



# MATHSENSEI: A Tool-Augmented Large Language Model for Mathematical Reasoning

Anonymous ACL submission

## Abstract

001 Tool-augmented Large Language Models  
 002 (*TALM*) are known to enhance the skillset of  
 003 large language models (LLM), thereby, lead-  
 004 ing to their improved reasoning abilities across  
 005 many tasks. While, *TALMs* have been success-  
 006 fully employed in different question-answering  
 007 benchmarks, their efficacy on complex math-  
 008 ematical reasoning benchmarks, and, the po-  
 009 tential complimentary benefits offered by tools  
 010 for knowledge retrieval and mathematical equa-  
 011 tion solving, are open research questions. In  
 012 this work, we present MATHSENSEI, a tool-  
 013 augmented large language model for mathe-  
 014 matical reasoning. Augmented with tools for  
 015 knowledge retrieval (Bing Web Search), pro-  
 016 gram execution (Python), and symbolic equa-  
 017 tion solving (Wolfram-Alpha), we study the  
 018 complimentary benefits of these tools through  
 019 evaluations on mathematical reasoning datasets.  
 020 We perform exhaustive ablations on MATH,  
 021 a popular dataset for evaluating mathemati-  
 022 cal reasoning on diverse mathematical disci-  
 023 plines. We also conduct experiments involving  
 024 well-known tool planners to study the impact  
 025 of tool sequencing on the model performance.  
 026 MATHSENSEI achieves 13.5% better accuracy  
 027 over gpt-3.5-turbo with chain-of-thought on  
 028 the MATH dataset. We further observe that  
 029 *TALMs* are not as effective for simpler math  
 030 word problems (in GSM-8k), and the bene-  
 031 fit increases as the complexity and required  
 032 knowledge increases (progressively over AQuA,  
 033 MMLU-Math, and higher level complex ques-  
 034 tions in MATH).

## 1 Introduction

036 State-of-the-art Large language models (LLMs), includ-  
 037 ing gpt-3.5-turbo, GPT-4, and open-source counterparts  
 038 like Llama 2 have demonstrated impressive performance  
 039 across a broad spectrum of NLP tasks (Brown et al.,  
 040 2020; Radford et al., 2019; Chowdhery et al., 2022;  
 041 OpenAI, 2023). However, their consistent failure on  
 042 established reasoning dimensions, such as mathemat-  
 043 ical, commonsense, abductive, and multi-hop reason-  
 044 ing (Lu et al., 2023b; Cobbe et al., 2021; Huang and

045 Chang, 2023) have led the research community to ex-  
 046 plore various solutions for enhancing their reasoning  
 047 abilities. This pursuit has given rise to techniques, such  
 048 as - (1) **intelligent prompting variations**, such as chain  
 049 of thought (Wei et al., 2022), program of thought (Chen  
 050 et al., 2022), tree of thoughts (Yao et al., 2023), and  
 051 self-refinement (Madaan et al., 2023), (2) **program-**  
 052 **guided solving** that generates python code as intermedi-  
 053 ate steps and offloads execution to a symbolic interpreter  
 054 (Gao et al., 2023), (3) **multi-model interaction frame-**  
 055 **works**, such as Multi-agent Debate (Du et al., 2023;  
 056 Liang et al., 2023) and Round-Table Conference (Chen  
 057 et al., 2023b), (4) **tool-augmented LLMs** powered by ex-  
 058 ternal symbolic tools, APIs, and libraries (Schick et al.,  
 059 2023; Lu et al., 2023a; Paranjape et al., 2023; Yang and  
 060 Narasimhan, 2023; Xie et al., 2023).

061 In this work, we study the effectiveness of tool-  
 062 augmented LLMs (*TALM*) applied to problems involv-  
 063 ing mathematical reasoning. Recent advancements  
 064 in *TALM* frameworks, such as Chameleon (Lu et al.,  
 065 2023a), OlaGPT (Xie et al., 2023), ART (Paranjape  
 066 et al., 2023), and SocraticAI (Yang and Narasimhan,  
 067 2023) have explored the effectiveness of incorporating  
 068 external tools for solving knowledge-intensive reason-  
 069 ing tasks and fundamental mathematical problems (such  
 070 as, arithmetic and algebra). However, the effectiveness  
 071 of *TALM* framework is yet to be validated on mathemat-  
 072 ical reasoning tasks involving complex computations.  
 073 In this context, it is imperative to assess the suitability  
 074 of specific tool combinations across diverse mathemat-  
 075 ical domains (e.g., PreAlgebra, Calculus, Geometry,  
 076 Intermediate Algebra, Probability) at varying levels of  
 077 difficulty. This motivated us to undertake a thorough  
 078 evaluation of *TALM* framework in the context of com-  
 079 plex mathematical reasoning tasks. We propose and  
 080 develop MATHSENSEI, a *TALM*-based framework, com-  
 081 prising a distinct set of tools (also referred to as mod-  
 082 ules), combined in a sequential fashion. These modules  
 083 include LLM-based components, such as - knowledge  
 084 retriever (KR📖), python code generator (PG🐍), code re-  
 085 finer (CR🔧), and solution generator (SG🌱); and APIs,  
 086 such as - Bing-Web-Search-API (BS🔍) and Wolfram-  
 087 Alpha-API (WA🔗). As illustrated in Fig. 1, MATHSEN-  
 088 SEI adopts the modular architecture from Chameleon  
 089 (Lu et al., 2023a). Through systematic experiments of  
 090 MATHSENSEI, we aim to discern the effectiveness of  
 091 each module in addressing specific types of mathemat-  
 092 ical problems, having varying levels of difficulty.

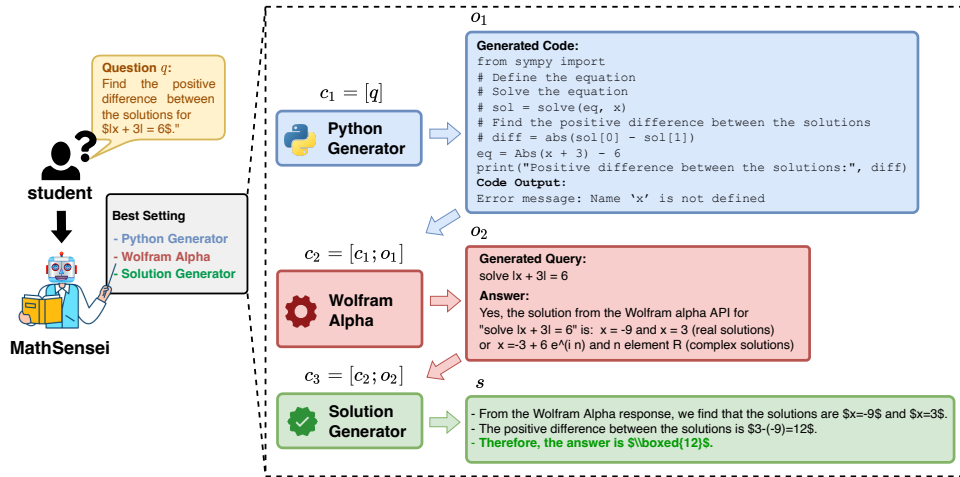


Figure 1: An end-to-end workflow of MATHSENSEI on the compositional setting from the MATH dataset. The final answer is highlighted in green font.

TALM	Math Discipline	Form	Search	Python	WAlpha	Plan	Tool-Study
OlaGPT	Algebra	MCQ	✓	✗	✗	Plan-And-Solve	✗
Chameleon	✗	✗	✓	✓	✗	Plan-And-Solve	✗
ART	Algebra	Open	✓	✓	✗	Call-as-req	✗
MATHSENSEI	Algebra, Precalculus, Geometry, Probability, Number Theory & more	Both	✓	✓	✓	Both	✓

Table 1: Comparison of MATHSENSEI with state-of-the-art Tool-Augmented LLMs; **Form** - Question-Answer Format (MCQ with multiple options, Open/Subjective), **Search** - Use of Web Search, **Python** - Python code guided problem solving, **WAlpha** - Wolfram Alpha, **Tool-Study** - Study of each tool, **Plan** - Planning Strategy used; **Plan-And-Solve** - Determine the sequence of tools to be executed beforehand, **Call-as-req** - Dynamically decide to call tool when required at a step during execution.

Our ablations show complimentary abilities of the modules, effect of ordering and combination (such as setting of WA + BS + SG (⚙️ + 📖 + 🌱) surpassing PG + SG (🧠 + 🌱)). This further highlights the need for planning strategies. We evaluate two advanced planning techniques within our pipeline, investigating methodologies such as Plan-And-Solve (Lu et al., 2023a) and REACT (Yao et al., 2022) with MATHSENSEI.

We make following contributions:

1. 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. Compared to MATH, MMLU-Math, our experiments on simpler mathematical datasets (e.g., GSM-8K, AQUA-RAT) reveal minimal benefit of using multiple modules on top of CoT prompting.

2. 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. We observe that the BS📖 module outperforms the KR📖 module for retrieving relevant knowledge for mathematical problems. The setting of WA + BS + SG (⚙️ + 📖 + 🌱) outperforms PG + SG (🧠 + 🌱), demonstrating that program-guided solving techniques (Gao et al.,

2023; Drori et al., 2022) may not be universally suitable for all mathematical problems. These findings motivate the necessity of exploiting better planning techniques. Our best configuration of MATHSENSEI, PG + WA + SG (🧠 + ⚙️ + 🌱) achieves an impressive performance accuracy of 47.6% on the MATH 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 and MMLU-Math are lower, 2.4% and 3.3% respectively, showing the efficacy decreases as requirement of external knowledge decreases.

3. 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. However, we do not observe benefit of using the planners over our best configurations of PG+WA+SG, which may indicate a need for developing targeted planning strategies for mathematical TALMs. We include our Planning related experiments in the Appendix.

## 2 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 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 (Yang et al., 2018).

### 3 Methodology

We first discuss some notations to formalize the problem. Let  $M$  denote the set of modules<sup>1</sup> (each performing a specific task),  $p_i$  be the input prompt for module  $m_i$ , and  $Q$  be the set of mathematical queries.

#### 3.1 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  $[m_1, \dots, m_t]$  be the ordered sequence of chosen modules for answering  $q$ , and  $[o_1, \dots, o_t]$  be the output sequence of the  $t$  modules. Let,  $s_i$ ,  $f_i$ , and  $c_i$  denote the instruction, in-context

<sup>1</sup>The modules can be viewed as external tools, where each module  $m \in M$  can be either powered by LLMs, such as Python code generators, Knowledge Retrievers, or they can be non-LLM API tools, such as WolframAlpha, Bing Web Search.

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:

$$p_i = \langle s_i; f_i; c_i \rangle \quad (1)$$

where context  $c_i$  is defined as:

$$c_i = \begin{cases} [q], & \text{if } i = 1; \\ [c_{i-1}; o_{i-1}], & \text{for } i = 2, \dots, t \end{cases} \quad (2)$$

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

### 3.2 Modules

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 14. A detailed description of the prompts used in each module is presented in the Appendix section.

- **LLM-based Knowledge Retrieval (KR<sup>📖</sup>)** - 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 14) 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 Table 9 (§A).

- **Bing Web Search (BS<sup>🌐</sup>)** - 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. 2 for an example).

- **Wolfram Alpha (WA<sup>🔧</sup>)** - 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<sup>🔧</sup> module is presented in Fig. 3.

- **Python Generator+Executor (PG<sup>🐍</sup>)** - We use an LLM that takes as input the current context as a part of a well-structured prompt (shown in Appendix Fig. 4). 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:

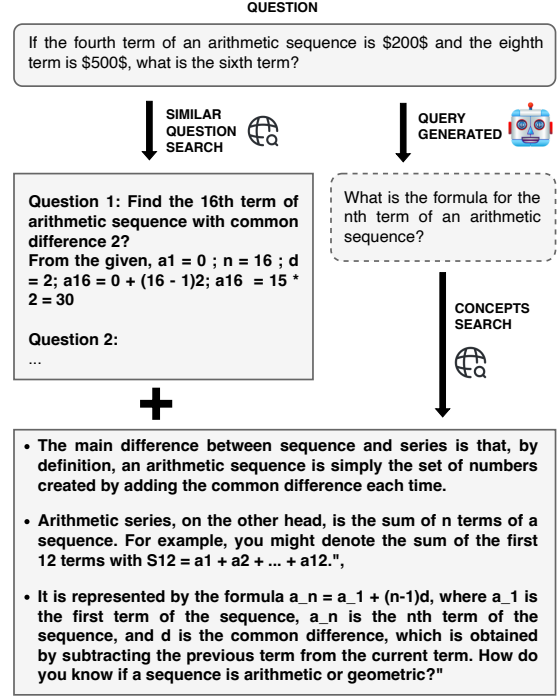


Figure 2: Overview of the BS<sup>🌐</sup> module; We concatenate the similar questions and concepts (which is then used by a downstream module).

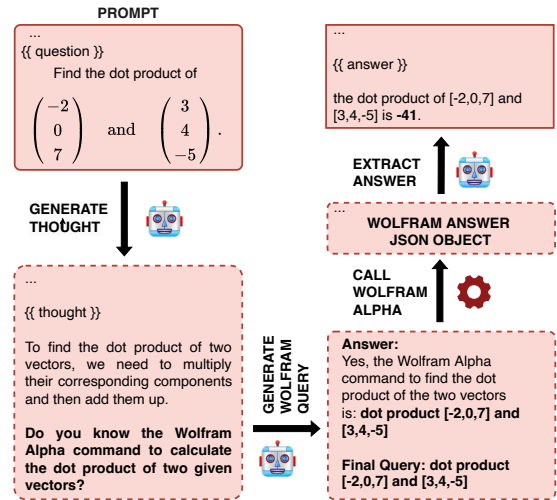


Figure 3: Overview of the WA<sup>🔧</sup> module.

- **Without refinement:** Here, if generated code produces syntax errors, we omit the output of PG<sup>🐍</sup> from the context for next module. 313-315
- **Code-Refinement (CR<sup>🔧</sup>):** 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 316-323

MATH dataset is presented in Appendix 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  $\boxed{\text{ }}$  (similar to the MATH dataset).

## 4 Experimental Setup

We first introduce the mathematical datasets used in our study (§4.1), followed by the experiments that we perform with various combinations of modules (§4.2). We use gpt-3.5-turbo (🌱) as the default LLM in LLM-based modules unless mentioned otherwise. This is mainly because it is more accessible and cheaper compared to models like GPT-4. For querying a search-engine, we use Bing-We-Search-API. Please refer to §A.1 for details about online resources that we use.

### 4.1 Datasets

**MATH.** The MATH dataset (Hendrycks et al., 2021b) serves as the primary dataset for our work. It covers 5000 mathematical problems, which are categorized into seven subject types (*Precalculus, Prealgebra, Algebra, Geometry, Intermediate Algebra, Counting and Probability, and Number Theory*) and *five levels* of difficulty (ranging from 1 to 5, where 1 denotes the least difficult and 5 denotes the most difficult). Our choice of the MATH dataset is motivated by its unique characteristics: Unlike many datasets, scaling up LLMs (in terms of model parameters) does not necessarily enhance accuracy on MATH. The dataset also poses intricate challenges, going beyond simple arithmetic or high school mathematics problems.

**AQUA-RAT.** The AQUA-RAT dataset (Ling et al., 2017) contains 253 *algebraic math word problems* with rationales. Unlike the MATH dataset, it has a *multiple-choice* answer format with five options. It allows us to evaluate MATHSENSEI on mathematical problems in the domain of algebra.

**GSM-8K.** GSM-8K (Cobbe et al., 2021) contains *high school level math word problems* which require basic arithmetic operations (addition, subtraction, multiplication, and division) to reach the final answer. The final answer is always an integer value. We use all 1319 examples from GSM-8K test set for evaluation.

**MMLU-Math.** The MMLU dataset (Hendrycks et al., 2021a) covers 57 diverse tasks (including *elementary mathematics, US history, computer science, etc.*), which require extensive problem solving abilities and world knowledge. For this work, we use the mathematical test subset of MMLU, known as MMLU-Math that contains 974 mathematical questions spanning 5 types - *abstract algebra, elementary mathematics, high-school mathematics, college mathematics, and formal logic*. Similar

to AQUA-RAT, MMLU-Math also has a *multiple-choice* answer format.

### 4.2 Experiments

We conduct several experiments by meticulous analysis of individual modules **in the domain of complex mathematical reasoning**, through systematic ablations on the module sequences. For some of our ablations, we use different variants of OpenAI models, such as text-davinci-002 (🌱) and text-davinci-003 (🌱) other than the default gpt-3.5-turbo (🌱). We also employ models from the Llama family, such as Llama-2-7B (🌱) and Phind-Code-Llama-34B-V2 (🌱). We use accuracy as our evaluation metric for comparing different settings. Our experiments enquire the following questions:

- What is the impact of adding LLM generated mathematical knowledge relevant to the question [**KRA🌱 module**] before invoking the Solution Generator module [**SG🌱 module**] (§5.1)
- How does the Bing Web Search [**BS🌱 module**] compare against the LLM-based knowledge generation [**KRA🌱 module**] for the task of adding relevant mathematical knowledge and information to the problem solving process? (§5.1, §5.2)
- What is the utility of augmenting mathematical knowledge-bases, such as Wolfram Alpha [**WA🌱 module**] with LLMs for solving problems across different levels of complexity? How does it compare against the paradigm of program-guided solving? (§5.3)
- What are the benefits of using program guided complex problem solving [**PG🌱 module**], and impact of LLM-based code refinement [**CR🌱 module**] in case of syntactical errors? (§5.4)
- What is the effect of using **multiple modules together**? How does the benefit vary with the **difficulty level, mathematical subject type, and dataset**? (§5.5)
- How to plan effective utilization of these modules? How does non-adaptive planning strategies [**Plan-And-Solve**] compare against dynamic planning strategies such as [**REACT**] which uses a thought, action, and observation based mechanism. (Appendix §A.1)

## 5 Effects of Adding Modules over LLMs

Here, we present results and analyze the impact of adding individual modules on top of the original LLM CoT variant (termed SG🌱): 1) **KRA🌱** in § 5.1, 2) **BS🌱** in §5.2, 3) **PG🌱** in §5.4, and 4) **WA🌱** in §5.3. For each module, we also provide ablations over different LLMs (as applicable).

### 5.1 LLM-Based Knowledge Retrieval (KR)

Recently, Chameleon (Lu et al., 2023a) demonstrated an accuracy boost for knowledge intensive QA datasets, such as ScienceQA and TabMWP by using the **KRA🌱** module. Skills-In-Context prompting (Chen et al., 2023a) also shows similar results by utilizing some basic skills (such as mathematical theorems, during generation). Following the literature, we investigate the

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

Table 2: 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.

433 impact of adding relevant knowledge (such as mathe-  
 434 matical concepts and formulae) using an LLM-based  
 435 KR🗄️ module in the context of SG🗣️ module, and exam-  
 436 ine the efficacy of the KR + SG (🗄️ + 🗣️) setting on the  
 437 MATH dataset (Table 4). We also ablate over different  
 438 LLMs (Table 2) to power the KR🗄️ module, while fixing  
 439 the SG🗣️ module to gpt-3.5-turbo (🗣️).

440 **Results.** As shown in of Table 4, the extra knowledge  
 441 retrieved by the KR🗄️ module is useful only for prob-  
 442 lems in Algebra, PreAlgebra, and Probability domains.  
 443 Moreover, the overall accuracy drops steadily as we  
 444 change KR🗄️’s LLM from gpt-3.5-turbo (🗣️) to other  
 445 variants (shown in Table 2). This indicates that, generic  
 446 LLMs (such as those mentioned in Table 2) are not  
 447 equipped with mathematical concepts of other domains  
 448 (Precalculus, Gemetry, Number Theory, Intermediate  
 449 Algebra). After analyzing different LLM variants for  
 450 the KR🗄️ module, we find that the knowledge retrieved  
 451 by weaker LLMs heavily degrades performance of the  
 452 downstream SG🗣️ module. This motivated us to explore  
 453 the impact of search engine-based knowledge retrieval  
 454 (detailed in §5.2).

## 455 5.2 Query Generation for Bing Web Search (BS)

456 We investigate the advantages of adding a search engine-  
 457 based knowledge retrieval module (BS🗄️) as an alterna-  
 458 tive of KR🗄️ for **similar questions search** and **concepts**  
 459 **search** before applying SG🗣️.

460 **Results.** In Table 3, we observe that BS + SG (🗄️  
 461 + 🗣️) setting is a clear winner over the SG🗣️ set-  
 462 ting, when gpt-3.5-turbo (🗣️) is used for generating  
 463 the BING-Web-Search-API (🗄️) query and getting fi-  
 464 nal solution from SG🗣️. This holds true even if the  
 465 stand-alone SG🗣️ is varied between text-davinci-003  
 466 (🗄️) (+22.5%) and gpt-3.5-turbo (🗣️) (+4.2%).  
 467 Thus, augmenting LLMs with knowledge (relevant to a  
 468 mathematical question) retrieved from the web proves to  
 469 be beneficial in improving problem solving capabilities.  
 470 The use of text-davinci-003 (🗄️) alone or in com-  
 471 bination with gpt-3.5-turbo (🗣️) for BS🗄️ and SG🗣️  
 472 modules, diminishes the performance of both BS + SG  
 473 (🗄️ + 🗣️) and SG (🗣️) settings , which is expected (Ye  
 474 et al., 2023).

## 475 5.3 Wolfram Alpha Search (WA)

476 We compare the performance of WA + SG (🗄️ + 🗣️) and  
 477 SG🗣️ settings on the MATH dataset in Table 3. We

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 3: 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.

perform ablations with text-davinci-003 (🗄️) and gpt-  
 3.5-turbo (🗣️) as the LLMs used in WA🗄️ for query gen-  
 eration and answer extraction.

481 **Results.** From Table 3, we observe that WA + SG  
 482 (🗄️ + 🗣️) outperforms the SG🗣️ approach by 8.1%, when  
 483 both WA🗄️ and SG🗣️ are powered by gpt-3.5-turbo  
 484 (🗣️). This shows a clear and significant contribution of  
 485 complementary strengths coming from the knowledge  
 486 retrieved through Wolfram Alpha (🗄️). Furthermore, it is  
 487 notable that the observed benefits of the WA🗄️ module  
 488 cannot be solely attributed to the characteristics of the  
 489 LLMs employed for query generation or answer extrac-  
 490 tion. This is evident from the substantial performance  
 491 gains (around 10.8%) achieved, even after enabling both  
 492 WA🗄️ and SG🗣️ with a comparatively weaker model,  
 493 such as text-davinci-003 (🗄️). Additionally, the mix  
 494 of text-davinci-003 (🗄️) and gpt-3.5-turbo (🗣️) for  
 495 the WA + SG (🗄️ + 🗣️) setting demonstrates superior  
 496 performance compared to SG🗣️ with gpt-3.5-turbo  
 497 (🗣️), achieving improvements of 1.1% and 3.3%, respec-  
 498 tively. Thus, showcasing meaningful positive impact of  
 499 augmenting WA🗄️ with the stand-alone SG🗣️ module.

## 500 5.4 Python Generator (PG)

501 In this section, we investigate the effectiveness of the  
 502 Python Generator (PG🗣️) module in using python code,  
 503 and an interpreter to solve mathematical problems (uti-  
 504 lizing external symbolic libraries from sympy). Fol-  
 505 lowing, PAL (Program Aided Language Models) (Gao  
 506 et al., 2023), Program of thought (Chen et al., 2022),  
 507 our PG🗣️ module consists of a program generator and  
 508 an executor. The generated code and corresponding  
 509 output are added in context of the next module in se-  
 510 quence. We present the results of the PG + SG (🗣️ + 🗣️)  
 511 setting in Table 4 for the MATH dataset. For MATH,  
 512 we present three variations: (i) PG + SG (🗣️ + 🗣️) with  
 513 no code refinement, (ii) PG + CR + SG (🗣️ + 🗣️ + 🗣️) with  
 514 code refinement, (iii) PG’[🗄️] + SG (🗣️ + 🗣️) with Phind-  
 515 CodeLLama-34B-V2 model used as the LLM for PG  
 516 Module. We choose **Phind-CodeLLama-34B-V2** for  
 517 our ablation since it is the best model from the hug-  
 518 gingface Code-LLM leaderboards. The Phind family of  
 519 models are finetuned versions of CodeLLama-34B on a  
 520

Method	Alg	P.Cal	P.Alg	Geom	Prob	N.Th	Int.Alg	O.Acc
<b>Baselines with gpt-3.5-turbo (🧠)</b>								
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
<b>Baselines with GPT-4</b>								
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
<b>Ours</b>								
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 (🧠 + 📐 + 🧠)	59.7	25.2	63.9	26.9	48.3	43.0	26.9	44.8
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); 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).

Phind dataset consisting of 80k high quality programming problems and solutions.

**Results.** In Table 4, we observe that the PG + SG (🧠 + 🧠) setting using the sympy library without code refinement can improve upon the performance accuracy of SG (🧠) on the MATH dataset by a 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, on the other hand, has support for symbolically representing such objects using well defined functions, classes, methods, and sub-packages. We find that this helps PG outperform SG on all mathematical types in MATH. The outcomes of our experiment with PG + CR + SG (🧠 + 📐 + 🧠) setting only yields marginal enhancements on overall accuracy. We also observe a drop in the accuracy by 5% when using Phind-CodeLLama-34B-V2 as the LLM in PG module.

## 5.5 Results of Multiple Module Experiments

We experiment with various module combinations on four datasets MATH, AQUA-RAT, GSM-8K, and

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

MMLU-Math and report in Tabs. 4 & 6. Our findings reveal that distinct modules exhibit specialized efficacy in addressing specific categories of mathematical problems. On the **MATH** dataset, (1) WA (🔧) emerges as a valuable resource for tackling intricate mathematical subdomains, particularly in Intermediate Algebra (**Int.Alg**) and Number Theory (**N.Th**). The PG+WA+SG (🧠+🔧+🧠) setting outperforms SG (🧠) by 19% on Int.Alg. We conduct a qualitative analysis of PG+SG (🧠+🧠) on 106 randomly sampled questions

Setting	GSM-8K	AQUA	M.Math
(🧠)	77.0	61.4	66.2
(📚+🧠)	71.8	57.5	64.5
(🔧+🧠)	61.7	57.9	66.0
(🌐+🧠)	56.0	53.5	67.6
(🔗+🧠)	74.1	55.1	68.1
(🔧+🔗+🧠)	69.1	63.8	65.1
(🔗+🔧+🧠)	67.6	62.6	67.1
(🔗+🌐+🧠)	67.6	58.3	67.2
(🌐+🔗+🧠)	69.2	56.3	69.5
(🌐+🔗+🔧+🧠)	70.7	61.4	66.9

Table 6: Comparison of Multi-Module Settings for GSM-8K, AQUA-RAT (AQUA), and MMLU-Math (M.Math) datasets.

from MATH spanning all types and difficulty levels, presented in Table 13. We find that the majority of errors in Int.Alg 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.(2) For Algebra-related problems (**Prealgebra** and **Algebra**) having complex computations, the generation of Python code guided by PG (🔗) and the sympy library proves to be an effective choice. The WA+PG+SG (🔧+🔗+🧠) setting elevates the performance of SG (🧠) by 15.8% on Algebra. The PG+SG (🔗+🧠) setting performance is also significantly better compared to SG (🧠) (10.4%) on Prealgebra showing the utility of code representations over natural language in this subdomain. (3) Table 8 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 levels with diverse modular configurations. This reaffirms the importance of judiciously selecting tools and configurations based on the specific features and attributes of the given problem.

**Effectiveness of MATHSENSEI on MMLU-Math.** Results in Table 6 reveal that the BS+PG+SG (🌐+🔗+🧠) configuration enhances the accuracy of the SG (🧠) setting by 3.3%. As the performance is gain is low, we further perform a type wise analysis in Table 5. 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. More specifically we find that: (1) The PG+WA+SG (🔗+🔧+🧠) setting improves the accuracy of the SG (🧠) setting from 84.6% to 92.1% on **Elementary mathematics** problems. (2) Interestingly, problems in **formal logic** are best solved using SG (🧠) alone. The drop in performance for the PG+SG (🔗+🧠) setting (53.9-> 49.5) is due to the inability of PG to adequately represent predicate logic, First Order Logic (FOL) sentences through python code, (3) For **College Mathematics**, the Wolfram Alpha (🔧) module demonstrates highest efficacy, as evidenced by the

substantial benefits observed in both the WA+SG (🔧+🧠) and WA+PG+SG (🔧+🔗+🧠) settings. Notably, WA+SG (🔧+🧠) outperforms the SG setting by a significant margin of 17%. Our analysis in MMLU-Math further supports the complimentary benefit of the tools used in MATHSENSEI framework for various mathematical types.

**Decreased Effectiveness of MATHSENSEI on GSM-8k, and AQUA-RAT.** From Table 6, we observe marginal improvement of using multiple modules on AQUA-RAT and GSM-8k, over the standalone SG (🧠) module. Both datasets mainly comprise simpler algebraic and arithmetic word problems. GSM-8K consists of problems requiring simple arithmetic operations, and mostly does not require external mathematical knowledge. The complexity in GSM-8K stems more from linguistic diversity. We conduct a case study on a randomly sampled set of 20 examples from **GSM-8K**, where PG + SG (🔗 + 🧠) is incorrect and SG (🧠) is correct, we find that 18 (out of 20) have incorrect outputs generated by PG (🔗) (due to reasoning errors)(Table 15). For all these 18 examples, the LLM generated python code tries to solve a simple problem by using complex objects in sympy, which in turn degrades the performance. For the remaining two examples, one has an execution error, while for the other one, SG (🧠) alters the correct answer to incorrect.

Similar to GSM-8K, **AQUA-RAT** primarily focuses on problems that require generic language-based reasoning skills. We find that settings with WA (🔧) 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 (🧠). As we saw previously in case of GSM-8K, a significant proportion of errors in PG + SG (🔗 + 🧠) can be linked to the application of sympy for simple problems (Table 15). These outcomes highlight the diminishing utility of employing additional modules for tasks requiring minimal external knowledge.

## 6 Conclusion

We introduce a Tool-augmented Large Language Model (TALM) framework, aka MATHSENSEI, targeted for Mathematical Reasoning. We utilize tools for web-based knowledge retrieval, program execution, and symbolic equation solving. We perform extensive ablations over the individual tools, along with varying the order and combination 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 tool-sequencing methods does not improve over our best configuration. We also observe that benefit of mathematical TALMs are minimal for simpler math word problems (in GSM-8k) and its benefit increases as the required complexity and knowledge for the problem increases through AQUA, MMLU-Math.



## 656 Limitations

657 We propose a Tool-Augmented LLM framework  
658 (*TALM*), uniquely targeted towards complex mathematical  
659 reasoning. Here, we discuss three types of lim-  
660 itations: 1) choice of the set of tools, 2) variants of  
661 the PG module for simpler problems and 3) developing  
662 mathematical *TALM*-specific planning methods.

663 1. Here, we choose tools, which intuitively offers  
664 knowledge about complex mathematical disciplines and  
665 complex equation solving capabilities such as Python  
666 with sympy library, Wolfram-Alpha-API and Bing Web  
667 Search API. However, we have not explored other  
668 solvers which are targeted towards logical complexity  
669 or adding commonsense knowledge. In future, a more  
670 universal *TALM* can target adding Z3, SAT solvers and  
671 OMCS knowledge base query capabilities.

672 2. Our Program Generator (PG) module is not only  
673 inspired by the program-guided solving methods, but  
674 also targetedly use sympy library to access complex  
675 mathematical equation solving skills. Such skills may  
676 not be required for simpler math word problems, as  
677 present in GSM-8k. In future, we plan to work on  
678 generalizing the PG module so that it is adaptive for  
679 simpler problems and focuses mainly on representing  
680 the problems in code, only accessing sympy capabilities  
681 when required.

682 3. Lastly, we worked on vanilla adaptation of the  
683 available planning or tool-sequencing methods directly  
684 in the mathematical *TALM* (or MATHSENSEI) context.  
685 From our experiments, it is clear that we need to develop  
686 more efficient planners that can dynamically choose  
687 a sequence of tools based on the problem type (say  
688 WA+PG+SG for algebra and PG + CR+ SG for Probabil-  
689 ity), striking a balance between planning beforehand  
690 (Plan-And-Solve) and example-wise planning (REACT).  
691 We hope our work will inspire researchers to work on  
692 such planning methods for mathematical *TALMs*.

## 693 References

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

700 Jiaao Chen, Xiaoman Pan, Dian Yu, Kaiqiang Song,  
701 Xiaoyang Wang, Dong Yu, and Jianshu Chen. 2023a.  
702 [Skills-in-context prompting: Unlocking compositionality in large language models.](#)

704 Justin Chih-Yao Chen, Swarnadeep Saha, and Mohit  
705 Bansal. 2023b. [Reconcile: Round-table conference improves reasoning via consensus among diverse llms.](#)

708 Wenhu Chen, Xueguang Ma, Xinyi Wang, and  
709 William W Cohen. 2022. Program of thoughts

prompting: Disentangling computation from reason-  
ing for numerical reasoning tasks. *arXiv preprint*  
*arXiv:2211.12588*.

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

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.  
2021. Training verifiers to solve math word prob-  
lems. *arXiv preprint arXiv:2110.14168*.

Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard  
Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda  
Chen, Sunny Tran, Newman Cheng, Roman Wang,  
Nikhil Singh, Taylor L. Patti, Jayson Lynch, Avi Sh-  
porer, Nakul Verma, Eugene Wu, and Gilbert Strang.  
2022. [A neural network solves, explains, and gener-  
ates university math problems by program synthesis  
and few-shot learning at human level.](#) *Proceedings*  
*of the National Academy of Sciences*, 119(32).

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

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

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

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

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

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

Jie Huang and Kevin Chen-Chuan Chang. 2023. [To-  
wards reasoning in large language models: A survey.](#)

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish  
Sabharwal, Oren Etzioni, and Siena Dumas Ang.  
2015. [Parsing algebraic word problems into equa-  
tions.](#) *Transactions of the Association for Computa-  
tional Linguistics*, 3:585–597.

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



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

<b>[Instruction]</b>
Read the following question, generate the background knowledge or relevant mathematical information as the context information that could be helpful for answering the question.
<b>[In-context example(s)]</b>
The smallest distance between the origin and a point on the graph of $[y = \frac{1}{2}x^2 - 9]$ can be expressed as $a$ . Find $a^2$ .
Knowledge:
- The question involves the concept of distance between two points in a coordinate system.
- The distance between the origin and a point $(x, y)$ on the graph is given by the formula $\sqrt{(x^2 + y^2)}$ .
- Minimizing $(x^2 + y^2)$ is same as minimizing $\sqrt{(x^2 + y^2)}$ .
- Substitute $y = 1/2x^2 - 9$ into the equation to get $x^2 + (1/2x^2 - 9)^2$ .
- Take derivative of above equation and set it to 0.
<b>[Input Question]</b>
What is the following value when expressed as a common fraction: $\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^8} + \frac{1}{2^9} + \frac{1}{2^{10}}$ ?
<b>Output</b>
- 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 $S_n = a(1 - r^n)/(1 - r)$ , 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.

Table 9: Example of Knowledge Retrieval (KR) Prompt and Output.

<b>[Instruction]</b>
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.
<b>[In-context example(s)]</b>
Question: Determine the number of ways to arrange the letters of the word ELEVEN.
Modules: ['bing-search', 'solution-generator']
<b>[Input Question]</b>
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?
<b>Output</b>
Modules: ['python-generator', 'solution-generator']

Table 10: Example of Planner Prompt and Output in Plan-And-Solve (PAS).

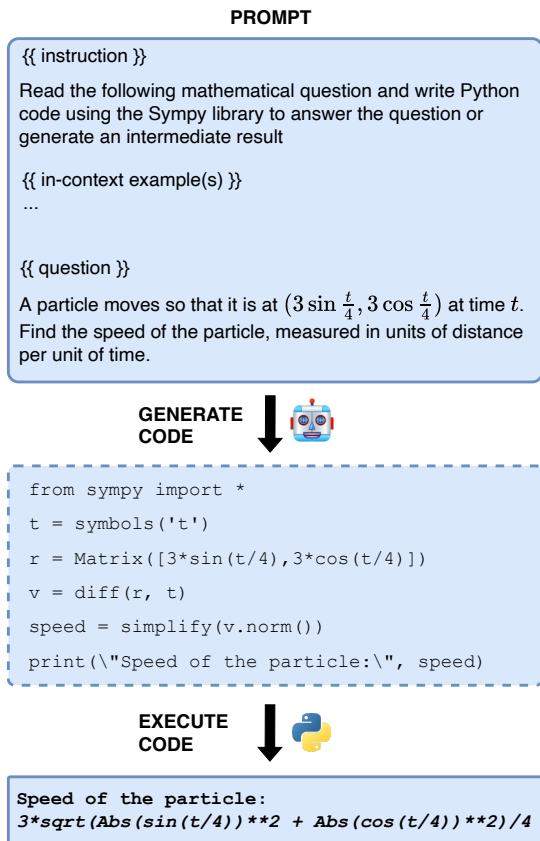


Figure 4: Overview of the PG Module (without code refinement).

### 867 A.1 Planning Strategies

868 We explore two state-of-the-art planning strategies  
 869 based following the Chameleon (Lu et al., 2023a) and  
 870 the REACT (Yao et al., 2022) frameworks and report in  
 871 in Table 11.

872 **Plan-And-Solve** Within the Plan-And-Solve (PAS)  
 873 framework, a dynamic planner (LLM), generates a plan  
 874 for a given mathematical problem before the start of exe-  
 875 cution. In our context, the plan consists of the sequence  
 876 of modules to be run. Notably, this planning approach is  
 877 inherently non-adaptive, as the strategy lacks the capa-  
 878 bility to determine the next module based on feedback  
 879 and the output of the previously executed modules. To  
 880 instruct the planner LLM, we provide input prompts  
 881 containing information about each module, along with  
 882 a few-shot example representing a possible sequence.  
 883 The prompts utilized for the planner model are detailed  
 884 in Table 10.

885 **MATHSENSEI with REACT Planner.** The previous  
 886 modular settings, have a fixed order of execution of the  
 887 modules. However, we also wish to test out settings  
 888 where there is power given to the central LLM to call  
 889 different modules as and when required. This is done  
 890 by executing (thought, action request, action execution)  
 891 triplets. The thought serves as a summary of what we  
 892 have till now in relation to answering the question, the  
 893 action request is the specific action we wish to take in

894 the next step, and the action execution step calls the nec-  
 895 essary module from the modules library to execute the  
 896 action. We call this setting **thought-request-execution**  
 897 (**Tho-Re-Exec**), an overview of the setup is presented  
 898 in Figure 2. The results for this setting corresponding to  
 899 each problem type is presented in Table 11.

900 **Online Resources.** List of online resources that we  
 901 use are as follows:

- 902 • open-source icons: [https://iconduck.com/](https://iconduck.com/icons/)  
903 [icons/](https://iconduck.com/icons/)
- 904 • llama-2 icon: [https://llama-2.](https://llama-2.ai/wp-content/uploads/2023/08/Llama-2-icon-150x150.png)  
905 [ai/wp-content/uploads/2023/08/](https://llama-2.ai/wp-content/uploads/2023/08/Llama-2-icon-150x150.png)  
906 [Llama-2-icon-150x150.png](https://llama-2.ai/wp-content/uploads/2023/08/Llama-2-icon-150x150.png)
- 907 • codellama icon: [https://codellama.dev/](https://codellama.dev/icons/black-transparentbg.png)  
908 [icons/black-transparentbg.png](https://codellama.dev/icons/black-transparentbg.png)
- 909 • python icon: [https://s3.dualstack.](https://s3.dualstack.us-east-2.amazonaws.com/pythondotorg-assets/media/community/logos/python-logo-only.png)  
910 [us-east-2.amazonaws.com/](https://s3.dualstack.us-east-2.amazonaws.com/pythondotorg-assets/media/community/logos/python-logo-only.png)  
911 [pythondotorg-assets/media/community/](https://s3.dualstack.us-east-2.amazonaws.com/pythondotorg-assets/media/community/logos/python-logo-only.png)  
912 [logos/python-logo-only.png](https://s3.dualstack.us-east-2.amazonaws.com/pythondotorg-assets/media/community/logos/python-logo-only.png)
- 913 • azure openai service: [https://azure.](https://azure.microsoft.com/en-us/products/ai-services/openai-service/)  
914 [microsoft.com/en-us/products/](https://azure.microsoft.com/en-us/products/ai-services/openai-service/)  
915 [ai-services/openai-service/](https://azure.microsoft.com/en-us/products/ai-services/openai-service/)
- 916 • bing web search api service: [https:](https://www.microsoft.com/en-us/bing/apis/bing-web-search-api)  
917 [//www.microsoft.com/en-us/bing/apis/](https://www.microsoft.com/en-us/bing/apis/bing-web-search-api)  
918 [bing-web-search-api](https://www.microsoft.com/en-us/bing/apis/bing-web-search-api)

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 11: 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 12: Comparing Performance of different Planning Strategies (§??) with two of our Top Performing Settings by varying Difficulty Level of Problems from the MATH dataset.

Subject	PG-Exec-Err	PG-R-Err	SG-Err	Tot.Examples
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
<b>Overall Count</b>	<b>51</b>	<b>48</b>	<b>7</b>	<b>106</b>

Table 13: Summary of Error types with PG + SG (🌐 + 🌱) setting on a random subset of 106 examples (MATH dataset); 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. Here we report the absolute count of errors across different subjects.

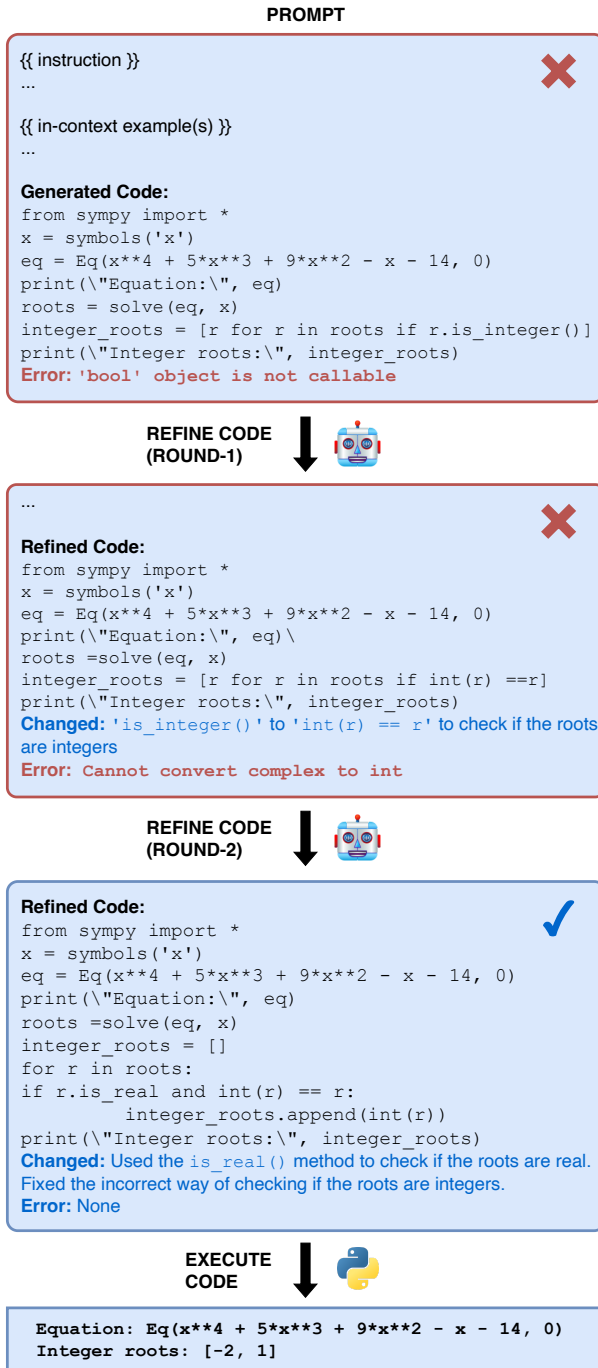


Figure 5: Overview of the Code Refiner module.

Module	Models/APIs or Tools
KR	text-davinci-002 text-davinci-003 Llama-2-7B gpt-3.5-turbo
BS	text-davinci-003 gpt-3.5-turbo Bing-Web-Search-API
WA	text-davinci-003 gpt-3.5-turbo Wolfram-Alpha-API
PG	gpt-3.5-turbo Phind-CodeLLama-34B-V2 Python Interpreter
CR	gpt-3.5-turbo
SG	text-davinci-003 gpt-3.5-turbo

Table 14: Module Inventory.

Dataset	PG-R-Err	PG-Exec-Err	SG-Err
GSM-8K	18	1	1
AQUA-RAT	13	6	1

Table 15: Case study of error types of PG+SG on GSM-8K and AQUA-RAT datasets for 20 randomly selected examples on which PG fails and 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

**[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  $30 \times .25 = 7.5$  He needs to remove 7.5 more pounds because  $15 - 7.5 = 7.5$  He needs to remove 15 toys because  $7.5 / .5 = 15$

**[PG+SG Output]**

Python generator:

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

Python output:

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

**[SG Output]**

Solution:

- 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  $30 \times (1/4) = 7.5$  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  $x \times (1/2) = 0.5x$  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

Table 16: Example showing PG+SG (🟡+🟢) (Incorrect) vs SG (🟢)(correct) for GSM-8K