.9
Ours								
SG (✅)	46.7	18.1	55.7	25.3	32.9	30.2	16.2	34.5
KR + SG (📖 + ✅)	49.1	15.0	58.0	24.4	34.3	29.6	12.0	34.4
BS + SG (🌐 + ✅)	51.6	20.1	63.3	27.1	36.1	39.6	16.3	38.7
PG + SG (🐍 + ✅)	60.0	26.5	66.1	30.7	42.1	40.5	21.1	44.6
PG + CR + SG (🐍 + ⚗️ + ✅)	60.0	26.5	66.1	30.7	48.3	43.0	26.9	45.6
PG′[🦙] + SG (🐍 + ✅)	55.4	23.5	58.0	22.9	32.7	42.2	17.9	39.6
WA + SG (⚙️ + ✅)	57.8	26.1	58.5	26.3	37.6	37.8	31.5	42.6
PG + BS + SG (🐍 + 🌐 +✅)	53.1	20.7	58.7	28.6	37.8	36.6	19.9	39.0
BS + PG + SG (🌐 + 🐍 +✅)	55.0	23.1	61.2	27.5	35.4	35.4	20.5	39.8
WA + PG + SG (⚙️ + 🐍 +✅)	62.5	28.9	61.5	27.1	42.6	45.7	33.4	46.3
PG + WA + SG (🐍 + ⚙️ +✅)	61.6	28.7	64.7	30.5	42.8	49.1	35.0	47.6
BS + WA + SG (🌐 + ⚙️ +✅)	56.2	22.9	61.0	29.8	37.5	44.0	28.9	42.9
WA + BS + SG (⚙️ + 🌐 + ✅)	60.0	27.0	65.0	29.0	40.5	42.2	31.4	45.4
BS + PG + WA + SG (🌐 + 🐍 + ⚙️ + ✅)	60.2	26.4	65.0	31.3	44.7	48.7	31.6	46.7

Table 4: Comparison of our Modular Settings to Published Baselines on **MATH**. We use GPT-3.5-TURBO (🤖) as the default LLM for each setting (except one row). For PG′[🦙] + SG (🐍 + ✅) setting, we use `Phind-CodeLlama-34B-V2` as the underlying LLM for the PG🐍 module (while keeping GPT-3.5-TURBO (🤖) as the default LLM for SG✅ module); We ran PG+SG and PG+CR+SG settings independently on 5000 examples of the MATH dataset. For each subject type corresponding to PG+CR+SG, we present the maximum accuracy between the two settings; Alg: Algebra, P.Cal: Precalculus, P.Alg: Prealgebra, Geom: Geometry, Prob: Probability, N.Th: Number Theory, Int.Alg: Intermediate Algebra; We have taken the first four baseline results from SKiC (Chen et al., 2023a), and following two baselines from (Zhou et al., 2023).

LLMs	**BS+SG**	**WA+SG**	**SG**
	(🌐 + ✅)	(⚙️ + ✅)	(✅)
(🤖 + 🤖)	38.7	42.6	-
(🤖 + 👾)	27.4	35.6	-
(👾 + 🤖)	30.0	37.8	-
(👾 + 👾)	20.8	27.0	-
(🤖)	-	-	34.5
(👾)	-	-	16.2

Table 5: Ablations of BS+SG (🌐 + ✅), WA+SG (⚙️ + ✅), and SG (✅) settings using different combination of LLMs, such as GPT-3.5-TURBO (🤖) and TEXT-DAVINCI-003 (👾) on the MATH dataset.

Setting	Level 1	Level 2	Level 3	Level 4	Level 5
SG (✅)	71.8	53.1	41.0	25.6	12.2
WA+SG (⚙+✅)	74.6	60.5	46.6	37.6	21.3
PG+SG (🐍 + ✅)	83.6	62.4	52.6	40.0	19.8
WA+PG+SG (⚙+🐍+✅)	76.4	61.5	54.0	40.2	25.2
PG+WA+SG (🐍+⚙+✅)	79.1	62.8	53.9	41.8	26.9
BS+WA+SG (🌐+⚙+✅)	74.6	59.3	51.0	35.6	21.0
WA+BS+SG (⚙+🌐+✅)	76.0	60.1	52.0	39.9	24.6
BS+PG+WA+SG (🌐+🐍+⚙+✅)	81.0	60.5	52.9	41.6	25.4

Table 6: Performance of different Settings across varying Levels of Complexity (1-5) on the MATH dataset.

Model	Ovr Acc
text-davinci-002 (🤖)	22.8
text-davinci-003 (🤖)	27.1
Llama2-7B (🦙)	28.4
gpt-3.5-turbo (🤖)	34.4

Table 7: Performance of different backbone models used for KR📖 module in the KR + SG (📖 + ✅) setting. For all settings, we use GPT-3.5-TURBO (🤖) as the default LLM for the SG✅ module.

Setting	GSM-8K*	AQUA	M.Math
(✅)	79.2	61.4	66.2
(📖+✅)	71.8	57.5	64.5
(⚙+✅)	65.6	57.9	66.0
(🌐+✅)	56.0	53.5	67.6
(🐍+✅)	77.0	55.1	68.1
(⚙+🐍+✅)	70.1	63.8	65.1
(🐍+⚙+✅)	76.2	62.6	67.1
(🐍+ 🌐+✅)	67.6	58.3	67.2
(🌐+🐍+✅)	69.2	56.3	69.5
(🌐+🐍+⚙+✅)	70.7	61.4	66.9

Table 8: Comparison of Multi-Module Settings for GSM-8K, AQUA-RAT and MMLU-Math (M.Math) datasets with GPT-3.5-TURBO as LLM.

* We report the results without explicitly using *Sympy* for the PG Module for **GSM-8K**. Our observations show a 2-3 percent improvement by simply removing *Sympy*.

Setting	FL	AA	EM	CM	HM
(✅)	53.9	49.0	84.6	41.0	57.7
(📖+✅)	50.6	43.9	84.8	38.6	58.5
(⚙+✅)	52.4	54.5	88.1	58.0	67.0
(🌐+✅)	40.5	44.4	80.1	49.0	63.0
(🐍+✅)	49.5	50.0	81.6	44.0	69.4
(⚙+🐍+✅)	44.7	36.1	81.4	57.1	63.7
(🐍+⚙+✅)	45.7	55.5	92.1	42.3	68.0
(🐍+ 🌐+✅)	50.0	47.0	81.2	44.0	59.1
(🌐+🐍+✅)	46.8	38.0	84.9	47.5	63.3
(🌐+🐍+⚙+✅)	41.3	43.0	79.3	45.0	66.1

Table 9: MMLU Accuracy vs type of problem; FL:Formal logic, AA: Abstract Algebra, EM: Elementary Mathematics, CM: College Mathematics, HM: High School Mathematics

LLM	PG+SG	SG
GPT-3.5-TURBO	77.0	79.2
GEMINI PRO	75.0	73.6

Table 10: Comparison of PG+SG(🐍+ ✅) and SG (✅) for two models GPT-3.5-TURBO and GEMINI PRO on GSM-8K.

LLM	PG+SG	SG
GPT-3.5-TURBO	55.1	61.4
GEMINI PRO	52.7	53.1

Table 11: Comparison of PG+SG(🐍+ ✅) and SG (✅) for two models GPT-3.5-TURBO and GEMINI PRO on AQUA.

A.5 Analysis of Tools

A.6 Strengths of Bing Search (BS) Module

Previous investigations of retrieval-augmented generation (RAG) and Self-RAG have shown how conditional generation using retrieval-based approaches improve factuality in knowledge-intensive tasks. The BS module retrieves useful information (such as, formulas, concepts, and similar questions) from the Web and improves the effectiveness of the downstream SG module by adding relevant snippets to its context. We observe that simple retrieval based methods can also be equally beneficial in the domain of mathematical reasoning (offering improvements in both MATH and MMLU-Math). BS + SG also outperforms the KR + SG setting (which is powered by LLM-based Knowledge Retrieval (KR) module) on both these datasets. An example, where BS + SG is correct and SG is incorrect is presented in Table 1.

A.7 Limitations of Bing Search (BS) Module

(1) Our simplistic BS Module implementation uses raw outputs returned by the Bing Web Search API v7 directly, which can be prone to noisy retrievals. Additionally, we do not employ any critique mechanism to check the relative importance of multiple pieces of retrieved information.

(2) The addition of the BS module to SG reduces the performance on GSM-8K significantly. Like WA module, this is due to the diminishing utility of BS on simpler problems. This calls for a component which can effectively decide when it is required to retrieve information and when it is not necessary (future research).

(3) We observe substantial signs of *data contamination* through the Web for the MATH test dataset. We analyze a random sample of **59** examples for which the *BS+SG* setting was correct. Our qualitative analysis, shows that for **37/59** examples, snippets of the *exact same question* were partially or fully recovered from the Web. For **27** of these **37** examples, the answers from the Web were directly used while for the remaining 10 examples, the retrieved content was ignored (due to partially recovered noisy snippets containing only the question part or incoherent segments). This observation poses serious questions towards the suitability of such state-of-the-art datasets as evaluation benchmarks for LLMs.

A.8 Wolfram Alpha

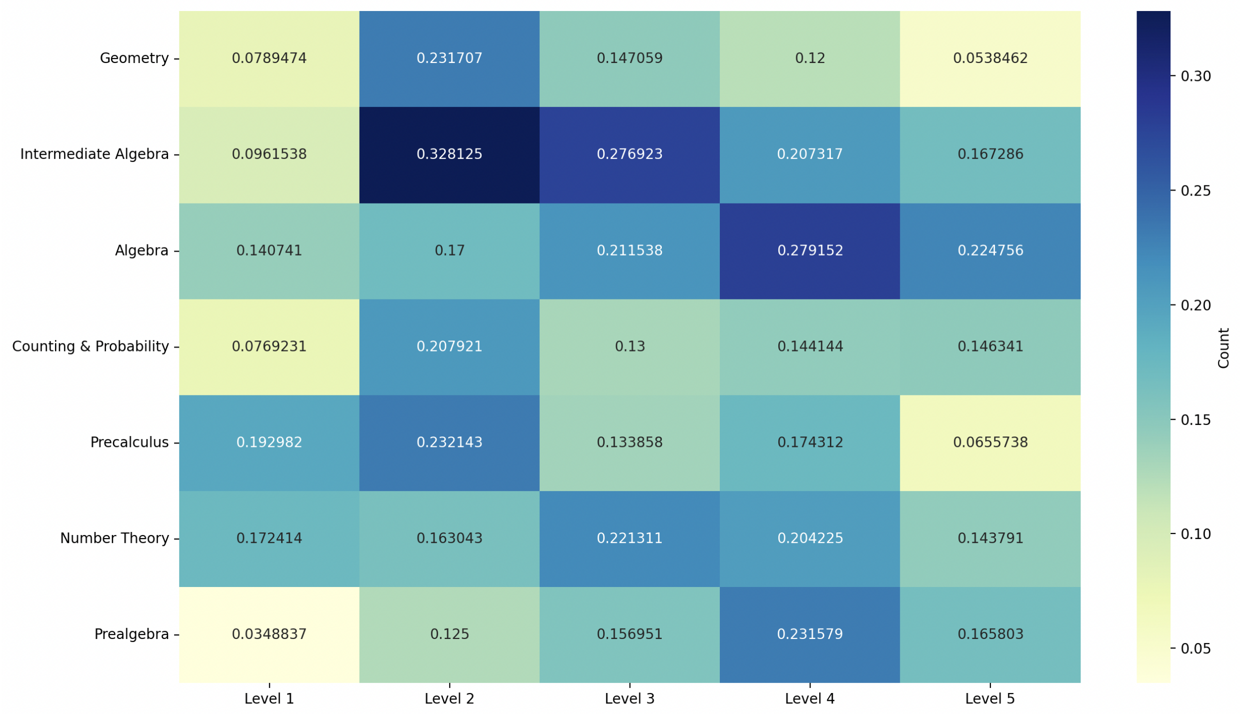


Figure 6: Distribution of examples where WA+SG (✿+✓) is correct and SG (✓) is wrong, across problem types and level of difficulty (1-5). The values are normalized by number of examples in each (subject - level) combination.

A.8.1 Example Wolfram Alpha commands

WA Query
$Sum[i^k, \{k, 1, 259\}]$
$Solve[\{725x + 727y == 1500, 729x + 731y == 1508\}, \{x, y\}]$
FactorInteger[3105]
$maximize[y/x, (x-3)^2 + (y-3)^2 == 6]$
period of $cos(bx)$

Table 12: Examples of generated WolframAlpha (WA✿) Queries.

A.8.2 Strengths of Wolfram Alpha Module

(1) The WA module overcomes the limitations of SG by constructing a query (in natural language/symbolic format) (Table 12), aiding the abstraction of the problem solving process when required by harnessing the computational intelligence of the Wolfram engine.

(2) In cases, where the query to the Wolfram Alpha API is syntactically and logically correct for solving a mathematical question, the returned answer is guaranteed to be correct, which is then processed by the SG module to compile the final answer.

(3) From Fig.6, we observe maximum benefit of Wolfram Alpha module for problems in the subdomains of Algebra (*Level 4*) and Intermediate Algebra (*Level 2 and 3*).

Some examples, demonstrating the benefits of Wolfram Alpha module are provided in Table 13.

A.8.3 Limitations of Wolfram Alpha

The limitations of the WA module (examples are shown in below Table) are mostly associated with:
(1) Logical errors in LLM-generated WA API queries (*Question 1* in Table 14).

(2) Wrong interpretation of correct WA response by the downstream SG module (*Question 2* in Table 14).

(3) Since WA module returns a single line answer (in many cases), the downstream SG module fails to provide a step-by-step reasoning (*Question 3* in Table 14).

QUESTION 1: Find the remainder when $x^5 + x^4 + x^3 + x^2 + x$ is divided by $x^3 - 4x$	**QUESTION 2:** Let P be the point $(0, 5)$, let Q be the point $(6, 9)$, and let R be the point $(12, 0)$. Determine the area of right-angled $\triangle PQR$.
GOLD ANSWER: $\boxed{5x^2 + 21x}$	**GOLD ANSWER:** $\boxed{39}$
Setting: SG (✔)	
Response: The result of the division is $x^2 + 5x + \frac{5x}{x^3-4x}$	**Response:** $QR = \sqrt{(12 - 6)^2 + (0 - 9)^2} = \sqrt{180}$
[*INCORRECT; Calculation error in intermediate step*]	[*INCORRECT; Error in distance calculation in intermediate step.*]
Setting: WA+SG (⚙+✔)	
WA API Query: PolynomialRemainder[$x^5 + x^4 + x^3 + x^2 + x, x^3 - 4x, x$] *[CORRECT]*	**WA API Query:** triangle((0,5),(6,9),(12,0)) [*CORRECT*]
WA Output: Yes, the answer is $5x^2 + 21x$ [*CORRECT*]	**WA Output:** Yes, the answer is: Triangle shape: right triangle; Edge lengths: $(2\sqrt{13}, 3\sqrt{13}, 13) \approx (7.211, 10.816, 13)$; - Area: 39; Perimeter: $13 + 5\sqrt{13} \approx 31.027$; Interior angles: $cos^{-1}(3/\sqrt{13})$ rad; $cos^{-1}(2/\sqrt{1})$ rad; $\pi/2$ rad $\approx$ (0.588 rad, 0.982 rad, 1.570 rad); Interior angle sum: $180° = \pi$ rad $\approx$ 3.142 rad. [*CORRECT*]
SG Output: The answer from Wolfram Alpha is found to be $5x^2 + 21x$. Therefore, the answer is $\boxed{5x^2 + 21x}$ [*CORRECT*]	**SG Output:** We call the Wolfram Alpha API to find the area of the triangle formed by the points $(0, 5), (6, 9)$, and $(12, 0)$; The answer from Wolfram Alpha is found to be 39; Therefore, the answer is $\boxed{39}$ [*CORRECT*]

Table 13: Utility of the WA module in WA+SG (⚙+✔) setting.

Setting: WA+SG (⚙+✔)		
QUESTION 1: The positive integers up to 2007 are alternately subtracted and added: $[1-2+3-4+\cdots+2001-2002+2003-2004+2005-2006+2007.]$ What is the value of the expression? **GOLD ANSWER:** $\boxed{1004}$	**QUESTION 2:** When the expression $-2x^2-20x-53$ is written in the form $a(x+d)^2+e$, where a, d, and e are constants, what is the sum a+d+e? **GOLD ANSWER:** $\boxed{0}$	**QUESTION 3:** Solve $\frac{2x^2+x+3}{x^2+x+1}=\frac{2x+1}{x+1}$ for x. **GOLD ANSWER**: $\boxed{-2}$
WA API Query: Sum$[(-1)^n * n, \{n, 1, 2007\}]$ *[INCORRECT; since it finds negative of the required answer]* *CORRECT* Query : Sum$[(-1)^{(n+1)} * n, \{n, 1, 2007\}]$	**WA API Query:** CompleteSquare$[-2x^2-20x-53]$ [*CORRECT*]	**WA API Query:** solve $(2x^2+x+3)/(x^2+x+1)=(2x+1)/(x+1)$ for x [*CORRECT*]
WA Output: Yes, the answer is -1004 . *[Logical errors in LLM generated WA API queries].*	**WA Output:** Yes, the answer from this object is: $-2(x+5)^2-3$ [*CORRECT*]	**WA Output:** Yes, the answer is x = -2 [*Since WA module returns a single line answer (in many cases), the downstream SG module fails to provide a step-by-step reasoning.*]
SG Output: We use the Wolfram Alpha API to evaluate the expression Sum$[(-1)^n * n, n, 1, 2007]$. The answer from Wolfram Alpha is -1004. Therefore, the answer is $\boxed{-1004}$ [*INCORRECT*]	**SG Output:** From WolframAlpha response, we find $-2x^2-20x-53=-2(x+5)^2-3$.Thus we have $a=-2$, $d=-5$, and $e=-3$. So, $a+d+e=-2-5-3=-10$. Therefore, the answer is $\boxed{-10}$ [*INCORRECT; wrong interpretation of correct WA response by the downstream SG module. Value of d should be +5 and not -5.*]	**SG Output** We call the WolframAlpha API to solve the given equation.The answer from Wolfram Alpha is x = -2. Therefore, the answer is $\boxed{-2}$ [*CORRECT; but lacks reasoning chains*]

Table 14: Limitations of the WA Module in WA+SG (⚙+✔) setting.

A.9 Python Code Generation using Sympy

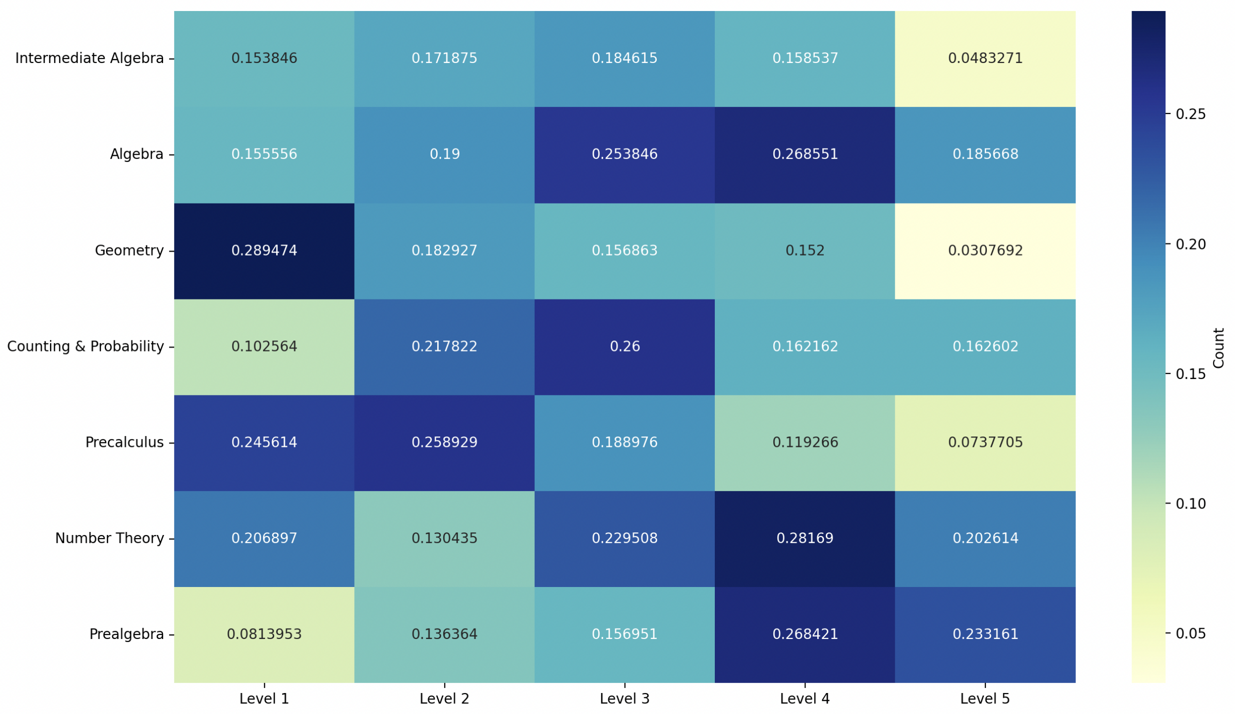


Figure 7: Distribution of examples where PG+SG (🐍+ ✓) is correct and SG (✓) is wrong, across problem types and level of difficulty (1-5). The values are normalized by number of examples in each (subject - level) combination.

A.9.1 Strengths of Python Code Generation Module

(1) For complex mathematical problems that are present in the MATH dataset, the *Sympy* library offers strong capabilities to the PG module. The benefits are caused by the ability of PG to symbolically represent the complex questions using Sympy objects, classes and methods, leveraging in-context learning with few-shot examples only.

(2) The deterministic program executor also helps in avoiding common errors committed by the SG module alone.

(3) The python code generation module (🐍) is useful for mathematical problems in multiple domains (*Geometry Lvl1, Number Theory, PreAlgebra, Algebra*), as shown in Fig.7.

The example presented in the Table.15 demonstrates how PG module helps in correcting the SG module's error.

QUESTION: The average of Amys, Ben, and Chris's ages is 6. Four years ago, Chris was the same age as Amy is now. In four years, Ben's age will be 35 of Amy's age at that time. How many years old is Chris now?

GOLD ANSWER: $\boxed{10}$

Setting: SG (✓)

Response:

$\frac{1}{3}(a+b+c)=6; \qquad c-4=a; \qquad b+4=\frac{3}{5}(a+4).$

Simplifying the third equation, we get $5b+20=3a+12$. Substituting $c=a+4$ into the first equation and simplifying, we get $a+b=10$.

[INCORRECT; Correct equation: 2a +b =14]

Setting: PG+SG (🐍+✓)

Response:

```
from sympy import *
a, b, c = symbols('a b c')
eq1 = Eq((a+b+c)/3, 6)
eq2 = Eq(c-4, a)
eq3 = Eq(b+4, Rational(3,5)*(a+4))
sol = solve((eq1, eq2, eq3), (a, b, c))
print("Chris's age is", sol[c], "years old")
```

[Leverages the classes **Rational**, **Eq** and method **solve** from Sympy]

PG Output: Chris's age is **10** years old

SG Output: Let Amy's age be a, Ben's age be b, and Chris's age be c. The average of their ages is $\frac{a+b+c}{3}=6$, Four years ago, Chris's age was c-4, which was the same as Amy's age at that time, so $c-4=a$, In four years, Ben's age will be $b+4$, which will be $\frac{3}{5}$ of Amy's age at that time, so $b+4=\frac{3}{5}(a+4)$, Solving the system of equations gives $a=6$, $b=2$, and $c=10$. Therefore, the answer is $\boxed{10}$ years old.

[CORRECT]

Table 15: Utility of the PG module in PG+SG (🐍+✓) setting.

QUESTION: Uriah's book bag is getting too heavy for him. He needs to remove 15 pounds from it. His comic books weigh 1/4 pound each and his toys weigh 1/2 pound each. If he removes 30 comic books, how many toys does he need to remove?

GOLD ANSWER: 30 comic books weigh 7.5 pounds because 30 x .25 = ¡¡30*.25=7.5¿¿7.5 He needs to remove 7.5 more pounds because 15 - 7.5 = 7.5 He needs to remove 15 toys because 7.5 / .5 = 15

Setting: SG ()

Response:

We know that Uriah needs to remove a total of 15 pounds from his bag. He removes 30 comic books, which weigh a total of 30*(1/4) = 7.5 pounds. So he still needs to remove 15 - 7.5 = 7.5 pounds.Let the number of toys he needs to remove be x.The weight of x toys is x*(1/2) = 0.5x pounds. We need to solve the equation 0.5x = 7.5 to find x. Solving for x, we get x = 15. Therefore, Uriah needs to remove 15 toys from his bag. The answer is 15. [*CORRECT*]

Setting: PG+SG (+)

Response:

```
from sympy import *
x = symbols('x')
eq = Eq((1/4)*(30-x) + (1/2)*x, 15)           [Wrong equation - reasoning error]
print("Equation:",eq)
sol = solve(eq, x)
print("Solution x:",sol)
ans = 30 - sol[0]
print("Toys to remove:",ans)
```

PG Output: Equation: Eq(0.25*x + 7.5, 15); Solution x: [30.0000000000000];
Toys to remove: 0;

SG Output: From Python code we get that Uriah needs to remove 0 toys.
The answer is 0. [*INCORRECT*]

Table 16: Demonstration of a Reasoning Error committed by the PG module in PG+SG (+) setting in a GSM-8K problem.

A.9.2 Limitations of Python Code Generation Module

(1) Errors in Intermediate Algebra and Prealgebra are mainly due to the inability of generated python code to express complex objects (syntax errors) and boundary cases, respectively. In Geometry and Precalculus, we found a large proportion of errors to be caused due to lack of understanding of plots/figures (expressed in latex format) accompanying the question. We present some examples of common syntactical errors that the PG module makes (on MATH) in Table 17. An example of a reasoning error by PG on GSM-8K dataset is presented in Table 16.

(2) We conduct a qualitative analysis of PG+SG (+) on 106 randomly sampled questions from MATH dataset presented in Table 18. We find that the majority of errors in Intermediate Algebra arise from python code execution errors, clearly showcasing the inability of python code to represent complex math objects in this domain. In contrast, the WA () module effectively interacts with

the API using symbolic Wolfram code language to address these issues, resulting in substantial enhancements. A similar study is also conducted for the AQUA and GSM-8K dataset as presented.

Error Type	**Error message**
Undefined Symbol	name 'x' is not defined
Incorrect handling of objects	'FiniteSet' object has no attribute 'subtract'
Undefined functions	name 'divisibleby' is not defined
Library use without import	sympy package not found

Table 17: Common Errors committed by the PG (🐍) Module.

Dataset	**Subject**	**PG-Exec-Err**	**PG-R-Err**	**SG-Err**	**Egs.**
MATH	Alg	8	5	2	15
	P.Cal	6	9	0	15
	P.Alg	4	11	0	15
	Geom	3	12	0	15
	Prob	8	6	1	15
	N.Th	6	7	3	16
	Int.Alg	14	0	1	15
	O.Cnt	51	48	7	106
GSM-8K	-	1	18	1	20
AQUA	-	1	6	13	20

Table 18: Summary of Error types with PG+SG (🐍+✅) setting on a random subset of 106 examples (MATH dataset); for GSM-8K and AQUA we consider 20 random examples, where the setting SG (✅) is correct; **PG-Exec-Err:** Code generated by PG🐍 module having syntactical errors; **PG-R-Err:** Executable python code (from PG🐍) having reasoning errors; **SG-Err:** Solution Generator (SG✅) alters correct output from PG🐍 to incorrect; Alg: Algebra, P.Cal: Precalculus, P.Alg: Prealgebra, Geom: Geometry, Prob: Probability, N.Th: Number Theory, Int.Alg: Intermediate Algebra, O.Cnt: Overall Count, Egs.: Examples. Here we report the absolute count of errors across different subjects.

A.10 Combining the strengths of WA and PG:

In the presented figure (Fig. 8), the combination of the diverse capabilities offered by each module, namely PG (Python Generator) and WA (Wolfram Alpha), yields substantial enhancements across all mathematical subdomains. Notably, the combined approach yields additional benefits, particularly in problems of higher difficulty levels. For example, we see a large rise for *Precalculus lvl 2* problems, compared to *PG+SG* and *WA+SG* settings.

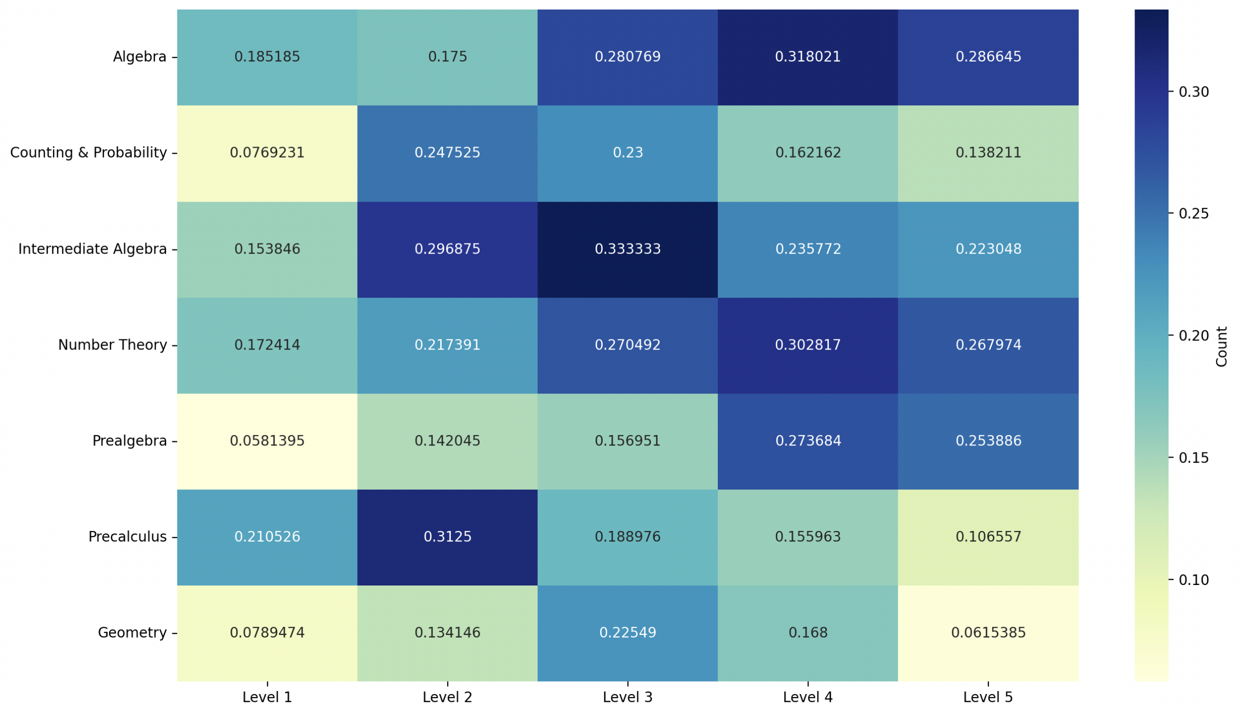


Figure 8: Distribution of examples where PG+WA+SG (🐍+⚙+ ✅) is correct and SG (✅) is wrong, across problem types and level of difficulty (1-5). The values are normalized by number of examples in each (subject - level) combination.

A.11 Planning Experiments

We explore two state-of-the-art planning strategies based following the Chameleon (Lu et al., 2023a) and the REACT (Yao et al., 2022) frameworks and report in in Table 19.

Plan-And-Solve Within the Plan-And-Solve (PAS) framework, a dynamic planner (LLM), generates a plan for a given mathematical problem before the start of execution. In our context, the plan consists of the sequence of modules to be run. Notably, this planning approach is inherently non-adaptive, as the strategy lacks the capability to determine the next module based on feedback and the output of the previously executed modules. To instruct the planner LLM, we provide input prompts containing information about each module, along with few-shot examples representing a possible sequence. The prompts utilized for the planner model are detailed in Table 23.

MathSensei with REACT Planner. The previous modular settings, have a fixed order of execution of the modules. However, we also wish to test out settings where there is power given to the central LLM to call different modules as and when required. This is done by executing (thought, action request, action execution) triplets. The thought serves as a summary of what we have till now in relation to answering the question, the action request is the specific action we wish to take in the next step, and the action execution step calls the necessary module from the modules library to execute the action. An overview of the REACT setting applied to the MATH dataset is presented in Fig. 9. The results for this setting corresponding to each problem type is presented in Table 19.

Results. We evaluate the performance of Plan-And-Solve and REACT on a randomly sampled subset of the MATH dataset of 3100 examples(for which REACT converges). The results show that simple vanilla implementation of the above planners is not sufficient for surpassing our best configuration PG+WA+SG. In particular, the majority of errors for REACT, were as a result of the failure of REACT to converge to a final solution (finish thought state). The variation of the accuracy as a function of the level of the problem shows, REACT* can surpass Plan-And-Solve (PAS) by a small percentage, however it still lags behind our best settings.

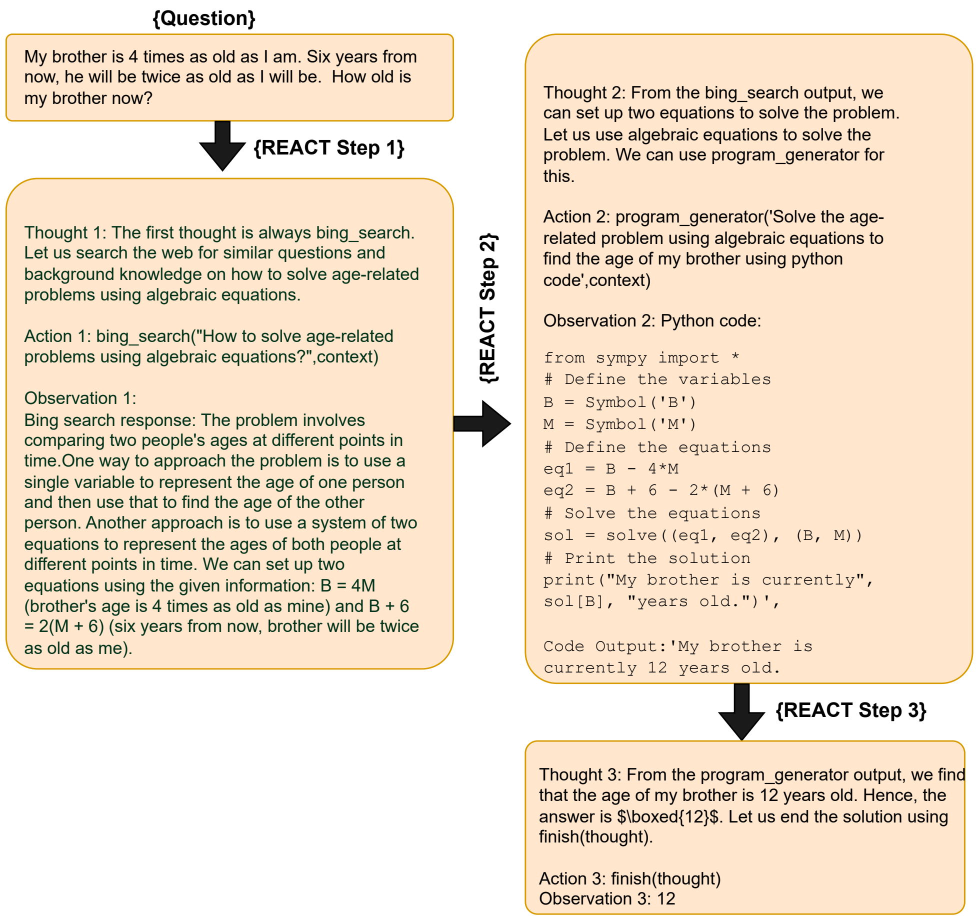
Plan Method	Alg	P.Cal	P.Alg	Geom	Prob	N.Th	Int.Alg	O.Acc
PAS*	57.3	29.8	65.0	32.4	42.0	47.7	31.9	47.3
REACT*	62.9	30.6	65.1	32.1	42.0	46.1	33.7	48.9
PG + WA + SG (🐍 + ⚙ + ✅)*	61.4	32.8	65.2	33.4	45.4	54.2	37.6	50.7
WA + PG + SG (⚙ + 🐍 + ✅)*	64.4	32.1	62.8	32.1	46.9	49.4	38.3	50.6

Table 19: Comparison of planning strategies: Plan-And-Solve (PAS) and REACT with two of our best performing settings on *3072* randomly sampled examples from the MATH dataset. Here X* denotes the use of *3072* samples for evaluating method X.

Setting	Level 1	Level 2	Level 3	Level 4	Level 5
PAS*	76.0	60.1	53.9	40.5	26.1
REACT*	78.3	62.0	55.4	41.6	27.9
PG + WA + SG (🐍 + ⚙ + ✅)*	79.3	65.3	54.3	43.9	31.1
WA + PG + SG (⚙ + 🐍 +✅)*	78.3	65.6	55.8	43.1	30.1

Table 20: Comparing Performance of different Planning Strategies (§A.11) with two of our Top Performing Settings by varying Difficulty Level of Problems from the MATH dataset.

Unlike planning in traditional closed world setup datasets such as Blocksworld, Logistics, Depot planning, etc., the task of planning in the mathematical reasoning domain presents multiple differences: Firstly, the set of possible actions is not finite as we can query each tool/module with any



My brother is 4 times as old as I am. Six years from now, he will be twice as old as I will be. How old is my brother now?

Thought 1: The first thought is always bing_search. Let us search the web for similar questions and background knowledge on how to solve age-related problems using algebraic equations.

Action 1: bing_search("How to solve age-related problems using algebraic equations?",context)

Observation 1:
Bing search response: The problem involves comparing two people's ages at different points in time.One way to approach the problem is to use a single variable to represent the age of one person and then use that to find the age of the other person. Another approach is to use a system of two equations to represent the ages of both people at different points in time. We can set up two equations using the given information: B = 4M (brother's age is 4 times as old as mine) and B + 6 = 2(M + 6) (six years from now, brother will be twice as old as me).

Thought 2: From the bing_search output, we can set up two equations to solve the problem. Let us use algebraic equations to solve the problem. We can use program_generator for this.

Action 2: program_generator('Solve the age-related problem using algebraic equations to find the age of my brother using python code',context)

Observation 2: Python code:

```
from sympy import *
# Define the variables
B = Symbol('B')
M = Symbol('M')
# Define the equations
eq1 = B - 4*M
eq2 = B + 6 - 2*(M + 6)
# Solve the equations
sol = solve((eq1, eq2), (B, M))
# Print the solution
print("My brother is currently",
sol[B], "years old.")',

Code Output:'My brother is
currently 12 years old.
```

Thought 3: From the program_generator output, we find that the age of my brother is 12 years old. Hence, the answer is $\boxed{12}$. Let us end the solution using finish(thought).

Action 3: finish(thought)
Observation 3: 12

Figure 9: Generated output for example from the MATH dataset for the REACT planning setting.

input string. Moreover, there are no preconditions that need to be satisfied for executing a particular action which makes the planning space much more unbounded. This can lead to long planning chains with (thought,action,execution) triplets where there may be multiple irrelevant actions. As seen from our work, the strengths and limitations of each tool also varies with the type of datasets, subdomains and difficulty levels, which makes the problem non-trivial. Hence, it turns out to be overwhelming to propose a novel planning strategy in this paper. We plan to explore this issue as a future research direction. A planner with a novel architecture and sufficient mathematical knowledge may be required to tackle this point.

A.12 Prompts

INSTRUCTION

Read the following question to generate thought, Query for searching Bing Web Search API, that will help to solve the entire problem or specific subproblems in the question.

IN-CONTEXT EXAMPLES

Question: When the expression $-2x^2 - 20x - 53$ is written in the form $a(x+d)^2 + e$, where a, d, and e are constants, what is the sum $a + d + e$?
Modules used till now: []
Mathematics Problem Type: Algebra
Level of Problem: Level 5
Thought: Since the question involves completing the square let us search how to complete the square.
Query: How do we complete the square of a quadratic equation?

Question: A circle is circumscribed about an equilateral triangle with side lengths of 6 units each. What is the area of the circle, in square units? Express your answer in terms of π.
Modules used till now: []
Mathematics Problem Type: Geometry
Level of Problem: Level 5
Thought: To solve this problem, we can search how to find the circumradius of an equilateral triangle.
Query: How to find the circumradius of an equilateral triangle?

... **[More few shot examples]**

INPUT QUESTION

Question: The point $P = (1, 2, 3)$ is reflected in the xy-plane, then its image Q is rotated by $180°$ about the x-axis to produce R, and finally, R is translated by 5 units in the positive-y direction to produce S. What are the coordinates of S?
Mathematics Problem Type: Precalculus
Level of Problem: Level 3

OUTPUT

Thought: To solve this problem, we can search the web for formulas related to reflection, rotation and translation.
Query: What are the formulas for reflection, rotation, and translation in 3D space?

Table 21: Bing Web Search Query generator Prompt for Concepts Search

INSTRUCTION

Read the following question to generate thought, answer and final query for searching Wolfram Alpha API, that is being used to solve the problem or specific subproblems.

IN-CONTEXT EXAMPLES

Question: When the expression $-2x^2 - 20x - 53$ is written in the form $a(x + d)^2 + e$, where a, d, and e are constants, what is the sum $a + d + e$?
Modules used till now: []
Mathematics Problem Type: Algebra
Level of Problem: Level 5
Thought: To find the answer to the problem, we need to rewrite the quadratic expression $-2x^2 - 20x - 53$ in the form $a(x+d)^2+e$, where a, d, and e are constants. Then, we find the sum a+d+e. Do you know the Wolfram Alpha command to do this?
Answer: Yes, the Wolfram Alpha command to factor the given quadratic expression is: factor($-2x^2 - 20x - 53$)
Final Query: factor($-2x^2 - 20x - 53$)

Question: Harold tosses a nickel four times. What is the probability that he gets at least as many heads as tails?
Modules used till now: []
Mathematics Problem Type: Counting and Probability
Level of Problem: Level 5
Thought: To find the probability that Harold gets at least as many heads as tails when tossing a nickel four times, we can use combinatorial techniques to count the favorable outcomes. However, to simplify the calculations, we can leverage Wolfram Alpha to compute the probability efficiently. Do you know the Wolfram Alpha command to find the probability for this problem?
Answer: Yes, the Wolfram Alpha command to find the probability of getting at least as many heads as tails when tossing a nickel four times is: $Binomial[4, k] * (1/2)^4$, where k ranges from 2 to 4 (inclusive).
Final Query: $Sum[Binomial[4, k] * (1/2)^4, k, 2, 4]$

... **[More few shot examples]**

INPUT QUESTION

Question: Find the constant c such that $(x^2 - 4x + 3)(x + 5) - (x^2 + 4x - 5)(x - c) = 0$ for all x.

OUTPUT

Thought: To find the constant c such that the given expression is equal to zero for all x, we can expand the expression and collect like terms. Do you know the Wolfram Alpha command to do this?

Answer: Yes, the Wolfram Alpha command to expand the given expression and set it equal to zero is: expand $((x^2 - 4x + 3)(x + 5) - (x^2 + 4x - 5)(x - c)) = 0$

Final Query: expand $((x^2 - 4x + 3)(x + 5) - (x^2 + 4x - 5)(x - c)) = 0$

Table 22: Wolfram Alpha API Query generator Prompt

INSTRUCTION
You need to act as a policy model, that given a question, determines the sequence of modules that can be executed sequentially to solve the question. The modules are defined as follows: - wolfram-alpha-search: Calls the Wolfram-Alpha API to solve the given question or intermediate steps in the question. - bing-search: Retrieves similar questions, background knowledge, useful information using the Bing Web Search API. - python-generator: This module generates an executable python program that can solve the given question using the Sympy library. - solution-generator: This module generates a detailed solution to the question based on the already generated context. Normally, solution-generator will incorporate the information from wolfram-alpha-search, bing-search, python-generator. It is always the last module to be executed.
IN-CONTEXT EXAMPLES
Question: Determine the number of ways to arrange the letters of the word ELEVEN. Modules: ['bing-search','solution-generator']
INPUT QUESTION
Question: If the numbers 4, 5 and 6 are each used exactly once to replace the letters in the expression $A(B - C)$, what is the least possible result?
OUTPUT
Modules: ['python-generator', 'solution-generator']

Table 23: Example of Planner Prompt and Output in Plan-And-Solve (PAS).

Resource	**URL**
open-source icons	https://iconduck.com/icons/
llama-2 icon	https://llama-2.ai/wp-content/uploads/2023/08/Llama-2-icon-150x150.png
codellama icon	https://codellama.dev/icons/black-transparentbg.png
python icon	https://s3.dualstack.us-east-2.amazonaws.com/pythondotorg-assets/media/community/logos/python-logo-only.png
azure openai service	https://azure.microsoft.com/en-us/products/ai-services/openai-service/
bing web search api service	https://www.microsoft.com/en-us/bing/apis/bing-web-search-api

Table 24: Online Resources