HYBRIDMIND: Meta Selection of Natural Language and Symbolic Language for Enhanced LLM Reasoning

Anonymous authors
Paper under double-blind review

Abstract

LLMs approach logical and mathematical reasoning through natural or symbolic languages. While natural language offers human-accessible flexibility but suffers from ambiguity, symbolic reasoning provides precise, machine-executable inferences at the cost of strict domain constraints. We introduce HYBRIDMIND, an adaptive strategy that selects the optimal reasoning approach for each reasoning problem. Through extensive experiments, we evaluate both prompting-based approaches with state-of-the-art LLMs and fine-tuned open-source models. We find that fine-tuning LLaMA-3.1-8B-Instruct as a meta-selector outperforms GPT-4o's natural language reasoning by 4.4% on FOLIO and 1.3% on MATH. More notably, using GPT-3.5-turbo as a prompted meta-selector yields a 10% improvement on FOLIO's challenging subset compared to GPT-4o. We will release our code and data to support future research.

1 Introduction

Mathematical reasoning with LLMs is typically approached through two paradigms. The first paradigm focuses on designing various prompting strategies to elicit detailed and natural language (NL) reasoning processes. This line of research continues from Chain-of-Thought prompting (Wei et al., 2023; Zhou et al., 2023; Zheng et al., 2024a). The second one leverages LLMs to generate solutions in the form of symbolic language, which can then be executed with external tools to derive the final answer (Olausson et al., 2023; Ye et al., 2023; Zhou et al., 2024a; Gu et al., 2024; Gou et al., 2024; Gao et al., 2023; Chen et al., 2023).

Reasoning with symbolic language (SL) incorporates external tools to perform logical deduction and arithmetic calculation, where existing LLMs suffer at when reasoning with purely natural language (Zhou et al., 2024b). Natural language reasoning is highly flexible and broadly accessible to humans, but it often suffers from ambiguities and imprecision (Gao et al., 2023; Liu et al., 2023). In contrast, SL reasoning, being grounded in formal rules and representations, can be executed by machines and offers precise, unambiguous inferences (Chen et al., 2023; Zhou et al., 2024a).

The strengths of each paradigm are complementary: NL reasoning handles semantic richness better, while symbolic representations excel at precise, unambiguous inference and error-checking (Chen et al., 2023; Olausson et al., 2023). Relying exclusively on one approach can thus lead to suboptimal performance. Pure NL reasoning may overlook logical subtleties, whereas purely symbolic methods risk overlooking the complexity and nuances in real-world language (Han et al., 2024; Olausson et al., 2023).

We propose HYBRIDMIND, which leverages the best of both worlds. By dynamically meta-selecting whether to reason in natural language, symbolic language, or a combination of the two, HYBRIDMIND tailors the solution style to the characteristics of each problem. Conceptual or explanatory tasks benefit from the clarity of step-by-step NL reasoning, while problems requiring exact heavy computation are more effectively handled in symbolic form (Gao et al., 2023; Zhou et al., 2024b). In other words, rather than forcing a single method to fit all problem types, HYBRIDMIND adaptively selects the right choice for a problem.

Through extensive experimentation, we demonstrate that fine-tuning or prompting with this meta-selector boosts performance over using only a single reasoning style. For fine-tuning, we adopt Low-rank Adaptation (LoRA) (Hu et al., 2021) for both fully-supervised finetuning and STaR, an iterative fine-tuning method (Zelikman et al., 2022). We propose a systematic process for producing labeled training data for our meta-selector. Specifically, we employ GPT-4o, a state-of-the-art large language model (LLM) to solve logical or mathematical reasoning problems using four approaches (NL reasoning only, SL reasoning only, two approaches of mixing NL and SL reasoning). The meta-selector's label is assigned to whichever approach successfully solves the problem; if multiple approaches are correct, we randomly select one.

The contributions of our paper are threefold. 1) We propose HYBRIDMIND, a method that dynamically chooses between natural language reasoning and symbolic (Python code or first-order logic formulas) reasoning based on a specific reasoning problem. This "meta-selection" strategy ensures that the language model uses the most effective approach, natural language or symbolic reasoning, depending on each problem's characteristics. 2) STaR finetuning over LLaMA-3.1-8B as a meta-selector yields a 4.4% accuracy gain on the FOLIO dataset, a logical reasoning dataset measuring complex logical reasoning capabilities of LLMs (Han et al., 2024). Using GPT-3.5-Turbo as a meta-selector yields a 10% improvement on the more challenging subset of FOLIO over GPT-4o. 3) We also identify key statistics beyond final performance and perform case study and prompt ablation study. We also provide reasons for the performance difference of FOLIO with MATH to interpret the performance improvement of our method, discussing how the difference between Python code and first-order logic formulas could affect the meta-selection results.

2 Related Work

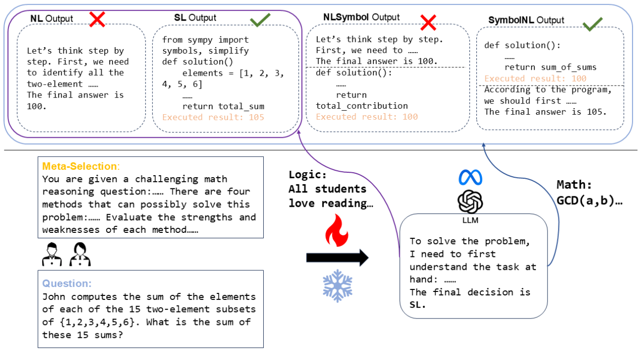
2.1 Reasoning with natural language and symbolic language

Reasoning with NL and SL. Recent research has substantially advanced the logical and mathematical reasoning capabilities of LLMs by incorporating step-by-step reasoning (Kojima et al., 2023; Wei et al., 2023; Wang et al., 2023; Zhou et al., 2023; Zheng et al., 2024a). These methods encourage models to produce explicit intermediate reasoning steps in natural language. While these methods have proven successful across various tasks, they depend entirely on language representations and can yield incoherent or unreliable solutions when confronted with logically complex problems (Han et al., 2024; Olausson et al., 2023).

Symbolic-based approaches take questions in natural language form and generate symbolic forms alone such as Python code or First-Order Logic or generate symbolic forms together with natural language reasoning steps(Gou et al., 2024; Gao et al., 2023; Chen et al., 2023; Imani et al., 2023; Zhou et al., 2024a). External code executors or symbolic engines are then leveraged for verification or deriving the final answer These methods guide the model to generate executable symbolic language forms, enabling direct checks of intermediate or final outputs. Meanwhile, SymbCoT (Xu et al., 2024) converts premises into a symbolic format and uses an LLM-based solver and verifier to validate each step. By grounding reasoning in a checkable programmatic or symbolic form, these methods can offer stronger verifiability than solely natural language–based solutions due to incorporation of an external execution engine.

2.2 Meta selection with LLMs

Zhou et al. (2024b); Gao et al. (2024) use meta prompts to enable LLMs to choose multiple fundamental reasoning modules, such as critical thinking and step-by-step reasoning, and organize them into a clear reasoning structure that guides the model during decoding. However, these methods only adopt natural language reasoning while HYBRIDMIND exploits the of natural language reasoning and symbolic language reasoning, taking into consideration the characteristics of logical and mathematical reasoning problems



```
Let's think step by step. First, we need to identify all the two-element ……
The final answer is 100.
```

```
from sympy import symbols, simplify
def solution()
    elements = [1, 2, 3, 4, 5, 6]
    ……
    return total_sum
Executed result: 105
```

```
Let's think step by step.
First, we need to ……
The final answer is 100.
def solution():
    ……
    return total_contribution
Executed result: 100
```

```
def solution():
    ……
    return sum_of_sums
Executed result: 100
According to the program, we should first ……
The final answer is 105.
```

Meta-Selection: You are given a challenging math reasoning question:…… There are four methods that can possibly solve this problem:…… Evaluate the strengths and weaknesses of each method……

Question: John computes the sum of the elements of each of the 15 two-element subsets of {1,2,3,4,5,6}. What is the sum of these 15 sums?

To solve the problem, I need to first understand the task at hand: …… The final decision is SL.

Figure 1: Illustration of HYBRIDMIND. HYBRIDMIND enables the model to analyze the problem and choose the most suitable approach among NL, SL, NLSymbol, and SymbolNL. We have different reasoning paths for logic and mathematical reasoning problems. In this example, HYBRIDMIND selects SL for a mathematical reasoning problem, which leads to the right solution (105).

3 Method

3.1 Approaches to reasoning

We consider four fundamental approaches involving reasoning with natural language and symbolic language. Natural Language (NL), Symbolic Language (SL), SymbolNL, and NLSymbol. NL represents that the LLM tackles the problem by generating a step-by-step breakdown of the reasoning process in natural language, guiding toward the solution. SL represents that the LLM is instructed to generate a symbolic language solution or symbolic form of the problem and it will be executed to arrive at the final solution. SymbolNL is a two-stage method. In the first stage, the LLM is prompted to write a solution in symbolic language or converts the problem into symbolic language. In the second stage, the LLM step-by-step analyzes the problem in natural language based on the symbolic form to obtain the final answer. NLSymbol is also a two-stage method. The LLM first generates a natural language solution or hints for solving the problem. Then the output will be used to generate symbolic language for solving the problem.

The SymbolNL approach emphasizes the importance of analyzing the code's execution through natural language reasoning, allowing for corrections and insights even if the initial code has errors. In contrast, NLSymbol starts with a natural language outline to clarify the problem before writing the code, which can enhances the likelihood of successful implementation. These two formats leverage both reasoning forms to improve problem-solving effectiveness. The optimal choice among these approaches varies significantly across problem types — some problems benefit from the precision of symbolic reasoning, while others require the flexibility of natural language decomposition (Olausson et al., 2023; Chen et al., 2023; Zhou et al., 2024a).

This observation motivates HYBRIDMIND, a meta-selection framework that analyzes each problem's characteristics to determine the most suitable reasoning approach. As illustrated in Figure 1, before using a specific approach to derive the final answer, HYBRIDMIND uses a meta-selection module to first analyze the problem and decide which reasoning approach to apply among NL, SL, SymbolNL and NLSymbol, before using the selected approach to generate a solution.

The core intuition behind HYBRIDMIND is that not all problems benefit equally from the same reasoning approach (Zhou et al., 2024c). Some problems may require step-by-step natural language reasoning, while others are better suited for symbolic solutions. HYBRIDMIND empowers the model to dynamically adapt its strategy by selecting the most appropriate method, ensuring flexibility and maximizing performance across different problem types.

3.2 Choice of symbolic language

In mathematical reasoning and logical reasoning, the choice between symbolic languages such as Python and formal logical systems like first-order logic (FOL) arises from their fundamental differences in execution and purpose (Russell & Norvig, 2010). Python is a procedural language designed for computation, making it well-suited for mathematical reasoning tasks that involve numerical calculations, symbolic algebra, and algorithmic problem-solving. We include examples where NL reasoning is better and where SL reasoning is better in the appendix. With built-in support for arithmetic operations, iterative processes, and specialized libraries such as NumPy and SymPy, Python efficiently performs both exact and approximate computations (Gao et al., 2023; Zhou et al., 2024a). In contrast, FOL provides a declarative framework for logical reasoning, allowing for the formal specification of knowledge, relationships, and inferential rules (Han et al., 2024; Olausson et al., 2023; Lyu et al., 2023). Unlike Python, which executes a sequence of computational steps, FOL is primarily used for theorem proving and rule-based inference, relying on formal logical operators, quantifiers, and axioms to establish truths within a domain. While Python can simulate logical reasoning through symbolic computation, its execution semantics is step-wise and does not allow general deduction, which may result in unnecessary steps to reach the final value (Russell & Norvig, 2010). Consequently, we select Python to be the symbolic language for math reasoning, and FOL to be the symbolic language for logical reasoning.

3.3 Training data generation

We present a systematic procedure for generating labeled training data for our meta-selector. We use a state-of-the-art LLM, GPT-4o to solve a logical or math reasoning problem based on NL, SL, NLSymbol, and SymbolNL. Each problem is solved in four aforementioned ways using the same LLM. Labels are automatically created by measuring answer correctness with exact match: whichever approach yields the correct solution is treated as the "label" for that problem. If multiple approaches work, one is chosen at random[1]. These "correctness-labeled" examples are then used to train a meta-selector model that can predict which solution strategy is most likely to be correct when given a new problem.

This training data generation method has two major advantages. 1) The data generation is fully automatic with the label directly generated with the original logical or math reasoning dataset answer labels. 2) Different prompting strategies can complement one another; the meta-selector can learn to pick the more suitable strategy for a given problem.

4 Experiments

Our experiments are designed to test whether large language models can reason more effectively by dynamically choosing between natural language reasoning and symbolic language reasoning (or mixing both) rather than relying on a single approach. We explore whether the model can analyze each problem first and then pick the most suitable method — natural language, symbolic language, or a combination to arrive at a solution. By doing so, we aim to see if different types of problems in math and logical reasoning benefits from different styles of reasoning, and whether a "meta-selection" strategy can yield higher accuracy. Implementation details can be found in Appendix A.1.

4.1 Datasets

In our experiments, we consider two challenging datasets: MATH (Hendrycks et al., 2021) for evaluating mathematical reasoning and FOLIO (Han et al., 2024) for evaluating logical reasoning. These two datasets are widely adopted (OpenAI et al., 2024; Xu et al., 2025) and offer a range of categories and difficulty levels, enabling us to conduct various analyses. The MATH dataset comprises 7,498 training samples and 5,000 testing samples in total across 7 categories and 5 difficulty levels; and the FOLIO dataset contains 1,001 training samples and 226 testing samples spanning 2 categories. Our finetuning methods are trained on the

---

[1] Notably, having 4 labels for whether each of the method solves the problem correctly produces a more rigorous setting, but it leads to severe label imbalance which is hard to be mitigated in our initial experiments

entire training set and evaluated based on the entire testing set. Since we reported model performance by category, our experimental setting consists of a total of 9 subsets. Detailed sample counts for each category are provided in Appendix A.2.

4.2 Model choice

We selected several advanced LLMs as the base models for our meta-selector. For *open-source* LLMs, we employ Llama-3.1-8B-Instruct; for *proprietary* LLMs, we utilize GPT-3.5-turbo, GPT-4o, and o3-mini. [2] Finetuning with our training data reduces the bias of keeping selecting NL reasoning, but hurts the final performance. We then transform these base models into meta-selectors by applying either finetuning or prompting. To ensure a fair evaluation, we use 8 shots for both NL and SL, whereas SymbolNL and NLSymbol both use 4 shots at each stage. Details of the complete prompt texts are in the Appendix A.3.

For open-source models, we consider both zero-shot prompting approach and fine-tuning approaches including Supervised Fine-Tuning (SFT) (Zhao et al., 2023; Zheng et al., 2024b; Liu et al., 2024) and iterative finetuning strategy, STaR (Zelikman et al., 2022), to train a meta-selector. Our fine-tuning experiments are based on LLaMA-factory (Zheng et al., 2024b), which is widely employed in LLM development (Liu et al., 2024; Zhao et al., 2023). Proprietary models are based on both the zero-shot prompting and few-shots prompting approaches to act as a meta-selector.

4.3 NL, SL, NLSymbol and SymbolNL

We perform meta selection over NL, SL, NLSymbol, and SymbolNL for MATH since all of the four methods achieve high performance on MATH. Furthermore, the oracle accuracy of at least one of the methods is at least 10.3% higher than that of only one other methods, showing the potential of these methods to be complementary. However, NLSymbol and SymbolNL both achieve very low performance on FOLIO. The first-order logic formulas generated by NLSymbol in the second stage tend to be unexecutable and SymbolNL and the output generated by SymbolNL in the first stage would distract the model from generating a reasoning path. Therefore, for FOLIO we adopt two-class meta-selection over NL and SL without considering NLSymbol and SymbolNL.

4.4 Experiments on FOLIO

For FOLIO, NL and SL denote solving the questions based on Chain-of-Thought (Wei et al., 2023) and LINC (Olausson et al., 2023) using GPT-4o. We only sample once for both methods for efficiency.

Baseline and upperbound. Although state-of-the-art LLMs achieve better performance at reasoning in natural language compared to symbolic language on both of the two subsets we tested (Table 1), the accuracy of at least one of the methods being correct (best-of-2) is around 17% higher than NL or SL reasoning. This score is higher than performing reasoning solely in natural language or symbolic language, demonstrating the complementary strengths of these different reasoning approaches and the potential benefits of selecting between one of them effectively.

Finetuning results. Regarding the analysis of finetuning methods, STaR achieves the best performance, demonstrating the importance of finetuning strategy for enhancing models' capabilities on logical reasoning tasks. Different from STaR, we experiment with applying SFT directly to Llama-3.1-8B to select between NL and SL. These settings cover all cases where a question can be solved using either NL or SL. As shown in Table 1, direct SFT performs poorly on logic reasoning questions, failing to improve the base models' accuracy and predominantly selecting NL as the final solution. This result suggests that the unique design of STaR, characterized by filtering out incorrectly predicted data during training and generating rationale information, might be crucial for effectively addressing logical reasoning tasks.

---

[2] In our initial experiments, we have also tested other models with 3-shot and 8-shot prompting, such as Qwen2.5-14B-Instruct, Qwen-32B-Coder-Instruct, and CodeLlama for meta-selection, however, these models only selected reasoning with NL for all of the tested examples, even when they are given multiple shots, which rendered them a non-optimal choice for testing with our method.

Method	`WikiLogic`	`HybLogic`	Avg.
Baselines			
Random	75.68	60.00	67.70
NL	80.18	60.87	70.35
SL	72.07	69.57	70.80
NL (8 MV)	81.98	66.09	73.89
Finetune			
2-class (base)	81.08	63.48	72.12
2-class SFT	81.08	60.87	70.35
STaR	**82.88**	66.69	74.78
Prompting (0-shot)			
GPT-3.5-Turbo	80.18	59.13	69.74
GPT-4o-mini	75.68	60.00	67.70
GPT-4o	80.18	66.96	73.45
o3-mini	80.18	62.61	71.24
Prompting (2-shot)			
GPT-4o-mini	81.08	66.09	73.45
GPT-4o	81.08	67.83	74.34
o3-mini	80.18	70.43	75.22
HYBRIDMIND (GPT-3.5-Turbo)	79.28	**73.91**	**76.55**
Upperbound			
Best-of-two (NL/SL)	92.79	83.48	88.05

Table 1: Model performance on FOLIO *test* set. **Bold** and underlined numbers indicate the best and second-best performance in each category, respectively. MV: Majority Vote. Best-of-2 (NL/SL): At least one of NL/SL is correct.

Prompting results. In addition to the above methods, we also try prompting advanced LLMs to act as a meta-selector. We generate few shot examples by prompting o3-mini with the premises and conclusion. The rationale generated effectively captures the characteristics of reasoning with NL versus SL. Notably, prompting GPT-3.5-Turbo with 2-shot examples generated by o3-mini achieves the best performance on `HybLogic`, outperforming the second-best method o3-mini by 4.9%. Furthermore, when evaluating the models by difficulty levels, HYBRIDMIND improves upon NL by 17.64% on `HybLogic`, highlighting its strong capacity for addressing complex logical reasoning scenarios. Overall, the best-performing model is GPT-3.5-Turbo, configured as a meta-selector using a few-shot prompting strategy. Moreover, HYBRIDMIND also surpassed NL/SL based on GPT-4o by 8.8% and 8.1%, respectively. This improvement is statistically significant, as validated by the Wilcoxon Rank Sum test (Virtanen et al., 2020) ($p < 0.05$) shown in Appendix A.4. A detailed analysis for these improvements is provided in section 5.2. Our method reduces the model's bias toward NL and utilizes the rationale generated during prompting to select the correct reasoning path for output generation.

Overall, the results demonstrate that HYBRIDMIND functions as an effective meta-selector in choosing the most suitable strategy for a given logic reasoning problem.

4.5 Experiments on MATH

Baseline and upperbound. The results of experiments on MATH are in Table 2. NL baseline achieves the highest performance, while SL performs the worst among the four methods [3]. NLSymbol shows apparently better results, indicating that outlining a reasoning path prior to generating code can improve reasoning with code, particularly in models with limited code capabilities. Each of these methods falls considerably short of the best-of-4 performance, highlighting the potential of dynamically selecting different methods for different problems (Section 5.2).

Prompting results. In the experiment of zero-shot prompting Llama-3.1-8B-Instruct model, the model performance falls closer to the average of the four approaches. This suggests that more advanced models, like o3-mini, are better at selecting the optimal approach based on the problem. Moreover, prompting the Llama-3.1-8B model to act as a meta-selector cannot

---

[3] While SL achieves better performance than CoT in Gao et al. (2023), our adopted dataset and base model are both different from the ones used in their study.

Method	**Algebra**	**Counting & Probability**	**Geometry**	**Number & Theory**	**Intermediate & Algebra**	**Precalculus**	**Prealgebra**	**Avg.**
				Baselines				
NL	**92.67**	78.69	**61.38**	81.48	58.91	60.07	**89.78**	76.98
SL	72.37	78.48	46.56	71.48	44.30	28.94	80.60	62.00
SymbolNL	79.87	74.05	56.99	81.48	58.69	57.51	81.63	71.43
NLSymbol	64.95	82.07	50.31	81.11	51.72	39.38	85.07	65.24
MV (4 methods)	88.29	**84.60**	57.62	85.56	59.36	53.85	**89.78**	75.98
				Finetune				
2-class (base)	72.37	78.48	46.56	71.78	44.30	28.94	80.60	62.00
2-class	**92.67**	78.69	**61.38**	81.48	58.91	60.07	**89.78**	76.98
4-class (base)	72.37	78.48	46.56	71.78	44.30	28.94	80.60	62.00
STaR (fine-tuning)	90.65	79.32	61.17	77.04	58.03	58.24	86.91	75.20
HybridMind (4-class)	92.42	79.54	**61.38**	**82.78**	60.35	**60.99**	**89.78**	**77.50**
				Prompting (0-shot)				
GPT-3.5-Turbo	84.58	78.27	58.04	75.93	55.26	47.44	85.30	71.28
GPT-4o	91.58	81.65	60.96	79.26	**60.69**	57.69	88.98	76.64
o3-mini	87.87	79.75	58.87	76.30	56.81	50.18	87.26	73.24
				Upperbound				
Best-of-2 (NL/SL)	95.79	87.55	66.81	89.63	68.77	64.10	93.57	82.84
Best-of-4	97.47	92.41	72.86	94.07	76.63	72.71	94.60	87.30

Table 2: Model performance on MATH *test* set. **Bold** and underlined numbers indicate the best and second-best performance in each category, respectively. MV: Majority Vote. Best-of-2 (NL/SL): At least one of NL/SL is correct. Best-of-4: At least of one of NL/SL/SymbolNL/NLSymbol is correct.

surpass NL reasoning with GPT-4o, which shows the limitation of the model's capacity for selecting the correct strategy for solving mathematical reasoning problems. On the other hand, prompting advanced models as a meta selector cannot lead to better performance either, which also demonstrates the constraint of prompting in this type of question.

Finetuning results. Methods incorporating NL reasoning (NL, NLSymbol, and SymbolNL) outperform SL, emphasizing the importance of NL in mathematical reasoning. Overall, HybridMind demonstrates higher selection accuracy, effectively identifying the most appropriate approach for each problem. HybridMind also surpasses selectors with the same scale or larger scales in different categories by 0.3% at least and 2.0% at most. Our improvement is statistically significant, as validated by Wilcoxon Rank Sum test (Virtanen et al., 2020) ($p = 0.002$) shown in Appendix A.4.

HybridMind particularly works well in questions belonging to geometry, number theory, precalculus, and prealgebra, and thus it shows the strong capacity of a fine-tuned model for solving complicated geometric or numerical problems. Questions with these types are discussed to be more challenging than other categories for general LLMs (Zhang et al., 2024; Ahn et al., 2024) to solve. A prompt and paired radiational generated by HybridMind can be found in Table 4b, which matches well with regular reasoning paths as a meta selector. Moreover, for finetuning experiments, we have tried different types of meta-selectors, which are summarized in Appendix A.1. The best performer for MATH is a meta-selector trained based on labels created with four methods. Our final choice also outperforms STaR with fewer shots. This could be because that training the meta-selector for math reasoning requires more examples.

Finally, while majority voting performs well on problems with higher accuracy, it struggles with more challenging problems, such as those in the precalculus category. We note that both majority voting (MV) [4] and random choosing cannot surpass our method as well as NL, especially in challenging problems in the precalculus category. This observation demonstrates the necessity of fine-tuning for improving LLMs' mathematical reasoning.

Overall, in this section, we demonstrate that a smaller-scale model can be finetuned as a meta-selector in selecting the most suitable strategy for a given mathematical reasoning problem, leading to better performance than state-of-the-art baselines.

---

[4] When comparing majority voting, it's important to consider the trade-off between effectiveness and cost. Majority voting requires calling the language model to execute each of the four methods, whereas HybridMind only necessitates executing a single selected method.

5 Analysis

5.1 Analysis on FOLIO

The FOLIO dataset consists of two subsets: `WikiLogic` and `HybLogic`. `WikiLogic` presents simpler logical reasoning tasks (1 – 5 reasoning steps), whereas `HybLogic` includes more logically complex problems (5 – 8 reasoning steps). HybridMind improves model performance on `HybLogic` set by 13.04% (Table 1).

Table 3: Analysis on FOLIO

Strategy	**`WikiLogic`**	**`HybLogic`**	**Avg.**
NL	80.18	60.87	70.35
Vanilla	80.18	62.61	71.24
+ wn info	80.18	59.13	69.47
+ wn info + 2-shots	79.28	73.91	76.55

(a) Comparison of different prompting strategies on the FOLIO *test* set using GPT-3.5-Turbo.

	w/o wn info		**w/ wn info**	
Subset	**NL**	**SL**	**NL**	**SL**
`WikiLogic`	111	0	102	9
`HybLogic`	110	5	31	84

(b) Distribution of NL vs. SL selectedby HybridMind on `WikiLogic` and `HybLogic` w/o weakness info and w/ weakness info.

Meta-selector statistics. Initially, the meta selector utilized only a simple instruction prompt, providing limited performance improvement (Vanilla in Table 3a). Subsequent prompt engineering informs the model of the weaknesses of NL: NL reasoning can be be prone to errors if the problem is especially logically challenging and its chain-of-though becomes long or convoluted. It also informs the model of the weakness of SL reasoning (weakness info): SL is not ideal if the premises and conclusion rely heavily on nuanced interpretations of language, or involve concepts that are hard to formalize in FOL. As shown in Table 3b, by informing LLM the weaknesses of both method, HybridMind now employs SL more frequently for more logically challenging `HybLogic` problems and NL for `WikiLogic` examples with more complex natural language patterns. In particular, within the logically challenging `HybLogic` set, there are 26 problems that can be solved by SL but not by NL. Of these 26 problems, HybridMind correctly routes 23 to SL, illustrating its effectiveness in detecting and handling harder reasoning questions.

Case study. Table 4a provides an example where HybridMind accurately discovers complex logical relationships between entities in different premises and identifies “the need for precise logical deductions to reach the conclusion.” It successfully selects SL. This underscores its capacity to dynamically select the most suitable reasoning approach.

5.2 Analysis on MATH

HybridMind can mitigate the bias of choosing correct methods. The bias of LLMs is shown to have negative effects towards generating correct outputs in reasoning tasks (Gupta et al., 2023; Reif & Schwartz, 2024). We noticed that small-scale models before fine-tuning, tended always to choose NL as their final solution, and thus, it is important to investigate the output distribution by choices under both mathematical reasoning and logical reasoning questions. Therefore, we investigated the frequency of choice to solve problems in the MATH and FOLIO datasets, shown in Table 6. Considering the case of random choice, the proportion of each choice will be 25% for MATH and 50 % for FOLIO, while the outputs from our method do not follow this proportion, and our model did not choose the solution randomly but performs rigorous inference. Moreover, base models before fine-tuning had 100% NL choices in the outputs, where our generated outputs only contained 94.10% NL choices for MATH and 58.84% NL choices for FOLIO. Considering the high performance of NL in solving mathematical reasoning problems, the outputs from our method should have a higher proportion of NL choices while it can also select correct choices for other questions that NL cannot resolve, and this improvement also happened in our testing results based on the FOLIO dataset. We also tried to downsample NL in our training data and increase the proportion of SL for both problems as a preliminary attempt

Table 4: HybridMind output example

Input
Premises All runners who ranked in the top 1000 in the Boston Marathon finished the marathon within 3 hours. All runners from the University Running Club ranked in the top 1000 in the Boston Marathon. Some of the runners wearing T-shirts with the bulldog logo are majoring in Computer Science. Tim is a runner and if Tim finishes the Boston Marathon within 3 hours, then he is not both from University Running Club and ranked in the top 1000 in the Boston Marathon. Tim is a runner and if Tim majors in computer science, then he is either wearing a T-shirt with the bulldog logo or is a runner from the University Running Club. **Conclusion** Tim majors in computer science.
Output
... While the premises are not overly complex, the relationships between the entities involved are intertwined and require careful consideration. Given the structured nature of the premises and the need for precise logical deductions ... **Selected method: SL (Symbolic Language).**

(a) HybridMind output on FOLIO.

Input
Question
Six-digit integers will be written using each of the digits 1 through 6 exactly once per six-digit integer. How many different positive integers can be written such that all pairs of consecutive digits of each integer are relatively prime? (Note: 1 is relatively prime to all integers.)
Rationale
... given the limited number of permutations and the straightforward nature of checking the pairwise condition—the SL (Symbolic Language) method is the most effective. ...
Outputs
Selected method: SL (Symbolic Language).

(b) HybridMind output on MATH.

to mitigate existing label bias, but the final performance was reduced by 5.3% in this setting. Therefore, the improvement of HybridMind can also be interpreted by the contribution of reducing selection bias compared with other baselines.

Case study Table 4b provides an example from the number theory category, where HybridMind selects the correct reasoning path in the problem that NL cannot address. HybridMind makes highest percentage of improvement in the problems from this category, and HybridMind will utilize the shots containing the description of problem statistics (e.g., length) as well as the most effective solution and then make a final decision. This specific property allows us to handle complicated metathetical reasoning problems with a stronger meta-selector. We include more examples where NL reasoning is better and where SL reasoning is better in the appendix.

5.3 Comparing FOLIO with MATH

HybridMind produces more substantial improvement on FOLIO than on MATH, as manifested by the smaller p-value. We hypothesize that there could be two reasons. First, FOL formulas are designed to capture the original FOL reasoning question. In contrast, reasoning with Python code like SL for MATH follows a procedural approach that more closely resembles natural language (NL) reasoning than generating FOL formulas for FOLIO. The smaller difference between best-of-2 performance and the NL or SL approach alone indicates that these two classes are more alike, which ultimately makes the classifier more difficult to train because distinguishing between more similar classes is more challenging. For future work, we will explore auto-formalization using Isabelle-form formal specifications as SL for MATH reasoning (Yang et al., 2024; Zhou et al., 2024a). Second, the performance of MATH is highly optimized by both the closed-source LLM, GPT4-o as well as the open-source LLM, such as LLama3.1-8b-Instruct. Therefore, directly prompting these LLMs to select between reasoning with natural language and python without significant prompt engineering does not lead to as much improvement as for FOLIO.

6 Conclusion and Future Work

In this paper, we proposed HybridMind, meta selection of natural language, symbolic language or a mix of both for enhancing LLM math and logical reasoning. Extensive experiments on MATH and FOLIO show that dynamically selecting between reasoning with natural language and reasoning with symbolic language improves the reasoning performance of state-of-the-art LLMs, especially on the harder subset of FOLIO. We performed extensive analysis to understand the performance improvement achieved by HybridMind.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*, 2024.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023. URL https://arxiv.org/abs/2211.10435.

Peizhong Gao, Ao Xie, Shaoguang Mao, Wenshan Wu, Yan Xia, Haipeng Mi, and Furu Wei. Meta reasoning for large language models, 2024. URL https://arxiv.org/abs/2406.11698.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Ep0TtjVoap.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. Cruxeval: A benchmark for code reasoning, understanding and execution, 2024. URL https://arxiv.org/abs/2401.03065.

Shashank Gupta, Vaishnavi Shrivastava, Ameet Deshpande, Ashwin Kalyan, Peter Clark, Ashish Sabharwal, and Tushar Khot. Bias runs deep: Implicit reasoning biases in persona-assigned llms. *arXiv preprint arXiv:2311.04892*, 2023.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alexander Wardle-Solano, Hannah Szabó, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Alexander Fabbri, Wojciech Maciej Kryscinski, Semih Yavuz, Ye Liu, Xi Victoria Lin, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Rex Ying, Arman Cohan, and Dragomir Radev. FOLIO: Natural language reasoning with first-order logic. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 22017–22031, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1229. URL https://aclanthology.org/2024.emnlp-main.1229/.

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, 2021. URL https://arxiv.org/abs/2103.03874.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models, 2023. URL https://arxiv.org/abs/2303.05398.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

Alisa Liu, Zhaofeng Wu, Julian Michael, Alane Suhr, Peter West, Alexander Koller, Swabha Swayamdipta, Noah A. Smith, and Yejin Choi. We're afraid language models aren't modeling ambiguity, 2023. URL https://arxiv.org/abs/2304.14399.

Tianyu Liu, Yijia Xiao, Xiao Luo, Hua Xu, Wenjin Zheng, and Hongyu Zhao. Geneverse: A collection of open-source multimodal large language models for genomic and proteomic research. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4819–4836, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.277. URL https://aclanthology.org/2024.findings-emnlp.277/.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning, 2023. URL https://arxiv.org/abs/2301.13379.

Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. Linc: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5153–5176. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.313. URL http://dx.doi.org/10.18653/v1/2023.emnlp-main.313.

OpenAI, Josh Achiam, and Others. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Yuval Reif and Roy Schwartz. Beyond performance: Quantifying and mitigating label bias in llms. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6784–6798, 2024.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17 (3):261–272, 2020.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Fangzhi Xu, Qika Lin, Jiawei Han, Tianzhe Zhao, Jun Liu, and Erik Cambria. Are large language models really good logical reasoners? a comprehensive evaluation and beyond. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee, and Wynne Hsu. Faithful logical reasoning via symbolic chain-of-thought, 2024. URL https://arxiv.org/abs/2405.18357.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal mathematical reasoning: A new frontier in ai, 2024. URL https://arxiv.org/abs/2412.16075.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. SatLM: Satisfiability-aided language models using declarative prompting. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=TqW5PL1Poi.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Jiaxin Zhang, Zhongzhi Li, Mingliang Zhang, Fei Yin, Chenglin Liu, and Yashar Moshfeghi. Geoeval: benchmark for evaluating llms and multi-modal models on geometry problem-solving. *arXiv preprint arXiv:2402.10104*, 2024.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models, 2024a. URL `https://arxiv.org/abs/2310.06117`.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*, 2024b.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=WZH7099tgfM`.

Jin Peng Zhou, Charles Staats, Wenda Li, Christian Szegedy, Kilian Q. Weinberger, and Yuhuai Wu. Don't trust: Verify – grounding llm quantitative reasoning with autoformalization, 2024a. URL `https://arxiv.org/abs/2403.18120`.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. Self-discover: Large language models self-compose reasoning structures, 2024b. URL `https://arxiv.org/abs/2402.03620`.

Yukai Zhou, Zhijie Huang, Feiyang Lu, Zhan Qin, and Wenjie Wang. Don't say no: Jailbreaking llm by suppressing refusal. *arXiv preprint arXiv:2404.16369*, 2024c.

A Appendix / supplemental material

A.1 Implementation details

Prompting-based HybridMind. In our experiment, we considered prompting different LLMs with various instructions to work as a meta selector for generating correct solution based on the given question. Regarding the format of prompting, we tried two different zero-shot instruction design and made comparison. We also tried few-shot prompting design, whose number of examples (shots) in the instruction is at most 2. The shot contains question, answer, and the rationale generated by GPT-o3-mini.

Fine-tuning-based HybridMind. We also performed experiments to fine-tune Llama 3.1-8B to perform complicated reasoning tasks. We prepared the training datasets by sampling the choices and paired one correct choice with one question. The implementation of SFT is based on Llama-factory and the hyper-parameter setting is same as the example configuration template. The implementation of STaR is modified from the original code base. STaR utilizes models to generate rationale and then select the corret samples as well as the generated rationales to perform fine-tuning. To fine-tune our models, we utilize both NVIDIA H100 GPU and Together AI API. Details of our training codes can be found in the attached zip file to this submission.

Information of baseline models. In our experiment, we considered prompting closed-source LLMs, including GPT-4o, GPT-3.5-turbo, GPT-o3-mini, and prompting open-source LLM including Llama 3.1-8b. The prompting experiment is performed with different strategies (NL, SL, SymbolNL, NLSymbol) for solving the question directly. MV represents performing majority voting based on the methods' outputs. We also considered fine-tuning the base model with different strategies, including making binary choices (NL or SL), or making 4-class choices (NL, SL, SymbolNL, NLSymbol). The fine-tuning process is for making a meta selector.

A.2 Statistics of testing data

Please check Tables 5, 6, and 7.

Category	Algebra	Counting & Probability	Geometry	Number & Theory	Intermediate & Algebra	Precalculus	Prealgebra	Total
Number	1187	474	479	540	903	546	871	5000

Table 5: Number of samples in the testing set by categories for MATH.

	MATH		FOLIO	
Method	**Freq.**	**Prop.**	**Freq.**	**Prop.**
NL	4705	94.10%	133	58.84%
SL	124	2.48%	93	41.16%
SymbolNL	163	3.26%	-	-
NLSymbol	8	0.16%	-	-

Table 6: The frequency and proportion of selected choices by HYBRIDMIND on MATH and FOLIO *test* set. We implemented 4-choice selector for MATH dataset and 2-choice selector for FOLIO.

Category	`WikiLogic`	`HybLogic`	Total
Number	111	115	226

Table 7: Number of samples in the testing set by categories for FOLIO.

A.3 Prompts and shots

Below are the prompts and shots used for the four approaches: NL, SL, SymbolNL, and NLSymbol.

A.3.1 NL

```
nl_system_prompt = '''
You are a helpful assistant who is good at solving math problems. You
    should follow the guidelines below:
- Present the final result in LaTeX using a `\boxed{}` without any units.
- Utilize the `pi` symbol, and simplify all fractions and square roots
    without converting them to decimal values.
'''

nl_instruction_prompt = "Please␣think␣step␣by␣step.␣"

nl_math_shots= [
'''Question: Kevin Kangaroo begins hopping on a number line at 0. He
    wants to get to 1, but he can hop only $\frac{1}{3}$ of the distance.
     Each hop tires him out so that he continues to hop $\frac{1}{3}$ of
    the remaining distance. How far has he hopped after five hops?
    Express your answer as a common fraction.
Answer: Let's think step by step
Kevin hops $1/3$ of the remaining distance with every hop.
His first hop takes $1/3$ closer.
For his second hop, he has $2/3$ left to travel, so he hops forward $
    (2/3)(1/3)$.
For his third hop, he has $(2/3)^2$ left to travel, so he hops forward $
    (2/3)^2(1/3)$.
In general, Kevin hops forward $(2/3)^{k-1}(1/3)$ on his $k$th hop.
We want to find how far he has hopped after five hops.
```

```
This is a finite geometric series with first term $1/3$, common ratio $2
    /3$, and five terms.
Thus, Kevin has hopped $\frac{\frac{1}{3}\left(1-\left(\frac{2}{3}\right)
    ^5\right)}{1-\frac{2}{3}} = \boxed{\frac{211}{243}}$.
The answer is \frac{211}{243}''',


'''Question: What is the area of the region defined by the equation $x^2+
    y^2 - 7 = 4y-14x+3$?
Answer: Let's think step by step
We rewrite the equation as $x^2 + 14x + y^2 - 4y = 10$ and then complete
    the square,
resulting in  $(x+7)^2-49 + (y-2)^2-4=10$,
or $(x+7)^2+(y-2)^2=63$.
This is the equation of a circle with center $(-7, 2)$ and radius $\sqrt
    {63},$
so the area of this region is $\pi r^2 = \boxed{63\pi}$.
The answer is 63\pi''',


'''Question: If $x^2+y^2=1$, what is the largest possible value of $|x|+|
    y|$?
Answer: Let's think step by step
If $(x,y)$ lies on the circle,
so does $(x,-y),$ $(-x,-y),$ and $(-x,-y),$ (which all give the same
    value of $|x| + |y|$),
so we can assume that $x \ge 0$ and $y \ge 0.$
Then $|x| + |y| = x + y.$  Squaring, we get
\[(x + y)^2 = x^2 + 2xy + y^2 = 1 + 2xy.\]
Note that $(x - y)^2 \ge 0.$
Expanding, we get $x^2 - 2xy + y^2 \ge 0,$ so $2xy \le x^2 + y^2 = 1.$
Hence,\[1 + 2xy \le 2,\]which means $x + y \le \sqrt{2}.$
Equality occurs when $x = y = \frac{1}{\sqrt{2}},$
so the maximum value of $|x| + |y|$ is $\boxed{\sqrt{2}}.$
The answer is \sqrt{2}''',


'''Question: If $f(x)=\frac{ax+b}{cx+d}, abcd\not=0$ and $f(f(x))=x$ for
    all $x$ in the domain of $f$, what is the value of $a+d$?
Answer: Let's think step by step
The condition $f(f(x))$ means that $f$ is the inverse of itself,
so its graph is symmetrical about the line $y = x$.
With a rational function of this form, we will have two asymptotes:
a vertical one at $x=-d/c$ if $cx+d$ does not divide $ax+b$,
and a horizontal one at $y=a/c$,
if we take the limit of $f(x)$ as $x$ goes to $\pm\infty$.
In order for $f$ to be its own inverse, the intersection of the
    asymptotes must lie on the line $y=x$
so that it and its asymptotes reflect onto themselves.
This means that $-d/c=a/c$,
and therefore $-d=a$ and $a+d=\boxed{0}$.
The answer is 0''',


'''Question: A math teacher requires Noelle to do one homework assignment
     for each of the first five homework points she wants to earn; for
    each of the next five homework points, she needs to do two homework
    assignments; and so on, so that to earn the $n^{\text{th}}$ homework
    point, she has to do $n\div5$ (rounded up) homework assignments. For
    example, when she has 11 points, it will take $12\div5=2.4\
    rightarrow3$ homework assignments to earn her $12^{\text{th}}$ point.
     What is the smallest number of homework assignments necessary to
    earn a total of 25 homework points?
Answer: Let's think step by step
Noelle only has to do 1 homework assignment to earn her first point,
and the same is true for each of her first five points.
She must then do 2 homework assignments to earn her sixth point, seventh
    point, and so on, up to her tenth point.
Continuing, we see that Noelle must do a total of \[1+1+1+1+1+2+2+2+2+2+\
    dots+5+5+5+5+5\] homework assignments to earn 25 points.
This sum may be rewritten as $5(1+2+3+4+5)=5(15)=\boxed{75}$.
The answer is 75''',


'''Question: The quadratic equation $x^2+mx+n=0$ has roots that are twice
     those of $x^2+px+m=0,$ and none of $m,$ $n,$ and $p$ is zero. What
    is the value of $n/p?$
Answer: Let's think step by step
Let $r_1$ and $r_2$ be the roots of $x^2+px+m=0.$
Since the roots of $x^2+mx+n=0$ are $2r_1$ and $2r_2,$ we have the
    following relationships: \[
m=r_1 r_2,\quad n=4r_1 r_2,\quad p=-(r_1+r_2), \quad\text{and}\quad
m=-2(r_1+r_2).
\] So \[
n = 4m, \quad p = \frac{1}{2}m,
\quad\text{and}\quad
\frac{n}{p}=\frac{4m}{\frac{1}{2}m}=\boxed{8}.
\]
Alternatively, the roots of \[
\left(\frac{x}{2}\right)^2 + p\left(\frac{x}{2}\right) + m = 0
\] are twice those of $x^2 + px + m = 0.$
Since the first equation is equivalent to $x^2 + 2px + 4m = 0,$
we have \[m = 2p \quad\text{and}\quad n = 4m, \quad\text{so}\quad \frac{n
    }{p} = \boxed{8}.\]
The answer is 8''',

'''Question: Expand $(2z^2 + 5z - 6)(3z^3 - 2z + 1)$.
Answer: Let's think step by step
$$\begin{array}{crrrrrrr}
& & & 3z^3 & & -2z & + 1 & \\
\times & & & & 2z^2 & +5z & -6 \\
\cline{1-7}\rule{0pt}{0.17in}
& & & -18z^3 & & +12z & -6 & \\
& & +15z^4 & & -10z^2 & +5z & & \\
+ & 6z^5 & & -4z^3 & +2z^2 & & & \\
\cline{1-7}\rule{0pt}{0.17in}
& 6z^5 & +15z^4 & -22z^3 & - 8z^2 &+17z & -6 &
\end{array}$$
The answer is 6z^5+15z^4-22z^3-8z^2+17z-6''',


'''Question: Find the mean of all solutions for $x$ when $x^3 + 3x^2 - 10
    x = 0$.
Answer: Let's think step by step
First, we factor the equation as $x(x^2 +3x - 10) = 0$.
So, one solution is $x=0$ and the other two solutions are the solutions
    to $x^2 + 3x-10=0$.
We could either factor the quadratic, or note that the sum of the
    solutions to this quadratic is $-(3/1)=-3$,
so the mean of the three solutions to the original equation is $-3/3=\
    boxed{-1}$.
The answer is -1
''',
]
```

A.3.2 SL

~~~
sl_system_prompt = '''
You are a helpful assistant who is good at sloving math problems and
    writing code. You should should follow the guidelines below:
- Utilize the `pi` symbol and `Rational`` from Sympy for $\pi$ and
    fractions, and simplify all fractions and square roots without
    converting them to decimal values
- You should only write code blocks and the function name should be `
    solution` and the returned value should be the final answer.
'''

sl_instruction_prompt = "Let's␣use␣python␣to␣solve␣the␣math␣problem.␣"


sl_math_shots= [
'''Question: Find the coefficient of $x^3$ when $3(x^2 - x^3+x) +3(x +2x
    ^3- 3x^2 + 3x^5+x^3) -5(1+x-4x^3 - x^2)$ is simplifie.

```python
from sympy import symbols, simplify

def solution():
    """Find the coefficient of $x^3$ when $3(x^2 - x^3+x) +3(x +2x^3- 3x
        ^2 + 3x^5+x^3) -5(1+x-4x^3 - x^2)$ is simplified."""
    x = symbols('x')
    expr = 3*(x**2 - x**3 + x) + 3*(x + 2*x**3 - 3*x**2 + 3*x**5 + x**3)
        - 5*(1 + x - 4*x**3 - x**2)
    simplified_expr = simplify(expr)

    x3_coefficient = simplified_expr.as_coefficients_dict()[x**3]
    result = x3_coefficient
    return result
```''',

'''Question: The surface area of a sphere with radius $r$ is $4\pi r^2$.
    Including the area of its circular base, what is the total surface
    area of a hemisphere with radius 6 cm? Express your answer in terms
    of $\pi$.

```python
import math

def solution():
    """The surface area of a sphere with radius $r$ is $4\pi r^2$.
        Including the area of its circular base, what is the total
        surface area of a hemisphere with radius 6 cm? Express your
        answer in terms of $\pi$"""
    radius = 6

    # Surface area of the hemisphere
    hemisphere_area = 2 * math.pi * radius**2

    # Area of the circular base
    base_area = math.pi * radius**2

    # Total surface area
    total_surface_area = hemisphere_area + base_area

    # Formatting the result in LaTeX
    result = r'{}\\pi'.format(total_surface_area / math.pi)
    return result
```''',

'''Question: Monica tosses a fair 6-sided die.  If the roll is a prime
    number, then she wins that amount of dollars (so that, for example,
    if she rolls 3, then she wins 3 dollars).  If the roll is composite,
~~~

~~~
    she wins nothing. Otherwise, she loses 3 dollars. What is the
    expected value of her winnings on one die toss? Express your answer
    as a dollar value to the nearest cent.

```python
def solution():
    """Monica tosses a fair 6-sided die.  If the roll is a prime number,
        then she wins that amount of dollars (so that, for example, if
        she rolls 3, then she wins 3 dollars).  If the roll is composite,
         she wins nothing. Otherwise, she loses 3 dollars. What is the
        expected value of her winnings on one die toss? Express your
        answer as a dollar value to the nearest cent."""
    # Probabilities of each outcome
    prime_prob = 1 / 6
    composite_prob = 1 / 3
    otherwise_prob = 1 / 6

    # Expected value of each outcome
    prime_expected_value = (2 * prime_prob) + (3 * prime_prob) + (5 *
        prime_prob)
    composite_expected_value = 0 * composite_prob
    otherwise_expected_value = -3 * otherwise_prob

    # Total expected value
    total_expected_value = prime_expected_value +
        composite_expected_value + otherwise_expected_value

    # Dollar value to the nearest cent
    result = "{:.2f}".format(total_expected_value)
    return result
```''',

'''Question: Given $\mathbf{a} = \begin{pmatrix} -7 \\ 0 \\ 1 \end{
    pmatrix}$ and $\mathbf{b} = \begin{pmatrix} 4 \\ 2 \\ -1 \end{pmatrix
    },$ find $\mathbf{a} - 3 \mathbf{b}.$

Solution:
```python
import numpy as np

def solution()
    """Given $\mathbf{a} = \begin{pmatrix} -7 \\ 0 \\ 1 \end{pmatrix}$
        and $\mathbf{b} = \begin{pmatrix} 4 \\ 2 \\ -1 \end{pmatrix},$
        find $\mathbf{a} - 3 \mathbf{b}.$"""
    a = np.array([-7, 0, 1])
    b = np.array([4, 2, -1])

    result = a - 3 * b

    result = r'\begin{{pmatrix}} {} \\ {} \\ {} \end{{pmatrix}}'.format(
        result[0], result[1], result[2])
    return result
```''',

'''Question: The endpoints of a diameter of circle $M$ are $(-1,-4)$ and
    $(-7,6)$. What are the coordinates of the center of circle $M$?
    Express your answer as an ordered pair.
```python
def solution():
    """The endpoints of a diameter of circle $M$ are $(-1,-4)$ and $
        (-7,6)$. Find the coordinates of the center of circle $M$."""
    x1, y1 = -1, -4
    x2, y2 = -7, 6

    # Midpoint formula
~~~

~~~
    center_x = (x1 + x2) / 2
    center_y = (y1 + y2) / 2

    # Result as an ordered pair
    result = (center_x, center_y)
    return result
```''',

'''Question: Find the remainder when $2x^6-x^4+4x^2-7$ is divided by $x
    ^2+4x+3$.
```python
from sympy import symbols, div

def solution():
    """Find the remainder when $2x^6-x^4+4x^2-7$ is divided by $x^2+4x+3$
        ."""
    x = symbols('x')
    numerator = 2*x**6 - x**4 + 4*x**2 - 7
    denominator = x**2 + 4*x + 3

    quotient, remainder = div(numerator, denominator)

    return remainder
''',

'''Question: Find $x$ so that the vectors $\begin{pmatrix} 2 \\ 5 \end{
    pmatrix}$ and $\begin{pmatrix} x \\ -3 \end{pmatrix}$ are orthogonal.
```python
from sympy import symbols, Eq, solve

def solution():
    """Find x so that the vectors (2, 5) and (x, -3) are orthogonal."""
    x = symbols('x')
    # Dot product of the vectors should be zero for orthogonality
    equation = Eq(2*x + 5*(-3), 0)
    result = solve(equation, x)
    return result[0]
```''',
'''Question: If $x$ is positive and $x^2 = 729$, what is the value of $x$
    ?
```python
from sympy import symbols, Eq, solve

def solution():
    """If $x$ is positive and $x^2 = 729$, what is the value of $x$?"""
    x = symbols('x')
    equation = Eq(x**2, 729)
    solutions = solve(equation, x)

    # Since x is positive, we select the positive solution
    result = [sol for sol in solutions if sol > 0][0]
    return result
```''',
]
~~~

A.3.3 Second stage of SymbolNL

````
symbolnl_instructio_prompt = "Please␣think␣step␣by␣step␣about␣the␣
    question␣based␣on␣the␣provided␣code(it␣may␣be␣wrong)␣and␣the␣executed
    ␣output.␣"

symbolnl_math_shots = [
'''Question: How many distinct positive factors does 32 have?
Code: ```python
````

~~~
def solution():
    """How many distinct positive factors does 32 have?"""
    number = 32
    factors = set()

    for i in range(1, int(number**0.5) + 1):
        if number % i == 0:
            factors.add(i)
            factors.add(number // i)

    result = len(factors)
    return result
```
Output: 6
Answer: To determine the number of distinct positive factors of 32, we
    can follow these steps:

1. Prime Factorization:
   - First, we find the prime factorization of 32.
   - \( 32 = 2^5 \)

2. Number of Factors:
   - The formula to find the number of distinct positive factors of a
       number based on its prime factorization is to add 1 to each of the
        exponents in the factorization and then multiply the results.
   - For \( 32 = 2^5 \), the exponent is 5.
   - Adding 1 to the exponent: \( 5 + 1 = 6 \).

3. Conclusion:
   - Therefore, the number of distinct positive factors of 32 is 6.

Thus, the final result is:
\[\boxed{6}\]''',

'''Question: In parallelogram $ABCD$, angle $B$ measures $110^\circ$.
    What is the number of degrees in the measure of angle $C$?
Code: ```python
def solution():
    """In parallelogram $ABCD$, angle $B$ measures $110^\circ$. What is
        the number of degrees in the measure of angle $C$?"""
    angle_B = 110

    # In a parallelogram, opposite angles are equal and adjacent angles
        are supplementary
    angle_C = 180 - angle_B

    return angle_C
```
Output: 70
Answer: In a parallelogram, opposite angles are equal and adjacent angles
     are supplementary. Given that angle \( B \) measures \( 110^\circ \)
    , we need to find the measure of angle \( C \).

Step-by-step reasoning:
1. In a parallelogram, adjacent angles are supplementary. This means that
     the sum of the measures of angle \( B \) and angle \( C \) is \(
    180^\circ \).
2. Therefore, we can write the equation: \( \text{angle } B + \text{angle
     } C = 180^\circ \).
3. Substituting the given measure of angle \( B \) into the equation: \(
    110^\circ + \text{angle } C = 180^\circ \).
4. Solving for angle \( C \): \( \text{angle } C = 180^\circ - 110^\circ
    = 70^\circ \).

Thus, the measure of angle \( C \) is \( \boxed{70} \).''',
~~~

```
'''Question: In triangle $ABC$, we have that $E$ and $F$ are midpoints of
     sides $\overline{AC}$ and $\overline{AB}$, respectively. The area of
     $\triangle ABC$ is 24 square units. How many square units are in the
     area of $\triangle CEF$?
Code: ```python
from sympy import Rational

def solution():
    """Calculate the area of triangle CEF given that E and F are
        midpoints of sides AC and AB respectively, and the area of
        triangle ABC is 24 square units."""
    # Area of triangle ABC
    area_ABC = 24

    # Triangle CEF is a median triangle, which is 1/4 the area of
        triangle ABC
    area_CEF = Rational(1, 4) * area_ABC

    return area_CEF
```
Output: 6
Answer: The problem involves finding the area of triangle $CEF$ where $E$
     and $F$ are midpoints of sides $\overline{AC}$ and $\overline{AB}$
    respectively in triangle $ABC$. Given that the area of triangle $ABC$
     is 24 square units, we need to determine the area of triangle $CEF$.

Since $E$ and $F$ are midpoints of their respective sides, line segment $
    \overline{EF}$ is parallel to side $\overline{BC}$ and is half its
    length. This makes triangle $CEF$ a median triangle of triangle $ABC$
    . A median triangle is one-fourth the area of the original triangle
    because it is formed by drawing lines from each vertex of the
    original triangle to the midpoints of the opposite sides.

Thus, the area of triangle $CEF$ is calculated as:
\[\text{Area of } \triangle CEF = \frac{1}{4} \times \text{Area of } \
    triangle ABC = \frac{1}{4} \times 24\]

Simplifying this, we get:
\[\text{Area of } \triangle CEF = 6\]

Therefore, the area of triangle $CEF$ is \(\boxed{6}\) square units.''',

'''Question: What is the greatest two-digit multiple of 13?
Code: ```python
def solution():
    """Find the greatest two-digit multiple of 13."""
    # Start from the greatest two-digit number and go downwards
    for number in range(99, 12, -1):
        if number % 13 == 0:
            return number
```
Output: 91
Answer: To find the greatest two-digit multiple of 13, we start by
    considering the largest two-digit number, which is 99. We need to
    find the largest number less than or equal to 99 that is divisible by
     13.

We perform the division of 99 by 13 to find the largest integer quotient:
\[ \frac{99}{13} \approx 7.615 \]
The integer part of this division is 7, indicating that the largest
    multiple of 13 within the two-digit range is \( 13 \times 7 \).

Calculating this product:
```

```
\[ 13 \times 7 = 91 \]

Thus, the greatest two-digit multiple of 13 is 91. The Python code
    provided confirms this by iterating from 99 downwards and checking
    divisibility by 13, stopping at the first number that satisfies this
    condition, which is 91.

Therefore, the final answer is:
\[ \boxed{91} \]''',
]
```

A.3.4 Second stage of NLSymbol

~~~
nlsymbol_instruction_prompt = "Write␣a␣Python␣code␣that␣translates␣a␣
    natural␣language␣(NL)␣reasoning␣path␣into␣executable␣code␣to␣answer␣a
    ␣given␣question.␣The␣output␣of␣the␣generated␣code␣should␣be␣the␣final
    ␣answer␣to␣the␣question.␣"

nlsymbol_math_shots = [
'''Question: The area of the floor in a square room is 225 square feet.
    The homeowners plan to cover the floor with rows of 6-inch by 6-inch
    tiles. How many tiles will be in each row?
Reasoning Path: Let's think step by step:

The area of the floor is given as 225 square feet. Since the room is
    square, each side of the room is the square root of the area. Thus,
    each side of the room is \(\sqrt{225} = 15\) feet.

Each tile is 6 inches by 6 inches, which is equivalent to \(0.5\) feet by
     \(0.5\) feet since there are 12 inches in a foot.

To find out how many tiles fit along one side of the room, we divide the
    length of the room by the length of one tile:
\[
\frac{15 \text{ feet}}{0.5 \text{ feet/tile}} = 30 \text{ tiles}
\]

Therefore, there will be \(\boxed{30}\) tiles in each row.
Code: ```python
from sympy import sqrt

def solution():
    # Area of the floor in square feet
    area = 225

    # Since the room is square, calculate the side length of the room
    side_length = sqrt(area)

    # Each tile's side length in feet (6 inches = 0.5 feet)
    tile_length = 0.5

    # Calculate the number of tiles in each row
    tiles_per_row = side_length / tile_length

    return tiles_per_row

```''',
'''Question: When the set of natural numbers is listed in ascending order
    , what is the smallest prime number that occurs after a sequence of
    five consecutive positive integers all of which are nonprime?
Reasoning Path: Let's think step by step:

~~~

~~~
To find the smallest prime number that occurs after a sequence of five
    consecutive nonprime positive integers, we need to identify such a
    sequence first.

1. Start by checking small numbers. The first few nonprime numbers are 4,
     6, 8, 9, and 10. However, these do not form a sequence of five
    consecutive nonprime numbers.

2. Continue checking further:
   - 8, 9, 10, 12, 14 are nonprime, but they are not consecutive.
   - 24, 25, 26, 27, 28 are nonprime and consecutive.

3. Check the number following 28:
   - 29 is the next number, and it is a prime number.

Thus, the smallest prime number that occurs after a sequence of five
    consecutive nonprime positive integers (24, 25, 26, 27, 28) is 29.

The answer is \(\boxed{29}\).
Code: ```python
from sympy import isprime

def solution():
    # Start checking from a reasonable number
    n = 1
    while True:
        # Check if the next five numbers are all nonprime
        if all(not isprime(n + i) for i in range(5)):
            # Check the number right after the sequence of five nonprimes
            if isprime(n + 5):
                return n + 5
        n += 1
```''',

'''Question: What is the slope of the line that is tangent to a circle at
     point (5,5) if the center of the circle is (3,2)? Express your
    answer as a common fraction.
Reasoning Path: Let's think step by step:

The slope of the line connecting the center of the circle at point
    \((3,2)\) to the point on the circle \((5,5)\) is given by the
    formula for the slope between two points:
\[m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{5 - 2}{5 - 3} = \frac{3}{2}\]

The line that is tangent to the circle at the point \((5,5)\) will be
    perpendicular to the line connecting the center of the circle to this
     point. The slope of a line perpendicular to another line with slope
    \(m\) is the negative reciprocal of \(m\). Therefore, the slope of
    the tangent line is:
\[m_{\text{tangent}} = -\frac{1}{\frac{3}{2}} = -\frac{2}{3}\]

Thus, the slope of the line that is tangent to the circle at point
    \((5,5)\) is \(\boxed{-\frac{2}{3}}\).
Code: ```python
from sympy import Rational

def solution():
    # Slope of the line connecting the center of the circle to the point
        (5,5)
    slope_radius = Rational(5 - 2, 5 - 3)

    # Slope of the tangent line, which is the negative reciprocal of the
        slope of the radius
    slope_tangent = -1 / slope_radius

~~~

~~~
    return slope_tangent

```''',

'''Question: What is the greatest integer less than 100 for which the
    greatest common factor of that integer and 18 is 3?
Reasoning Path: Let's think step by step

We are looking for the greatest integer less than 100 for which the
    greatest common factor (GCF) with 18 is 3.

First, note that 18 can be factored into prime factors as \(18 = 2 \times
     3^2\).

For the GCF of a number \(n\) and 18 to be 3, \(n\) must:
1. Be divisible by 3 (to include the factor of 3).
2. Not be divisible by 2 or 9 (to avoid increasing the GCF beyond 3).

We need to find the largest integer less than 100 that meets these
    criteria. We look for numbers that are multiples of 3 but not
    multiples of 2 or 9.

The largest multiple of 3 under 100 is 99. We check if it is divisible by
     2 or 9:
- 99 is not divisible by 2 (since it is odd).
- 99 is divisible by 9 (since \(9 + 9 = 18\), and 18 is divisible by 9).

Since 99 does not work (as it is divisible by 9), we check the next
    largest multiple of 3, which is 96.
- 96 is divisible by 2 (even number), so it does not work.

Next, we check 93:
- 93 is not divisible by 2 (odd number).
- 93 is not divisible by 9 (since \(9 + 3 = 12\), and 12 is not divisible
     by 9).

Thus, 93 meets the criteria of being divisible by 3 but not by 2 or 9.
    Therefore, the greatest integer less than 100 for which the GCF with
    18 is 3 is \(\boxed{93}\).
Code ```python
def solution():
    from math import gcd
    # Start from the largest number less than 100 and check downwards
    for n in range(99, 0, -1):
        if gcd(n, 18) == 3:
            return n

# The function will return the greatest integer less than 100 for which
    the GCD with 18 is 3
```''',
]
~~~

A.4 Significance test

Method	**Significance Level**
GPT-3.5-Turbo compare prompt (less)	¡ 0.05
GPT-4o compare prompt (less)	¡ 0.05
o3-mini prompt (less)	0.257
HybridMind (more)	¡ 0.05

Table 8: Testing of significance levels of MATH dataset between HybridMind and NL.

Method	**Significance Level**
GPT-4o compare prompt (less)	0.080
o3-mini prompt (less)	¡ 0.05
HYBRIDMIND (more)	¡ 0.05

Table 9: Testing of significance levels of FOLIO dataset between HYBRIDMIND and NL.

A.5 Analysis between SL and NL

Please check Tables10, 11, 12, and 13.

A.6 Meta-Selection prompt

Here is 2-class meta-selection prompt for MATH dataset.

```
two_class_shots = [
    '''
    Question: Four standard, six-sided dice are to be rolled. If the
        product of their values turns out to be an even number, what is
        the probability their sum is odd? Express your answer as a common
         fraction.

    Response: Below is an evaluation of the two methods for this specific
         four-dice probability problem:

    Natural LLanguage (Language Only):
    Strengths: This method lets you break the problem down step by step.
        You can explain your reasoning about conditionals (the even
        product and odd sum) and manually handle the combinatorial cases.
         This approach can offer insight into why the answer is what it
        is.
    Weaknesses: It requires careful bookkeeping to avoid mistakes in
        counting cases and may become tedious when managing several
        conditions simultaneously. There is more risk of a small
        oversight that leads to an error in the final probability.

    Symbolic Language (Coding Only):
    Strengths: By writing a short Python program to iterate over all 6ˆ4
        possible outcomes, you can systematically and accurately count
        the desired cases. It is very effective for this problem because
        the search space is small enough for exhaustive enumeration,
        thereby reducing human error.
    Weaknesses: It might seem like overkill for a problem that is
        solvable by combinatorial reasoning, and you must be careful to
        correctly implement the conditions in code. However, given the
        manageable sample space, this method is straightforward and
        reliable.

    For this specific question, because the number of outcomes is small
        and the conditions can be programmed explicitly, the program-
        aided approach minimizes error risk and offers a quick,
        systematic solution.

    Selected method: SL (Symbolic Language, Coding Only)

''',
    '''
    Question: A piece of cheese is located at $(12,10)$ in a coordinate
        plane.  A mouse is at $(4,-2)$ and is running  up  the line $y=-5
        x+18$.  At the point $(a,b)$ the mouse starts getting farther
        from the cheese rather than closer to it. What is $a + b$?
    Response: Below is an evaluation of each approach for this particular
         problem:

    Natural Language (NL):
```

Question

For how many integers a satisfying $1 \leq a \leq 23$ is it true that $a^{-1} \equiv a \pmod{24}$?

NL

Let's think step by step. We need to find integers a such that $1 \leq a \leq 23$ and $a^{-1} \equiv a \pmod{24}$. This condition implies that $a \cdot a \equiv 1 \pmod{24}$, or equivalently, $a^2 \equiv 1 \pmod{24}$. This congruence can be rewritten as $a^2 - 1 \equiv 0 \pmod{24}$, which factors as $(a-1)(a+1) \equiv 0 \pmod{24}$. Therefore, $(a-1)(a+1)$ must be divisible by 24. The prime factorization of 24 is $2^3 \times 3$. Thus, $(a-1)(a+1)$ must be divisible by $2^3 \times 3$. Since $a-1$ and $a+1$ are consecutive integers, one of them is even, and the other is odd. Therefore, one of them must be divisible by 4, and the other by 2, to ensure divisibility by $2^3 = 8$. Additionally, one of $a-1$ or $a+1$ must be divisible by 3.
We solve these systems of congruences using the Chinese Remainder Theorem:
Case 1: …… $a \equiv 17 \pmod{24}$
Case 2: …… $a \equiv 7 \pmod{24}$
Thus, there are two integers a that satisfy the condition: $a = 7$ and $a = 17$. Therefore, the number of integers a is 2.

SL

Python

```
from sympy import symbols, Eq, solve, mod_inverse
def solution():
  count = 0
  for a in range(1, 24):
    try:
      # Calculate the modular inverse of a modulo 24
      a_inv = mod_inverse(a, 24)
      # Check if the modular inverse is equal to a
      if a_inv == a:
        count += 1
    except ValueError:
      # If a modular inverse does not exist, continue
      continue
  return count
solution()
```

Executed Result

8

Table 10: An example where SL outperforms NL. While NL reasoning leads to logical errors in analysis and overlooks some possible scenarios, resulting in incorrect conclusions, writing Python code is more elegant and intuitive.

```
    Strengths:
    - It is especially useful for problems that rely on geometric insight
         and visual reasoning. For instance, one can break down the
        problem by first finding the point on the line that is closest to
         the cheese (using ideas like the perpendicular from the cheese
        to the mouse's path) and then compute the coordinates, which
```

Question

How many days in a year have a 3 in the tens digit of the day of the month?

NL

Let's think step by step. We need to count the number of days in a year where the tens digit of the day of the month is 3. This means we are looking for days numbered 30 through 39 in each month.
1. Identify the months with days 30-39:
- The months with at least 30 days are January, March, April, May, June, July, August, September, October, November, and December.
- February has only 28 or 29 days, so it does not have any days with a tens digit of 3.
2. Count the days for each applicable month:
-For each of the months with at least 30 days, the days 30 and 31 have a tens digit of 3.
- Therefore, each of these months contributes 2 days (30 and 31) to our count.
3. Calculate the total number of such days:
- There are 11 months with at least 30 days.
- Each of these months contributes 2 days.
- Total number of days = $11 \times 2 = 22$. Thus, the number of days in a year that have a 3 in the tens digit of the day of the month is 22.

SL

Python

```
def solution():
    # Not all months have 31 days, so we need to count carefully
    # Months with 31 days: January, March, May, July, August, October, December
    months_with_31_days = 7
    # Months with 30 days: April, June, September, November
    months_with_30_days = 4
    # February has 28 or 29 days, so it never has a 30th or 31st
    # Days with a 3 in the tens digit:
    # - 30th: occurs in every month except February
    # - 31st: occurs only in months with 31 days
    days_with_3_in_tens_digit = (months_with_30_days + months_with_31_days) + months_with_31_days
    return days_with_3_in_tens_digit
solution()
```

Executed Result

18

Table 11: An example where SL outperforms NL. Python code offers a more accurate and structured approach. While NL reasoning overlooks some edge cases and leads to incorrect conclusions, SL systematically handles the details and produces the correct result.

```
        directly leads to a conceptual understanding of why that point
        minimizes the distance.
    - The natural language breakdown allows the solver to easily explain,
         verify, and understand each step in the geometric reasoning
        process.
    Weaknesses:
```

```
- It relies on the solver's ability to carefully reason through each
    step without skipping important details. Errors might occur if a
    step is overlooked or misinterpreted, but these issues are often
    easier to spot and correct when the reasoning is laid out in
    natural language.

Symbolic Language (SL):
Strengths:
- A Python program can be very effective for computing an answer if
    one sets up the right system (for example, defining the distance
    function and finding its minimum via analytic methods or
    optimization). It is useful if one has a routine set of tools (
    like computational geometry libraries or symbolic manipulators)
    that can automate the calculation accurately.
- It minimizes the potential for simple arithmetic mistakes, as the
    computer does the heavy lifting once the input is correctly
    specified.
Weaknesses:
- Setting up the correct algorithm in code (especially for a geometry
     problem that involves minimizing a distance) can be nontrivial.
    One must translate the geometric intuition correctly into code,
    and a mistake there might lead to an incorrect answer without an
    immediately clear explanation.
- The program might properly produce a number but without conveying
    the underlying geometric reasoning, which can reduce the
    educational value of the solution process for problems like this.

For this specific problem-a geometry problem involving distances and
    perpendicular intersections-the Chain-of-Thought method is likely
     the more effective approach. It promotes understanding of the
    underlying geometric reasoning and highlights the process behind
    finding the point where the mouse's distance from the cheese is
    minimized.

Selected method: NL (Natural Language, Language Only)''',
'''
Question: Find the minimum of the function
\[\frac{xy}{x^2 + y^2}\]in the domain $\frac{2}{5} \le x \le \frac
    {1}{2}$ and $\frac{1}{3} \le y \le \frac{3}{8}.$
Response: Let's review what each approach offers for this problem.

Natural Lauguage (NL) method:
Strengths: It allows step-by-step reasoning, considering the
    structure of the algebraic expression and boundary behavior,
    which may reveal insights such as where the minimum might occur (
    for example, at the endpoints or at a critical point) and can be
    done entirely by hand.
Weaknesses: The algebra can become quite messy given the rational
    function and the two-dimensional domain. Tracking all possible
    cases (including boundary and potential critical points)
    increases the risk of error or oversight. The method relies on
    human intuition and careful logical decomposition, which can be
    challenging under time constraints.

Symbolic Language (SL) method:
Strengths: A Python program can brute-force or use well-tested
    optimization routines to accurately sample the function over the
    given domain. Because the domain is compact and small in extent,
    a numerical search (or even more refined optimization) is
    straightforward and likely to yield a correct answer with less
    chance of human error.
Weaknesses: One must be careful to cover the entire domain correctly
    and account for potential issues with numerical precision.
    Additionally, writing the optimization code correctly also
```

```
        requires some programming skill, though for a two-variable
        bounded problem this is well within reach.

    For this specific question-finding the minimum of a rational function
         over a small, compact domain-the program-aided approach is more
        directly effective. It reduces the intellectual overhead of
        dealing with all the subtle boundary and interior cases manually
        and tends to be more reliable given the tight constraints.

    Selected method: SL (Symbolic Language, Coding Only)
    ''',
    '''
    Question: Someone observed that $6! = 8 \cdot 9 \cdot 10$. Find the
        largest positive integer $n$ for which $n!$ can be expressed as
        the product of $n - 3$ consecutive positive integers.
    Response: For this particular problem the trade-offs are as follows:

     Natural Language (NL) Strengths and Weaknesses:
   - Strength: A step-by-step reasoning process can provide insight into
       the structure of the equation (relating factorials to products of
       sequences) and might lead to an elegant, human-readable argument.
     Weakness: The reasoning becomes quite intricate because one needs to
          relate rapidly growing factorials to a product of a long chain
         of consecutive integers. Handling large numbers and justifying
         bounds without resorting to heavy computations is tricky and
         error-prone for this specific task.

     Symbolic Language (SL) Strengths and Weaknesses:
    - Strength: A computer program can iterate through candidate values
        of n (and compute factorials and products of consecutive integers
        ) with precision. This method offloads the heavy number crunching
         and lets one quickly test many cases, which is especially
        valuable given the enormous growth of factorials.
    - Weakness: A coding solution requires careful handling of very large
         numbers (or the use of logarithms/approximations) to avoid
        computational inefficiencies or overflow issues. Still, with
        modern programming languages and libraries this issue can be
        managed.

    For this specific question, where the answer hinges on a precise
        numerical property and where direct computational verification
        seems feasible (once the search boundaries are known or estimated
        ), the Symbolic Language (SL) approach is more effective. It
        allows for a systematic exploration of candidates with the
        precision that the problem demands without running into lengthy
        manual derivations.

    Selected method: SL (Symbolic Language, Coding Only)''',

    '''Question: Evaluate $(2-w)(2-w^2)\cdots(2-w^{10})$ where $w=e^{2\pi
         i/11}.$
    Response: Below is an evaluation of both approaches for this problem:

    Natural Language (nl, Language Only):
    - Strengths: This method allows one to recognize the structure of the
         product. In this particular problem the factors are (2 - w^k)
        for the nontrivial 11th roots of unity, and one can quickly
        connect this to the cyclotomic polynomial formula (x^11 - 1)/(x -
         1). This insight leads directly to an exact, elegant solution
        without resorting to numerical approximation.
    - Weaknesses: Using pure logical reasoning requires a strong
        familiarity with algebraic properties of roots of unity and
        cyclotomic polynomials. If one's background is less strong in
        these areas, there is a risk of overlooking key steps or making
        errors in derivation.
    Symbolic Language (SL, Coding Only):
    - Strengths: A Python program (or similar) can compute the product
        numerically, providing a quick verification of the result. It
        automatically handles the multiplication of many complex numbers,
         which is useful when manual calculation is cumbersome.
    - Weaknesses: The numerical approach may introduce floating-point
        rounding errors, especially when working with complex
        exponentials. It also may hide the underlying structure of the
        problem and does not provide the exact algebraic insight that
        leads to a neat closed-form answer. Additionally, setting up the
        program correctly requires careful implementation of complex
        arithmetic.

    For this specific question, since the underlying algebraic structure
        (via cyclotomic polynomials) offers an elegant, exact answer, the
         chain-of-thought method is more effective and insightful.

    Selected method: NL (Natural Language, Language Only)
    ''',
    '''
    Question: In trapezoid $ABCD$ the lengths of the bases $AB$ and $CD$
        are 8 and 17 respectively. The legs of the trapezoid are extended
         beyond $A$ and $B$ to meet at point $E$. What is the ratio of
        the area of triangle $EAB$ to the area of trapezoid $ABCD$?
        Express your answer as a common fraction.
    Response: For this trapezoid geometry problem, the  NL method allows
        you to gradually set up the problem by assigning variables,
        drawing relationships (such as the similarity of triangles), and
        then carefully deriving the necessary formulas. This process can
        enhance understanding and make it easier to check each individual
         step, especially when dealing with geometric properties and
        ratios.

    On the other hand, the SL method would involve writing a Python
        program to symbolically manipulate or numerically approach the
        problem. While Python can be very powerful, setting up the
        symbolic algebra needed for a clear derivation of an algebraic
        ratio may require additional packages or careful coding. This
        approach might risk making the solution less transparent and
        could introduce potential programming errors for a problem that
        is more elegantly handled through traditional reasoning.

    Given that this problem benefits from step-by-step geometric
        reasoning and clear algebraic manipulation, the chain-of-thought
        method is more effective and straightforward for this specific
        question.

    Selected method: NL (Natural Language, Language Only)
    ''',
    '''
    Question: A 4-inch by 6-inch picture is enlarged for framing  by
        tripling its dimensions.  A 2-inch-wide border  is then placed
        around each side of the enlarged  picture, as shown.  Thin metal
        framing is sold only  in increments of one foot.  What is the
        minimum  number of linear feet of framing that must be  purchased
         to go around the perimeter of the border?

[asy]

draw((0,0)--(14,0)--(14,20)--(0,20)--cycle,linewidth(2));

draw((4,4)--(10,4)--(10,16)--(4,16)--cycle);

label("border",(7,17),N);

label("picture",(7,8),N);

label("frame",(14,5),E);

draw((17.5,7.5)--(14.5,7.5),Arrow);
draw((10.5,7.5)--(13.5,7.5),Arrow);

[/asy]
Response: Below is an evaluation of the two methods for this particular
    problem, followed by the selected method.

Natural Language (NL) - Language Only:
Strengths:
- It lets the solver reason through each step, clarifying the enlargement
     process, how the border is added to both dimensions, and the
    conversion from inches to feet.
- The explanation can detail the arithmetic (tripling dimensions, adding
    2 inches on each side for the border, computing the new perimeter,
    and then converting inches to feet while accounting for the need to
    round up) so that every piece of the problem is understood.
- This method can easily communicate the reasoning to others.
Weaknesses:
- If not careful, details might be overlooked or arithmetic errors made.
- The reliance on manual computations (even if done carefully) might lead
     to mistakes if any step is not double-checked.

Symbolic Language (SL) - Coding Only:
Strengths:
- A short Python program can accurately perform the arithmetic once the
    proper logic is implemented.
- Code can handle unit conversion and rounding (for example, using math.
    ceil) without any miscalculation due to human error.
- It is especially useful when the operations are straightforward and
    mostly numerical, as in this case.
Weaknesses:
- The problem requires careful interpretation (tripling the dimensions,
    adding a border to both sides) which must be encoded correctly; any
    misinterpretation could lead to a coding error.
- Writing code for such a simple arithmetic problem might be overkill and
     could miss conveying the geometric understanding behind the steps.
- There can also be distraction in the form of programming syntax rather
    than focusing on the underlying mathematics.

For THIS SPECIFIC QUESTION, the arithmetic and geometry are simple and
    direct. It is crucial to make sure that each conceptual step (
    enlargement, addition of borders, perimeter computation, and
    conversion unit) is clearly understood and communicated. The chain-of
    -thought method naturally lends itself to this clarity without the
    overhead of translating the problem into code. While coding could
    give the answer reliably, it may hide the reasoning steps that are
    important to verify each part of the computation.

Selected method: NL (Natural Language, Language Only)
''',
]
```

Question

Find the product of all positive integer values of c such that $3x^2 + 7x + c = 0$ has two real roots.

NL

To determine the product of all positive integer values of c such that the quadratic equation $3x^2 + 7x + c = 0$ has two real roots, we need to ensure that the discriminant of the quadratic is positive. The discriminant Δ of a quadratic equation $ax^2 + bx + c = 0$ is given by:

$$\Delta = b^2 - 4ac$$

……

Solving this inequality for c, we get:

$$49 > 12c \quad \Rightarrow \quad c < \frac{49}{12}$$

Calculating $\frac{49}{12}$, we find:

$$\frac{49}{12} \approx 4.0833$$

Since c must be a positive integer, the possible values for c are $1, 2, 3$, and 4. Now, we find the product of these values:

$$1 \times 2 \times 3 \times 4 = 24$$

Thus, the product of all positive integer values of c such that the quadratic equation has two real roots is 24.

SL

Python

```
from sympy import symbols, solve, Rational
def solution():
  x, c = symbols('x c')
  # For the quadratic equation to have
two real roots, the discriminant must be
non-negative
  # Discriminant of $3x^2 + 7x + c = 0$ is $7^2 -$
$4 * 3 * c$
  discriminant = 7**2 - 4*3*c
  # Solve for c when discriminant is greater
than 0
  c_values = solve(discriminant > 0, c)
  # c must be a positive integer
  positive_integer_c_values = [i for i in
range(1, int(c_values[0]) + 1)]
  # Calculate the product of all positive
integer values of c
  product = 1
  for value in positive_integer_c_values:
    product *= value
  return product solution()
```

Executed Result

TypeError: 'And' object is not subscriptable

Table 12: An example where NL outperforms SL. When writing python code to solve the problem, another key factor is that the language model has learned this symbolic language well. Otherwise, it may produce incorrect code, even if it understands the right concepts.

Question

How many vertical asymptotes does the graph of $y = \frac{x-3}{x^2+7x-30}$ have?

NL

Let's think step by step. To find the vertical asymptotes of the function $y = \frac{x-3}{x^2+7x-30}$, we need to determine where the denominator is equal to zero, as these are the points where the function is undefined and may have vertical asymptotes. First, we set the denominator equal to zero:

$$x^2 + 7x - 30 = 0.$$

Next, we factor the quadratic equation:

$$x^2 + 7x - 30 = (x + 10)(x - 3) = 0.$$

Setting each factor equal to zero gives us the solutions:

$$x + 10 = 0 \quad \Rightarrow \quad x = -10,$$
$$x - 3 = 0 \quad \Rightarrow \quad x = 3.$$

These are the potential vertical asymptotes. However, we must check if these points are also zeros of the numerator, which would indicate a hole instead of an asymptote. The numerator is $x - 3$. Setting it equal to zero gives:

$$x - 3 = 0 \quad \Rightarrow \quad x = 3.$$

Since $x = 3$ is a zero of both the numerator and the denominator, it is a removable discontinuity (a hole) rather than a vertical asymptote.
Therefore, the only vertical asymptote occurs at $x = -10$.
Thus, the graph of $y = \frac{x-3}{x^2+7x-30}$ has 1 vertical asymptote.

SL

Python

```
from sympy import symbols, solve
def solution():
  x = symbols('x')
  denominator = x**2 + 7*x - 30

  # Find the roots of the denominator
  roots = solve(denominator, x)

  # The number of vertical asymptotes is the number of distinct real roots
  num_vertical_asymptotes = len(roots)

  return num_vertical_asymptotes
solution()
```

Executed Result

2

Table 13: An example where NL outperforms SL. SL doesn't consider the case where the numerator equals zero, which highlights the weakness of SL reasoning when facing certain edge cases or special conditions.