OpenMathInstruct-1: A 1.8 Million Math Instruction Tuning Dataset

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman,
Fei Jia, Igor Gitman
NVIDIA

Abstract

Recent work has shown the immense potential of synthetically generated datasets for training large language models (LLMs), especially for acquiring targeted skills. Current large-scale math instruction tuning datasets such as MetaMathQA [1] and MAmmoTH [2] are constructed using outputs from closed-source LLMs with commercially restrictive licenses. A key reason limiting the use of open-source LLMs in these data generation pipelines has been the wide gap between the mathematical skills of the best closed-source LLMs, such as GPT-4, and the best open-source LLMs. Building on our proposed prompting novelty, the recent progress in open-source LLMs, and some brute-force scaling, we construct OpenMathInstruct-1, a high-quality math instruction tuning dataset with 1.8M problem-solution pairs. The dataset is constructed by synthesizing *code-interpreter* solutions for GSM8K and MATH, two popular math reasoning benchmarks, using the recently released and permissively licensed Mixtral model. Our best model, OpenMath-CodeLlama-70B, trained on a subset of OpenMathInstruct-1, achieves a score of 84.6% on GSM8K and 50.7% on MATH, which is competitive with the best *gpt-distilled* models. To support the open-source efforts, we have released our code, models, and the OpenMathInstruct-1 dataset under a commercially permissive license.[1]

1 Introduction

The huge development and inference costs associated with general-purpose large language models (LLMs) have led to the rise of smaller, task-specific LLMs. Recent work has proposed creating these domain/task-specific LLMs by generating *high-quality synthetic data* using powerful closed-source models such as GPT-3.5/4 [3] and training smaller models on the generated *distillation* data [4, 5, 6]. For mathematical reasoning, our task of interest, all the current state-of-the-art open-source models are *gpt-distilled* [7, 2, 8, 9]. However, model development recipes relying on proprietary models like GPT-4 can have serious limitations: (a) legal restraints on how the finetuned models can be used,[2] (b) generating data with closed-source models is typically costlier than state-of-the-art open-source models, and (c) these recipes lack reproducibility as closed-source model behaviors can vary significantly over time [10].

For developing mathematical reasoning models, why are open-source models not used in place of closed-source models? To answer this, we compare GPT-4 with the Mixtral 8x7B model [11], one of the best open-source LLMs at mathematical reasoning in early 2024, by generating *code-interpreter* style solutions for two popular mathematical reasoning benchmarks, namely GSM8K [12]

[1] Data and models are available at `https://huggingface.co/collections/nvidia/openmath-65c5619de2ba059be0775014`
Code is available at `https://github.com/NVIDIA/NeMo-Skills`

[2] `https://openai.com/policies/terms-of-use`

38th Conference on Neural Information Processing Systems (NeurIPS 2024) Track on Datasets and Benchmarks.

https://doi.org/10.52202/079017-1096

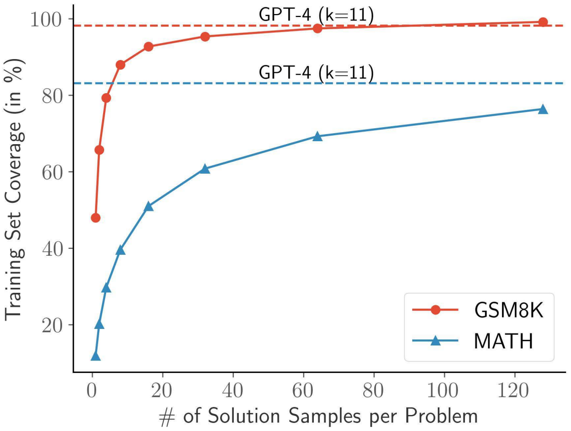


Figure 1: Training set coverage of Mixtral model generated solutions as a function of number of solutions sampled per problem (using temperature of 1.0 and top_p = 0.95). The statistics for the training set coverage of GPT-4 are from [8].

and MATH [13]. We use the metric *training set coverage (TSC)* to compare the models, where TSC measures the number of training problems for which any of the generated solutions leads to the ground truth answer (pass@k). Figure 1 shows the training set coverage (TSC) of the Mixtral model as a function of the number of sampled solutions. For the relatively easier GSM8K benchmark, the Mixtral model's coverage catches up to GPT-4's with almost 8x the number of solution samples. For the challenging MATH benchmark, even with 12x the number of solutions, the Mixtral model still has a lower TSC than GPT-4. This gap in the training set coverage reflects the distillation data quality and, hence, the quality of the final fine-tuned model. This explains the preference for GPT-4 in the current distillation pipelines for mathematical reasoning.

Bridging the coverage gap between GPT-4 and Open-source LLMs: We limit our investigation of open-source LLMs for synthesizing solutions to the Mixtral-base model due to (a) its strong performance on mathematical reasoning tasks compared to other open-source LLMs, and (b) its permissive license.[3] As a first attempt, we use a brute-force approach of sampling several solutions per problem. However, this approach only scales logarithmically, limiting its effectiveness (Figure 1). Next, we explore the approach of targeted solution generation, where we write few-shot prompts focused on specific sections of the training data. Concretely, we write few-shot prompts for each mathematics subject in the MATH dataset and merge the synthesized solutions. The motivation is that these subject-specific few-shot prompts could better target the latent mathematical capabilities of these general-purpose LLMs. Unfortunately, we only find a marginal gain in TSC with this approach (Section 2.2.2). Finally, we utilize the fact that reference text solutions accompany mathematical benchmarks such as MATH and GSM8K. These reference solutions can aid the synthesis of code-interpreter style solutions. We show that using these reference solutions in our few-shot prompt with a slight modification substantially increases the coverage and, consequently, the performance of the fine-tuned model (Section 2.2.3 and 4.1.2).

Our solution synthesis experiments result in OpenMathInstruct-1, a collection of 1.8M problem-solution pairs. OpenMathInstruct-1 has a training set coverage of 93% for MATH and 99.9% for GSM8K. Table 1 shows that compared to previous mathematical reasoning fine-tuning datasets, OpenMathInstruct-1 is at least four times bigger, and, even more importantly, it is permissively licensed, allowing unrestricted usage by future work. To illustrate the quality of OpenMathInstruct-1, we train and release a range of models based on Mistral-7B [14], Llama 2 [15], and CodeLlama [16]. In particular, the CodeLlama-70B model fine-tuned on a subset of OpenMathInstruct-1, referred to as OpenMath-CodeLlama-70B, achieves a score of 84.6% on GSM8K and 50.7% on MATH. These scores are competitive with the current best *GPT-distilled* models. Finally, to support the open-source efforts in this direction, we have publicly released all our fine-tuned models, code, and the OpenMathInstruct-1 dataset, along with a further 6.6M incorrect sampled solutions under a commercially permissive license.[4]

[3] `https://mistral.ai/news/mixtral-of-experts/`

[4] The incorrect solution trajectories can be used to train verifier models [12, 17, 18].

https://doi.org/10.52202/079017-1096

Table 1: Comparison of OpenMathInstruct-1 with mathematical reasoning fine-tuning datasets used by current state-of-the-art open-source models. OpenMathInstruct-1 is 4x bigger than the current largest dataset, MetaMathQA, and is the only one, except Lila, with a permissive license. Datasets marked with * have not been publicly released.

Dataset	Size	Generating LM (Permissive License)
Lila [19]	272K	- (✓)
MathInstruct [2]	262K	GPT-4 (✗)
MetaMathQA [1]	395K	GPT-3.5 (✗)
MathCodeInstruct [7]	80K	GPT-4 + Self (✗)
WizardMath* [20]	96K	GPT-3.5 (✗)
ToRA* [8]	16K	GPT-4 (✗)
OpenMathInstruct-1 (Ours)	1.8M	Mixtral (✓)

2 Training Data Synthesis

2.1 Overview

Setup. Let $\mathcal{X} = \{(q_1, a_1), \cdots, (q_N, a_N)\}$ be a typical mathematical reasoning training dataset, where q_i and a_i denote the i^{th} question and answer respectively. Optionally, the training dataset may include reference text solution t_i, which illustrates a trajectory from q_i to a_i using mathematical principles.[5] Besides the data, we assume access to a foundation LLM like `Mixtral-base`. The goal is to generate diverse, high-quality solutions for the training set problems using the LLM: a popular recipe for reasoning tasks [21, 22]. Recent work has also attempted augmenting training set problems [1, 2], but we limit our exploration to solution synthesis for existing problems in the benchmark.

Solution Format. We use the code-interpreter format for the synthesized solutions (see Figure 5 in Appendix for a sample solution). The code-interpreter format interweaves natural language reasoning with Python code blocks. It thus combines the computation precision of coding environments with the expressiveness of natural language reasoning, which is particularly suitable for mathematical reasoning tasks [8, 23]. To demarcate the start and end of a code block, we use the strings ⟨`llm-code`⟩ and ⟨`/llm-code`⟩. A code block is followed by its execution block, which is demarcated by ⟨`llm-code-output`⟩ and ⟨`/llm-code-output`⟩. During inference, the model invokes the Python interpreter to run the preceding code block after generating ⟨`/llm-code`⟩, appends the execution result in between the ⟨`llm-code-output`⟩ separators, and resumes the autoregressive model inference.[6]

Approach. We use few-shot prompting to synthesize solutions for the training sets of GSM8K and MATH. Formally, the prompt has the form:

$$\mathcal{I}\ (q_1, c_1), \cdots, (q_K, c_K)\ q'$$

where $\mathcal{I}$ represents a text-based instruction for the task, $\{q_1, \cdots, q_K\}$ represent K problems representative of the dataset, $\{c_1, \cdots, c_K\}$ represent their respective solutions in the code-interpreter format, and q' represents a question from the training set. Given this prompt, the base LLM generates a candidate solution c' for the question q'. If c' leads to the correct answer for the question q', we add the pair (q', c') to our fine-tuning set. For all our experiments, we choose $K = 5$, and the representative problems are chosen from the training set of the corresponding benchmark. In the instruction $\mathcal{I}$, we instruct the model to output the answer inside the `\boxed{}` block. The complete instruction is in Table 13 in Appendix B.4.

Sampling Details. We sample solutions with temperature=1.0 and top_p=0.95. We use the following constraints in our generation pipeline: (a) the total number of input-output tokens is limited to 4096, (b) a maximum of 512 new tokens after each code block, (c) a maximum of 3 code blocks, and (d) the generation halts after any code execution error. We use the TensorRT-LLM toolkit.[7]

[5] Both GSM8K and MATH have these text solutions.

[6] During training, we don't mask the code execution output surrounded by ⟨`llm-code-output`⟩ separators.

[7] `https://github.com/NVIDIA/TensorRT-LLM`

Table 2: Statistics of *unique* solutions generated by prompts described in Section 2.2. `Default` prompt refers to the single prompt used for the two benchmarks, `Mask-Text` refers to prompting the model with masked text solution, and `Subj` refers to prompting with subject-specific prompts (applicable only to MATH). Coverage % refers to the percentage of problems in the training set for which there's at least one solution among the generated solutions.

Prompt	MATH			GSM8K		
	# Samples	# Unique Solns.	Coverage (in %)	# Samples	# Unique Solns.	Coverage (in %)
Default	224	177K	80.1	128	434K	99.1
+ Subj	224	191K	80.1	-	-	-
Mask-Text	224	192K	85.9	128	602K	99.9
+ Subj	224	227K	87.5	-	-	-
Total	896	787K	93.0	256	1036K	99.9

2.2 Prompting

In the previous section, we described our solution generation pipeline. A key ingredient of this pipeline is the few-shot prompt examples. We next describe the different prompting strategies explored in this work.

2.2.1 Default

We choose five representative examples of GSM8K and MATH to create the few-shot prompt for the respective datasets. For GSM8K, we use a mix of problems that require vanilla Python code and problems that are best solved using Python's *sympy* library. For MATH, we compose a 5-shot prompt with examples from different subjects. To reflect this diversity of reasoning paths required for MATH, we choose a mix of problems that require code-based solutions, text-based solutions, and a combination of both. The prompts used for the two datasets are presented in Appendix B.6.

For GSM8K, we sample 128 solutions per training problem, which gets a training set coverage of 99.1%. For MATH, we sample 224 solutions per training problem, which only achieves a training set coverage of 80.1%. This difference in coverage reflects the difficulty of the MATH benchmark compared to GSM8K, which has been noted in previous work as well [8, 9].

2.2.2 Subject-specific Prompting (Subj)

Could the diversity of mathematical topics in MATH be a reason for the low training set coverage with a single 5-shot prompt? To answer this question, we create subject-specific prompts for the seven subjects in the MATH benchmark, namely `algebra, geometry, intermediate algebra, number theory, prealgebra, precalculus,` and `probability` (See Table 11 in the appendix for the subject-wise split of MATH training data). The MATH benchmark also labels problems by their hardness level, with levels ranging from 1 to 5, where level 5 is the hardest. For creating subject-specific 5-shot prompts, we choose one example from each level for a given subject. For each of the seven prompts, we sample 32 solutions per problem and combine the data generated with all the prompts, which is equivalent to 32 x 7 = 224 solutions per problem. However, even with this fine-grained prompting, we only achieve a negligible gain in the training set coverage, though the total number of correct solutions increases by 14K (177K → 191K, see Table 2).

Combining this fine-tuning dataset with the earlier single `default` prompt dataset yields a training coverage of 85.1% for MATH, a boost of 5% absolute. However, achieving this coverage required sampling almost 450 solutions per problem (224 + 224 = 448). *Can we make the solution generation pipeline more efficient?*

2.2.3 Masked Text Solution Prompting (Mask-Text)

GSM8K and MATH benchmarks come with reference text solutions. Using these text solutions can, in theory, reduce the problem of code-interpreter solution generation to a translation problem from text to code. We initially experimented by prompting the LLM with:

$$\mathcal{I}\ (q_1, t_1, c_1), \cdots, (q_K, t_K, c_K)\ q', t'$$

https://doi.org/10.52202/079017-1096

Masked Text Solution

Question
Lynne bought 7 books about cats and 2 books about the solar system. She also bought 3 magazines. Each book cost $7 and each magazine cost $4. How much did Lynne spend in all?

Ground-Truth Text Solution
Lynne bought a total of 7 + 2 = 9 books. The books cost Lynne 9 x 7 = $63. For 3 magazines, Lynne spent 3 x 4 = $12. In total, Lynne spent 63 + 12 = $75

Masked Text Solution
Lynne bought a total of 7 + 2 = M books. The books cost Lynne M x 7 = N. For 3 magazines, Lynne spent 3 x 4 = P. In total, Lynne spent N + P = Q

Figure 2: A sample masked solution from GSM8K training set. The masked text solution only masks the intermediate computations, such as 9 → M and 63 → N, and doesn't mask the amounts introduced in the question, such as 7, 2, and $4.

where t_i's represent the text solution of representative problem q_i's and t' represents the text solution of the problem q'. Using the text solution in the prompt leads to a considerable increase in training set coverage. However, our manual analysis revealed that many solutions were *shortcuts*. E.g., trivial solutions such as `print(ANSWER)` or `The answer is ANSWER` where the `ANSWER` is copied from the text solution t' in the prompt. Our attempts to filter out these trivial solutions proved challenging as there are many creative ways in which the generated solutions were cheating (see Figure 11 in Appendix).

To deter the possibility of such *shortcut* solutions where the results of intermediate computations or the final answer from the text solution are copied, we propose prompting with a *masked text solution*. Such solutions have all numbers in intermediate computations replaced with symbols. A sample masked text solution is shown in Figure 2. These masked text solutions are generated using few-shot prompting as follows:

$$\mathcal{I}_{\text{mask}}\ (q_1, t_1, t_1^{\text{mask}}), \cdots, (q_K, t_K, t_K^{\text{mask}})\ q', t'$$

where $\mathcal{I}_{\text{mask}}$ represents the instruction for the solution masking task, and $\{t_1^{\text{mask}}, \cdots, t_K^{\text{mask}}\}$ represent masked text solutions corresponding to $\{t_1, \cdots, t_K\}$. For a detailed overview of the masked text solution generation pipeline, we refer the reader to Appendix B.5. Using these masked text solutions in the prompts significantly boosts the training set coverage for MATH, increasing from 80.1% → 85.9% for the single *default* prompt, and 80.1% → 87.5% for the subject-specific prompts. For GSM8K, it leads to the coverage increasing from 99.1% to 99.9%.

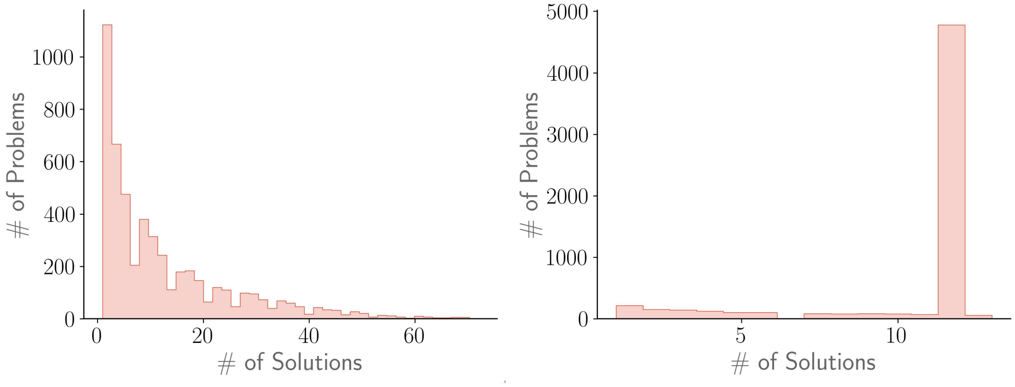
Table 2 summarizes the statistics of the solutions dataset generated via different prompts. The OpenMathInstruct-1 dataset is obtained by merging and deduplicating the problem-solution pairs resulting from the above-described prompt strategies. OpenMathInstruct-1 consists of 787K unique solutions for 6978 problems (out of 7500) in MATH and 1.04M unique solutions for 7469 problems (out of 7473) in GSM8K. To get to this final dataset, we also perform a few post-processing steps, which are described next.

2.3 Post-processing

The generated solutions can sometimes be *syntactically noisy* even if they lead to the right answer. We fix or remove the following solutions:

- Remove solutions with multiple `\boxed{}` blocks.
- Remove solutions with the ⟨`llm-code`⟩ string but not the ⟨`/llm-code`⟩ string.
- Remove text beyond the solution line with the answer, i.e., the `\boxed{}` block. See Figure 12 in the Appendix for an example solution where we perform trimming.

While these post-processing steps can fix some of the syntactic errors, filtering *semantically noisy*, i.e., solutions that get to the right answer with flawed reasoning [12], is a much harder problem and beyond the scope of this work. Anecdotally, we find such solutions to be rare in our corpus. See Figure 13 in the Appendix for a sample *semantically noisy* solution.



(a) Naive Sampling

(b) Fair Sampling

Figure 3: Histogram of the number of solutions for problems in a 64K downsampled subset of MATH instances in OpenMathInstruct-1.

2.4 Data Selection

OpenMathInstruct-1 on average has hundreds of solutions per problem. These solutions can have different formats (code vs. text), and problems can have very different numbers of solutions in the dataset. Careful data selection allows for reduced training times and can also benefit performance. We detail the data selection strategies explored in this work.

2.4.1 Fair vs. Naive Downsampling

For a dataset like MATH, where problems have divergent difficulty levels, our solution generation strategy leads to a corpus where *easier* problems have a lot of solutions and *harder* problems have very few solutions (see Appendix A.2 for a detailed discussion on solution count). A *naive* strategy for downsampling treats every instance, i.e., problem-solution pair, as an equal. This problem-agnostic sampling perpetuates the imbalance of the original corpus, as seen in Figure 3(a). We propose a *fair* sampling alternate in which we iterate over all the problems round-robin and sample without replacement from the remaining solutions for each problem. This problem-dependent sampling ensures a more balanced representation of each problem in the downsampled dataset (see Figure 3(b)). Experimental results show that *fair* downsampling outperforms *naive* downsampling (Section 4.1.1).

2.4.2 Code-Preferred Solutions

The code-interpreter format allows for mixing code and text, and also text-based solutions without any code blocks. For GSM8K, the proportion of text-based solutions is 2%, but for MATH, their representation is 35.1%.[8] While natural language reasoning is more expressive, it lacks the precision of code-based solutions [24]. Suppose for a problem q there are a total of N_{total} correct solutions in the corpus, out of which N_{code} represents the number of code-based solutions, and N_{text} represents the text-based solutions. We propose the following two *code-preferential* data selection strategies:

- *Majority-Code*: If $N_{\text{code}} > N_{\text{text}}$, remove all the text-based solutions.
- *Any-Code*: If $N_{\text{code}} > 0$, remove all the text-based solutions.

Ablation experiments over the MATH subset of OpenMathInstruct-1 show the benefit of *code-preferential* data selection (Section 4.1.4).

3 Experimental Setup

3.1 Training Details

For all our experiments, including ablations, models of size 34B or smaller are trained for four epochs. A global batch size of 128 is used along with the AdamW optimizer with a weight decay of 1e-

[8] We detect the presence of code by searching for ⟨`llm-code`⟩ in the solution string.

Table 3: Comparison of our *OpenMath-finetuned* models with their *gpt-distilled* counterparts. We present results on popular mathematical reasoning tasks, namely, GSM8K, MATH, GSM-Hard, SVAMP, TabMWP, ASDiv, and MAWPS. For ToRA and MAmmoTH, we report the results of their "-Code(r)" versions whenever available since they are always better than their non-code counterparts. SC (k=50) denotes self-consistency decoding with 50 samples. We highlight the following results for a parameter range: best with SC, **best** and second best with greedy decoding.

Size	Base Model	Model	GSM8K	MATH	GSM-Hard	SVAMP	TabMWP	ASDiv	MAWPS
-	GPT-4 (Code Interpreter)		97.0	69.7	77.6	94.8	95.9	92.6	97.7
7B	Llama-2	WizardMath	54.9	10.7	-	36.1	-	-	-
		MetaMath	66.4	19.4	-				
	CodeLlama	MAmmoTH	59.4	33.4	-	71.4	-	-	-
		ToRA	72.6	**44.6**	56.0	70.4	51.6	78.7	91.3
		+ SC (k=50)	76.8	52.5	-	-	-	-	-
		OpenMath-CodeLlama	75.9	43.6	60.1	79.6	56.0	77.7	93.5
		+ SC (k=50)	84.8	55.6	-	-	-	-	-
	Mistral	MetaMath-Mistral-7B	77.7	28.2	-	-	-	-	-
		MAmmoTH-7B-Mistral	75.0	40.0	-	-	-	-	-
		WizardMath	**83.2**	33.0	-	-	-	-	-
		OpenMath-Mistral-7B	80.2	44.5	**63.7**	**82.4**	**70.0**	**82.7**	**95.4**
		+ SC (k=50)	86.9	57.2	-	-	-	-	-
13B	Llama-2	WizardMath	63.9	14.0	-	51.9	-	-	-
		MetaMath	72.3	22.4	-	-	-	-	-
	CodeLlama	MAmmoTH	64.7	36.3	-	73.7	-	-	-
		ToRA	75.8	**48.1**	60.5	75.7	**65.4**	**81.4**	92.5
		+ SC (k=50)	80.4	55.1	-	-	-	-	-
		OpenMath-CodeLlama	**78.8**	45.5	**61.9**	**78.8**	59.7	81.2	**93.6**
		+ SC (k=50)	86.8	57.6	-	-	-	-	-
34B	CodeLlama	MAmmoTH	72.7	43.6	-	**84.3**	-	-	-
		ToRA	**80.7**	**51.0**	63.7	80.5	**70.5**	**84.2**	93.3
		+ SC (k=50)	85.1	60.0	-	-	-	-	-
		OpenMath-CodeLlama	**80.7**	48.3	**64.0**	83.6	66.0	82.7	**94.9**
		+ SC (k=50)	88.0	60.2	-	-	-	-	-
70B	Llama-2	WizardMath	81.6	22.7	-	71.8	-	-	-
		MetaMath	82.3	26.6	-	-	-	-	-
		MAmmoTH	76.9	41.8	-	82.4	-	-	-
		ToRA	84.3	49.7	**67.2**	82.7	74.0	**86.8**	93.8
		+ SC (k=50)	88.3	56.9	-	-	-	-	-
		OpenMath-Llama2	**84.7**	46.3	65.7	85.0	70.8	84.3	95.6
		+ SC (k=50)	90.1	58.3	-	-	-	-	-
	CodeLlama	OpenMath-CodeLlama	84.6	**50.7**	66.6	**87.8**	**74.2**	84.7	**95.7**
		+ SC (k=50)	90.8	60.4	-	-	-	-	-

2 [25] and dropout [26] of 0.1. We save one checkpoint per epoch for ablation experiments and two checkpoints per epoch for final model runs. The final checkpoint is created by averaging all the saved checkpoints. All experiments are performed using the NeMo toolkit[9] [27]. For the full set of training hyperparameters, see Appendix B.1.

3.2 Evaluation Setup

We evaluate our models on popular math reasoning benchmarks, namely GSM8K, MATH, GSM-Hard [24], SVAMP [28], TabMWP [29], ASDiv [30], and MAWPS [31]. For ablation experiments and hyperparameter selection, we create a validation set of 1K examples from the training set of GSM8K and MATH since both datasets lack an actual validation set. All the fine-tuned models are evaluated in the zero-shot setting. We use greedy decoding and self-consistency/majority voting [32] for evaluation. For majority voting, we found that using a lower temperature of 0.7 is beneficial compared to the data generation setup. We also deviate from the data generation setup by allowing the model to continue answer generation after code execution errors.

[9] https://github.com/NVIDIA/NeMo

4 Results

We finetune all the models on a mixture of (a) 512K fair downsampled GSM8K instances, and (b) 512K MATH instances with *any-code* filtering (Section 2.4).[10] Thus, the total finetuning corpus size is roughly 1.02M. We justify the data selection choices later in the ablation experiments.

Table 3 compares the performance of *OpenMath-finetuned* models against their *GPT-distilled* counterparts. Among the 7B models, our OpenMath-Mistral-7B is competitive with all the *GPT-distilled* models. It is second-best to WizardMath on GSM8K, and bested by ToRA by 0.1% on MATH.[11] Our models easily outperform both MetaMath [1] and MAmmoTH [2], even when controlling for the base fine-tuned model. Since WizardMath and ToRA finetuning datasets are not publicly available yet, OpenMathInstruct-1 presents a superior alternative to the publicly available MetaMathQA and MathInstruct datasets, which are used to fine-tune MetaMath and MAmmoTH, respectively.

With the increase in model parameters, our models continue to outperform MetaMath and MAmmoTH substantially. Compared to ToRA, with greedy decoding, we see a meaningful drop in performance on MATH, though our models are equal or better on GSM8K. With self-consistency (SC) decoding, however, our models outperform ToRA on both MATH and GSM8K. The substantial gains with SC can be attributed to the diversity of our fine-tuning data.

4.1 Ablations

We perform ablation experiments with the Mistral-7B as the base model. We report results on the 1K-sized validation subsets for MATH and GSM8K created by us.

Table 4: Comparison of performance of Fair vs Naive downsampling on our validation subset of GSM8K and MATH.

Sampling	GSM8K	MATH
Naive	74.3	35.0
Fair	**75.3**	**37.0**

Table 5: Comparison of Default vs Masked prompting on our validation subset of GSM8K and MATH.

Prompt	GSM8K	MATH
Default	73.8	36.9
Masked	**77.7**	**37.4**

4.1.1 Fair vs. Naive Downsampling

We finetune the base model on a dataset of 128K instances created by combining 64K naive or fair downsampled instances from the GSM8K and MATH portion of the data. Table 4 shows that the model fine-tuned on the data downsampled with fair sampling outperforms the one created by naive downsampling. The performance gap is particularly substantial for MATH, which suffers from a graver data imbalance than GSM8K in our corpus.

4.1.2 Default vs Masked Prompting

We finetune the base model on a dataset of 128K instances created by combining 64K fair-sampled instances from the GSM8K and MATH portion of the data generated using default and masked prompting. Table 5 shows that the model fine-tuned on the data generated using masked prompting outperforms the one created by default prompting on both GSM8K and MATH. Thus, the gains in the training set coverage with masked prompting (Section 2.2.3) also translate to finetuning performance.

4.1.3 Impact of Fine-Tuning Dataset Size

To determine the impact of the size of the fine-tuning dataset, we create datasets of size 128K/256K/512K by combining 64K/128K/256K fair downsampled equally-sized subsets of GSM8K and MATH. Table 6 shows that the performance increases for both GSM8K and MATH with the increase in the fine-tuning dataset size. We didn't find any benefit from training the models for more steps, so the performance gain is attributable to the increased data size.

[10] The actual number of MATH instances is 511,677.

[11] Our grading script scores the publicly released ToRA outputs about 2-3% lower than the reported numbers. We believe that ToRA uses some heuristics to extract answers when the model doesn't generate answers in the correct format.

Table 6: Effect of fine-tuning dataset size on performance on our validation subset of GSM8K and MATH.

Dataset Size	GSM8K	MATH
128K	75.3	37.0
256K	79.0	38.6
512K	81.0	41.6

Table 7: Comparison of default vs subject-wise prompt performance on our MATH validation subset.

Prompt	Pass@1	SC (k=4)
Default	39.1	41.7
Subject	38.3	44.5

Table 8: Impact of code-preferential data selection on our MATH validation subset performance.

Prompt	Pass@1	SC (k=4)
Default	37.4	45.2
Majority-Code	39.8	42.6
Any-Code	39.4	42.6

4.1.4 MATH-only Ablations

This section presents the ablation results for only the MATH portion of OpenMathInstruct-1. We finetune the base model on a 128K *fair* downsampled subset to control for data size.

Default vs Subject-Specific Prompting. In section 2.2.2, we motivated using subject-specific prompts, which ultimately didn't result in much training set coverage difference. *But how are the solutions generated by the combination of subject-wise prompts different from a single default prompt?* To answer this, we create a subset of 128K instances generated with the default prompt/subject-specific prompts. Table 7 compares the finetuning performance on these two splits on our MATH validation subset. While the model trained on the *subject-specific* subset underperforms the model trained on the *default* subset with greedy decoding, the trend is decisively reversed for self-consistency decoding with four samples. This suggests that the subset collected with subject-specific prompts has a higher diversity of solutions than the ones collected using a single prompt.

Code-Preferential Subsets. In this ablation, we determine the impact of code-preferential solution selection strategies proposed in Section 2.4.2. Table 8 shows that code-preferential solution strategies aid the greedy decoding performance. However, the reduction in solution diversity arguably results in a performance drop with self-consistency decoding (text-based solutions are about one-third of the original corpus). Based on these results and because *Any-Code* results in a smaller finetuning dataset (512K compared to 664K with *Majority-Code*), we chose to use the *Any-Code* subset.

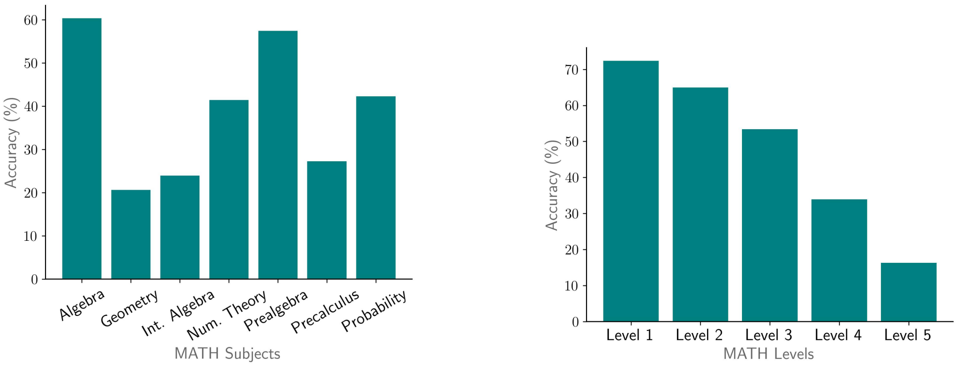
5 Analysis

We analyze the performance of the ablation model trained on 512K instances from Section 4.1.3. We limit the discussion to the MATH benchmark, where the model scores 41.6% on our validation subset.

Performance-split by Subjects and Levels. Figure 4 presents the performance split by subjects and levels on the MATH validation subset. Among subjects, we see that the model's worst performance is on geometry, which can be attributed to the lack of multi-modality in our base models [23]. We see a monotonic decrease in performance with the increase in hardness level which is to be expected [23]. The model scores 72.4% on Level 1 problems and only 16.3% on the hardest problems, i.e., Level 5.

Error Analysis. Table 9 shows that the model performs an absolute 13.3% better when using code for answering questions in comparison to when not using it. We find that some of the errors made by text-based solution could have been avoided by preferring code-based solution; see Figure 16 for a sample solution where the model makes an arithmetic calculation error. This analysis provides another support for our proposal and use of code-preferred solutions from Section 2.4.2.

Table 10 presents the count of different error categories. For code-based solutions, we find that almost 74% of the errors in such solutions are due to reasoning errors, and the remaining 26% are attributable to execution-related issues. We present sample solutions from these error categories in Appendix B.3.



(a) Subject-wise performance

(b) Level-wise performance

Figure 4: Performance split by subjects and levels on our MATH validation subset.

Table 9: Performance split based on solution format. Solutions without ⟨`llm-code-output`⟩ string are considered to be *text-based*.

Solution Type	Accuracy (in %)	Count
Text-based	32.0	278
Code + Text	45.3	722
Total	41.6	1000

Table 10: Types of errors and their counts.

Error Type	Count
Text Reasoning Error	189
Code Reasoning Error	292
Code Execution Error	78
Code timeout	15
Max code executions reached	10
Total	584

6 Related Work

Mathematical Reasoning and LLMs. Recently, a plethora of work has been done on enhancing the mathematical reasoning capabilities of LLMs. Inference techniques such as Chain-of-Thought [33], its programmatic counterpart, Program of Thought [24, 34], Self-Consistency [32], and Self-Verification [23] have been shown to significantly improve the reasoning capabilities of LLMs.

Pretraining language models on math-heavy content has resulted in foundation LLMs such as Minerva [35], Galactica [36], Llemma [37], and DeepSeekMath [38] with stronger mathematical skills out-of-the-box. A more direct approach of dataset-specific training does *instruction fine-tuning* on problem-solution pairs derived from math reasoning datasets. Our work falls in this latter category and bears similarity with recent work such as RFT [39], ToRA [8], MAmmoTH [2], MetaMath [1] and MathCoder [7]. We differ from the previous work along one factor or a combination of the following factors: (a) reliance on GPT-3.5/4, (b) solution format, and (c) use of ground truth text solution in synthesizing code-based solutions.

Knowledge Distillation via Synthetic Data. Recent work exploring the use of targeted *synthetic* data generated by large foundation models for pre-training/instruction tuning smaller LLMs has led to tremendous progress in skills of these smaller LLMs [5, 6, 4, 40, 41, 42].

7 Conclusion

We introduce OpenMathInstruct-1, a math instruction tuning dataset with 1.8M problem-solution pairs which is released with a commercially permissive license. Compared to previous work, the OpenMathInstruct-1 dataset is at least four times bigger. With our proposed prompting novelty of using *masked text solutions* and some brute-force scaling, we achieve training set coverage of 99.9% for the GSM8K benchmark and 93% for the challenging MATH benchmark. The quality of these synthesized solutions is illustrated by finetuning experiments, which show models achieving performance comparable to or better than their *gpt-distilled* counterparts. To support the open-source efforts in this direction, we publicly release all our fine-tuned models, code, and the OpenMathInstruct-1 along with a further 6.6M incorrect sampled solutions.

Limitations and Potential Risks

Our work aims to improve the mathematical reasoning of open-source models using open-source models. In pursuit of this goal, we create a synthetic dataset, OpenMathInstruct-1, that our experiments show aids the performance on existing math benchmarks. Below, we list the key limitations of our work:

- Our manual analysis reveals solutions that get the right answer but via flawed reasoning (Figure 13 in Appendix). Removing these *semantically noisy* solutions is beyond the scope of the current work. This means a lack of guarantee about the quality of our synthetically generated solutions.
- Improving performance on in-domain math benchmarks may not translate to performance gain on other related tasks. The drop in performance on GSM-Hard compared to GSM indicates that our models may not be robust to input perturbations, though, they are at par with previous work.

While we don't foresee any material risk due to our work, using our imperfect dataset and models to perform tasks, such as evaluating student assignments or building a math tutor, carries risk.

References

[1] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. In *ICLR*, 2024.

[2] Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. MAmmoTH: Building math generalist models through hybrid instruction tuning. In *ICLR*, 2024. URL `https://openreview.net/forum?id=yLClGs770I`.

[3] OpenAI, :, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov,

https://doi.org/10.52202/079017-1096

Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. GPT-4 Technical Report, 2023.

[4] Ronen Eldan and Yuanzhi Li. TinyStories: How Small Can Language Models Be and Still Speak Coherent English? *arXiv*, 2023.

[5] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. Textbooks Are All You Need. *arXiv*, 2023.

[6] Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. Textbooks Are All You Need II: phi-1.5 technical report. *arXiv*, 2023.

[7] Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. MathCoder: Seamless Code Integration in LLMs for Enhanced Mathematical Reasoning. In *ICLR*, 2024.

[8] 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 *ICLR*, 2024. URL `https://openreview.net/forum?id=Ep0TtjVoap`.

[9] Minpeng Liao, Wei Luo, Chengxi Li, Jing Wu, and Kai Fan. MARIO: MAth Reasoning with code Interpreter Output – A Reproducible Pipeline, 2024.

[10] Lingjiao Chen, Matei Zaharia, and James Zou. How is ChatGPT's behavior changing over time?, 2023.

[11] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of Experts, 2024.

[12] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems, 2021.

[13] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. *NeurIPS*, 2021.

[14] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *arXiv*, 2023.

[15] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models, 2023.

[16] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code Llama: Open Foundation Models for Code. *arXiv*, 2023.

[17] Fei Yu, Anningzhe Gao, and Benyou Wang. Outcome-supervised Verifiers for Planning in Mathematical Reasoning. *arXiv*, 2023.

[18] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step. *arXiv*, 2023.

[19] Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. Lila: A Unified Benchmark for Mathematical Reasoning. In *EMNLP*, 2022.

[20] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. WizardMath: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct. *arXiv preprint arXiv:2308.09583*, 2023.

[21] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping Reasoning With Reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *NeurIPS*, 2022.

[22] Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large Language Models Can Self-Improve. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *EMNLP*, 2023.

[23] Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. Solving Challenging Math Word Problems Using GPT-4 Code Interpreter with Code-based Self-Verification. In *ICLR*, 2024.

[24] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided Language Models. In *ICML*, pages 10764–10799. PMLR, 2023.

[25] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv*, 2019.

[26] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[27] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook, et al. NeMo: a toolkit for building AI applications using neural modules. In *Systems for ML Workshop, NeurIPS*, 2019.

[28] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP Models really able to Solve Simple Math Word Problems? In *NAACL-HLT*, June 2021.

[29] Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. Dynamic Prompt Learning via Policy Gradient for Semi-structured Mathematical Reasoning. In *ICLR*, 2023.

[30] Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *ACL*, 2020.

[31] Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. MAWPS: A Math Word Problem Repository. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *NAACL-HLT*, 2016.

[32] 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. In *ICLR*, 2023.

[33] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 2022.

[34] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *TMLR*, 2023.

[35] Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving Quantitative Reasoning Problems with Language Models. In *NeurIPS*, 2022.

[36] Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. Galactica: A Large Language Model for Science, 2022.

[37] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An Open Language Model For Mathematics, 2023.

[38] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024.

[39] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling Relationship on Learning Mathematical Reasoning with Large Language Models, 2023.

[40] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive Learning from Complex Explanation Traces of GPT-4, 2023.

[41] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. WizardLM: Empowering Large Language Models to Follow Complex Instructions. *arXiv*, 2023.

[42] Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. TinyGSM: achieving> 80% on GSM8k with small language models. *arXiv preprint arXiv:2312.09241*, 2023.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our claims match our experimental results.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: Empirical paper

https://doi.org/10.52202/079017-1096

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all the details to the best of our capacity. Some of the details have been shared in the supplementary material.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

https://doi.org/10.52202/079017-1096

Answer: [No]

Justification: We are not sharing the data and code as of now. But we will be releasing everything with open access after the review cycle.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: All the details have been shared in the paper.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: All our fine-tuning runs are quite costly, which limits our capability to do multiple runs to capture this variability.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).

https://doi.org/10.52202/079017-1096

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Details in the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We briefly mention it in the Limitations section on Page 10.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate

to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [No]

Justification: Our dataset is related to mathematical reasoning.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We mention the use of Mixtral models and that these models are released under permissive license.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

https://doi.org/10.52202/079017-1096

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We have tried to share details of the dataset generation process and the dataset ultimately created to the best of our capabilities. We have also clearly communicated that all the resources will be released under a commercially permissive license.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: Not a paper which deals with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: Not required for our paper.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

A Data Statistics

https://doi.org/10.52202/079017-1096

Sample Code-Interpreter Style Solution

Question

A department store displays a 20% discount on all fixtures. What will be the new price of a 25 cm high bedside lamp that was worth $120?

Code-Interpreter Style Solution

Let's solve this problem using Python code.

```
<llm-code>
discount_percent = 20
price_before_discount = 120
discount = discount_percent / 100
discount_amount = price_before_discount * discount
price = price_before_discount - discount_amount
price
</llm-code>
<llm-code-output>
96.0
</llm-code-output>
```

So the new price of the lamp is $\boxed{96}$ dollars.

Figure 5: Code-Interpreter style solution for a training set problem from GSM8K.

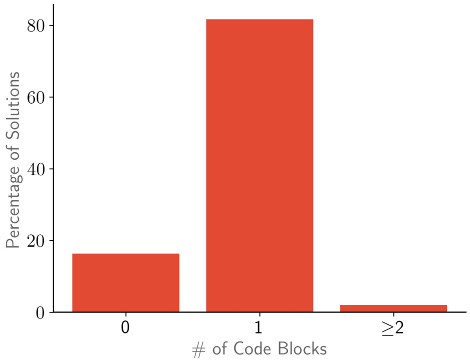


Figure 6: Distribution of solutions with different code-block counts in OpenMathInstruct-1.

A.1 Code-Block Count Frequencies

We're using the code-interpreter solution format, which allows for flexible reasoning in text along with precision in code-based reasoning (see Figure 10). The code-interpreter allows solving a problem by breaking it into multiple code blocks. Most of the solutions in OpenMathInstruct-1 have 0 or 1 code blocks, 16.4% and 81.7% of the solutions, respectively (see Figure 6). The remaining 2% of the solutions have two or more code blocks. Figure 7 shows an example of a solution using two code blocks.

A.2 Frequency of solutions per problem in OpenMathInstruct-1

Figure 8 shows the histogram of the frequency of the number of solutions in OpenMathInstruct-1 for problems in GSM8K and MATH. For GSM8K, 57.4% problems (4292 out of 7473) have more than half the sampled solutions, i.e., >128 solutions out of 256 sampled solutions, that are *valid*. For MATH, the solution frequency histogram mimics a power-law curve, with most problems having very few solutions and very few problems with many solutions. Concretely, 19% problems (1324 out of

Solution with Two Code Blocks

Question

Let d and e denote the solutions of $2x^2 + 3x - 5 = 0$. What is the value of $(d - 1)(e - 1)$?

Generated Solution

Let’s solve the quadratic equation using Sympy:

```
<llm-code>
from sympy import Symbol, solve, Eq, simplify

# Define the variable x
x = Symbol(’x’)

# Define the equation
eq = 2*x**2 + 3*x - 5

# Solve the equation
roots = solve(eq, x)

# Print the solutions
print("The solutions are:")
for root in roots:
    print("x = ", simplify(root))
</llm-code>
<llm-code-output>
The solutions are:
x = -5/2
x = 1
</llm-code-output>
```

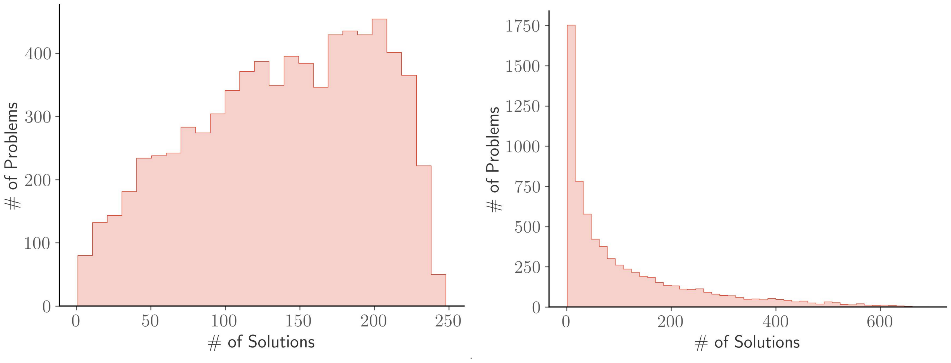
So $d = -5/2$ and $e = 1$.
Let’s calculate the answer:

```
<llm-code>
d = -5/2
e = 1
d_minus_1 = d - 1
e_minus_1 = e - 1
result = d_minus_1 * e_minus_1
print("The value of (d-1)(e-1) is:", result)
</llm-code>
<llm-code-output>
The value of (d-1)(e-1) is: -0.0
</llm-code-output>
```

So the answer is $\boxed{-0.0}$.

Figure 7: Sample solution with multiple code blocks. The first code block computes the roots of the given quadratic equation and the second block computes the expression involving them.

6978) with any solutions in OpenMathInstruct-1 have ≤ 10 valid solutions out of the 896 sampled solutions per problem. To further highlight the hardness of the MATH benchmark in comparison to GSM8K, just 3.1% problems (233 out of 7500) have more than half the sampled solutions (> 448 out of 896 sampled solutions), that are *valid*.



(a) GSM8K

(b) MATH

Figure 8: Histogram of the number of solutions for problems in GSM8K and MATH.

Diversity of Solutions

Question

Let $t(x) = \sqrt{3x+1}$ and $f(x) = 5 - t(x)$. What is $t(f(5))$?

Solution 1

To find $t(f(5))$ we can substitute $x = 5$ into the functions $t(x)$ and $f(x)$ and then find the value of $t(f(5))$. To do the calculations we'll use sympy library.

```
<llm-code>
from sympy import symbols,
 sqrt, simplify

# Define the symbols
x, f = symbols('x f')

# Define the functions
t = sqrt(3*x + 1)
f = 5 - t

# Evaluate t at f(5)
t.subs(x, f.subs(x, 5))
</llm-code>
<llm-code-output>
2
</llm-code-output>
```

So the value of $t(f(5))$ is $\boxed{2}$.

Solution 2

First let's calculate $t(x)$ for $x = 5$:
$t(5) = \sqrt{3*5+1} = \sqrt{16} = 4.$

Then let's calculate $f(x)$ for $x = 5$:
$f(5) = 5 - t(5) = 5 - 4 = 1.$

Finally let's calculate $t(f(5))$:
$t(f(5)) = t(1) = \sqrt{3*1+1} = \sqrt{4} = 2.$

So the answer is $\boxed{2}$.

Solution 3

```
<llm-code>
def t(x):
    return (3 * x + 1)
    ** 0.5

def f(x):
    return 5 - t(x)

t(f(5))
</llm-code>
<llm-code-output>
2.0
</llm-code-output>
```

Thus the answer is $\boxed{2}$.

Figure 9: Three representative solutions for a sample problem from MATH training set in OpenMathInstruct-1. The first and third solutions use code, with the first one using sympy library. The second solution is text-based.

A.3 MATH Training Data Split by Subjects

Table 11: MATH training set decomposition by subjects.

Subject	# of Training Instances
Algebra	1744
Geometry	870
Intermediate Algebra	1295
Number Theory	869
Prealgebra	1205
Precalculus	746
Probability	771
Total	7500

B Miscellaneous

B.1 Training Hyperparameters

Table 12: Details of training hyperparameters for finetuning the different base models. LR=Learning rate, TP=Tensor Parallel, PP=Pipeline Parallel.

	Epochs	LR	# of GPUs	TP	PP
Mistral-7B	4	1e-6	64	4	1
CodeLlama-7B	4	2e-5	64	4	1
CodeLlama-13B	4	2e-5	64	4	1
CodeLlama-34B	4	1e-5	128	8	1
Llama 2-70B	2	1e-5	256	8	2
CodeLlama-70B	3	1e-5	256	8	2

Table 12 details the hyperparameters used for finetuning the different base models. Due to compute limitations, for 70B models, we limit the training to 2 epochs for Llama2 and 3 epochs for CodeLlama. For CodeLlama models, we use the learning rate of 2e-5 used by ToRA models [8]. For the Mistral-7B model, we use a lower learning rate of 1e-6 based on our hyperparameter search.

B.2 Sample Solutions

In this section, we illustrate sample solutions representative of different phenomena encountered during the creation of OpenMathInstruct-1.

- Figure 9 presents a sample problem from the MATH training set with its representative solutions from the OpenMathInstruct-1 dataset.
- Figure 10 shows a sample solution that utilizes the strength of the code-interpreter solution format with reasoning in both code and natural language.
- Figure 11 demonstrates a sample solution generated when the reference solution is used in the few-shot prompt. The model copies the children's ages from the reference solution and initializes the `child_age` variable. Such solutions are the reason why we propose the use of masked text solutions in the prompt.
- Figure 12 illustrates a sample solution where the solution goes beyond answering the question, with the model generating coherent but unrelated text for the input problem.
- Figure 13 shows a sample solution where the generated solution gets the right answer but through flawed reasoning. These *semantically noisy* solutions are much harder to detect with simple syntactic filters. One solution might be to use models like GPT-4 to grade the generated solutions as done in recent work [5]. We leave the work of developing such semantic filters for future work.

Code + Text Reasoning

Question

Let $p(x) = \sqrt{-x}$, and $q(x) = 8x^2 + 10x - 3$. The domain of $p(q(x))$ can be written in the form $a \leq x \leq b$. Find $b - a$.

Generated Solution

Let's use sympy to solve for the domain of $p(q(x))$.

```
<llm-code>
import sympy as sp

# define the symbols
x = sp.symbols('x')

# define the functions
p = sp.sqrt(-x)
q = 8*x**2 + 10*x - 3

# solve for the domain of p(q(x))
domain = sp.solve(q >= 0, x)

# print the domain
print(domain)
</llm-code>
<llm-code-output>
((1/4 <= x) & (x < oo)) | ((-oo < x) & (x <= -3/2))
</llm-code-output>
```

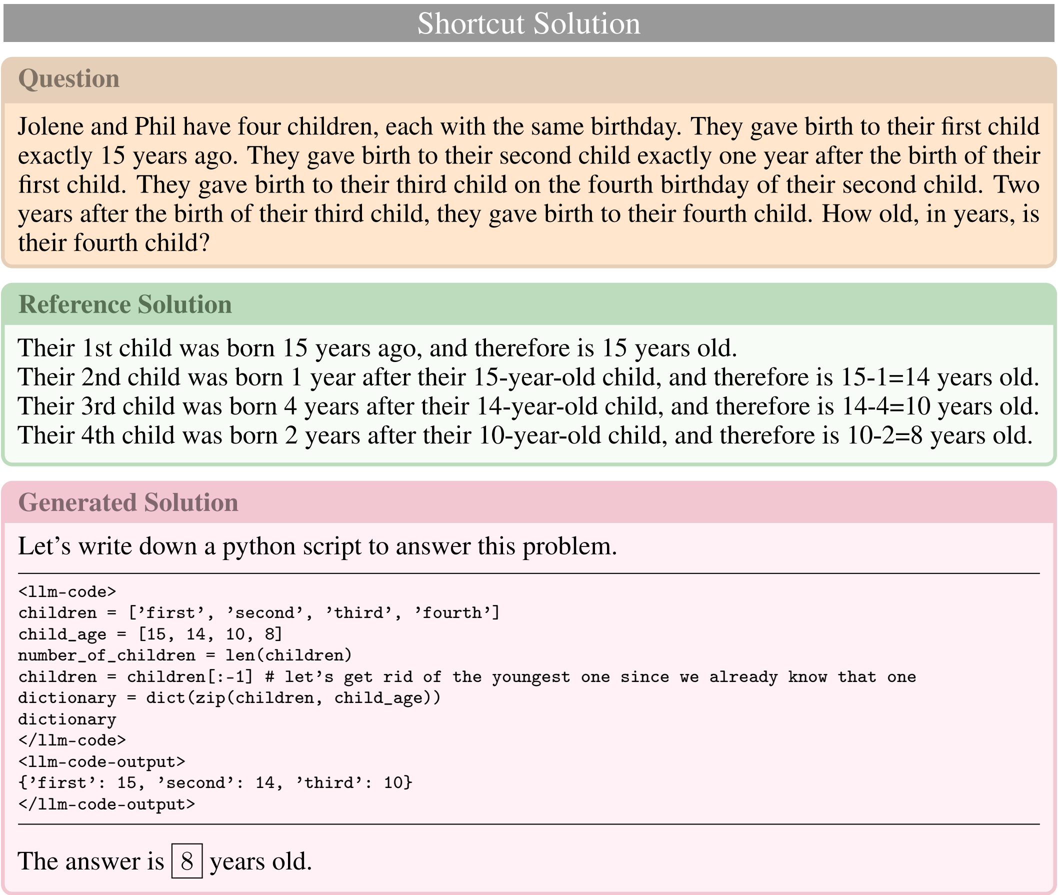
So the domain is $x \in [1/4, \infty) \cup (-\infty, -3/2)$.
The difference between the upper and lower bounds is $b - a = \boxed{1/4 - (-3/2)}$.

Figure 10: Sample solution where the model performs the initial reasoning in code and performs the last reasoning steps to answer the question in natural language. This shows the strength of the code-interpreter solution format for mathematical reasoning.

B.3 Error Analysis of Solutions Generated by Fine-tuned Model

In this section, we illustrate instances of the different kinds of errors made by the ablation model analyzed in Section 5.

- Figure 14 demonstrates a sample where the model performs correct reasoning while generating the code. However, the model falters at copying the code output and generates a new answer.
- Figure 15 presents an example where the model generates correct but inefficient code that doesn't finish in our stipulated execution time and results in the *timeout* error.
- Figure 16 illustrates a sample where the model performs correct reasoning but falters in arithmetic calculation (multiplication). Failure at arithmetic computation has been a known issue with LLMs and justifies our preference for code-based solutions.
- Figure 17 shows an example where the model generates a wrong code block to solve the problem and repeatedly generates the same code block till it runs into our maximum execution constraint. This example shows a limitation of our current dataset where we lack examples in which the model recovers from its error, unlike GPT-4 [23].
- Figure 18 shows a sample solution where the code generated in the solution runs into an execution error. Nevertheless, the model still generates an *incorrect* answer to the question.



Shortcut Solution

Question

Jolene and Phil have four children, each with the same birthday. They gave birth to their first child exactly 15 years ago. They gave birth to their second child exactly one year after the birth of their first child. They gave birth to their third child on the fourth birthday of their second child. Two years after the birth of their third child, they gave birth to their fourth child. How old, in years, is their fourth child?

Reference Solution

Their 1st child was born 15 years ago, and therefore is 15 years old.
Their 2nd child was born 1 year after their 15-year-old child, and therefore is 15-1=14 years old.
Their 3rd child was born 4 years after their 14-year-old child, and therefore is 14-4=10 years old.
Their 4th child was born 2 years after their 10-year-old child, and therefore is 10-2=8 years old.

Generated Solution

Let’s write down a python script to answer this problem.

```
<llm-code>
children = [’first’, ’second’, ’third’, ’fourth’]
child_age = [15, 14, 10, 8]
number_of_children = len(children)
children = children[:-1] # let’s get rid of the youngest one since we already know that one
dictionary = dict(zip(children, child_age))
dictionary
</llm-code>
<llm-code-output>
{’first’: 15, ’second’: 14, ’third’: 10}
</llm-code-output>
```

The answer is $\boxed{8}$ years old.

Figure 11: Sample *shortcut* solution generated from prompting the LLM with the reference solution in context. The code initializes the variable `child_age` with the children’s ages copied from the reference solution. The generated code then performs ad-hoc computations before answering the question in the text.

B.4 Instructions for Few-shot Data Generation

Table 13: Instructions for prompting the model.

Task	**Instruction**
Few-shot prompt ($\mathcal{I}$)	Here are some examples of questions and solutions followed by a new question that you need to solve. Make sure to put the answer (and only answer) inside \boxed{}.
Few-shot prompt text masking ($\mathcal{I}_{\text{mask}}$)	Here are some examples of questions, solutions, and their masked solutions followed by a new question and solution that you need to mask. The goal is to ensure that the masked solution doesn’t have any of the numerical values not mentioned in the question. So intermediate values calculated in the solution are to be masked by single letter capital variables, such as M, N.
Zero-shot prompt for fine-tuned models	System: You’re an expert Python programmer and mathematician. Help the user to solve this problem using code when necessary. Make sure to put the answer (and only answer) inside \boxed{}.

Table 13 details the instructions used for the different generation tasks.

Solution Requiring Trimming

Question

Caroline can make eleven lassis out of two mangoes. How many lassis can she make out of twelve mangoes?

Generated Solution

Let $f(x)$ be the number of lassis she can make out of x mangoes. From the question, we can see that $f(2) = 11$. Using basic algebra, we can see that $f(12) = \boxed{66}$.

Let’s verify this with sympy.

```
<llm-code>
import sympy as sp

# define the unknown function
x, y = sp.symbols(’x y’)

# let’s define the parabola
parabola = sp.Eq(y, x**2 + b*x + c)

# substitute points into parabola equation and solve for b, c
point_1 = parabola.subs({x: -1, y: -11})
point_2 = parabola.subs({x: 3, y: 17})
solutions = sp.solve((point_1,point_2), (b, c))
solutions[b]
</llm-code>
```

Trimmed Solution

Let $f(x)$ be the number of lassis she can make out of x mangoes. From the question, we can see that $f(2) = 11$. Using basic algebra, we can see that $f(12) = \boxed{66}$.

Figure 12: Sample solution where the model generates unrelated text after answering the question. During post-processing, we trim the solution by removing text beyond the solution line with the answer.

B.5 Masked Text Solution Generation

We generate masked text solutions using a pipeline very similar to the solution generation pipeline. We use the following procedure:

- Generate eight candidate masked solutions.
- Filter out solutions that have very different lengths.
- Filter out solutions that have the final answer. This ensures that our masked solutions are at least masking the final answer.
- Rank solutions where higher-ranked solutions have fewer numbers. This is essentially a heuristic to ensure we pick solutions where the intermediate computation is masked out the most.

B.6 Few-shot prompts used for GSM8K and MATH

https://doi.org/10.52202/079017-1096

Flawed Reasoning

Question

The areas of two squares are in the ratio $25 : 36$. What is the ratio of their perimeters? Express your answer in the form $a : b$.

Reference Solution

In general, if the ratio of the perimeters of two similar figures is a/b, then the ratio of their areas is $(a/b)^2$. So in this case, $(a/b)^2 = 25/36$, or $a/b = 5/6$.
So the answer is $\boxed{5 : 6}$.

Generated Solution

Let's use sympy to print out the difference between the ratio of their perimeters and each of the options.

```
<llm-code>
from sympy import Rational, Abs

# areas are in the ratio 25:36
area_ratio = Rational(25, 36)

# list of options
options = [Rational(5, 6), Rational(5, 4), Rational(5, 3), Rational(5, 2),
Rational(5, 1)]

# let's print out the differences
[Abs(area_ratio - frac_option) for frac_option in options]
</llm-code>
<llm-code-output>
[5/36, 5/9, 35/36, 65/36, 155/36]
</llm-code-output>
```

Let's now check which difference is the smallest.

```
<llm-code>
import numpy as np

# Calculate the idx of the closest option
min_idx = np.argmin([5/36, 5/9, 35/36, 65/36, 155/36])

# Print the closest option
print(options[min_idx])
</llm-code>
<llm-code-output>
5/6
</llm-code-output>
```

So the answer is $\boxed{5 : 6}$.

Figure 13: Sample solution where the model uses completely flawed reasoning to arrive at the correct answer. Such flawed reasoning is hard to detect, but fortunately, these solutions are rare.

Last Step Mishap

Question

What is the 100th digit to the right of the decimal point in the decimal representation of $\frac{13}{90}$?

Generated Solution

We can use sympy to calculate the decimal representation of $\frac{13}{90}$ and then extract the 100th digit.

```
<llm-code>
from sympy import Rational, N

# Calculate the decimal representation of 13/90
decimal_rep = N(Rational(13, 90), 100)

# Extract the 100th digit
digit = int(str(decimal_rep)[-1])

print(digit)
</llm-code>
<llm-code-output>
4
</llm-code-output>
```

So the 100th digit to the right of the decimal point in the decimal representation of $\frac{13}{90}$ is $\boxed{7}$.

Figure 14: The model performs correct reasoning till code generation. But rather than copying the code output (4) it concludes with a different answer (7).

Timeout

Question

Let $a_1, a_2, \ldots$ be a sequence for which $a_1 = 2$, $a_2 = 3$, and $a_n = \frac{a_{n-1}}{a_{n-2}}$ for each positive integer $n \geq 3$. What is a_{2006}?

Generated Solution

Let's write a function that calculates a_n for a given n.

```
<llm-code>
def a_n(n):
    if n == 1:
        return 2
    elif n == 2:
        return 3
    else:
        return a_n(n-1) / a_n(n-2)

print(a_n(2006))
</llm-code>
<llm-code-output>
None
</llm-code-output>
```

So $a_{2006} = \boxed{1/5}$.

Figure 15: The model generates recursion-based code to solve the problem. While the code is technically correct, the lack of memoization means the code ends up being extremely inefficient and runs into timeout.

Calculation Error

Question

Our basketball team has 12 members, each of whom can play any position. In how many ways can we choose a starting lineup consisting of a center, a power forward, a shooting forward, a point guard, and a shooting guard?

Generated Solution

We can choose a starting lineup in $12 \times 11 \times 10 \times 9 \times 8 = \boxed{11880}$ ways.

Figure 16: Sample solution where the model performs the right text reasoning but makes calculation mistake.

Maximum Executions Reached

Question

In $\triangle ABC$, $AB = 425$, $BC = 450$, and $AC = 510$. An interior point P is then drawn, and segments are drawn through P parallel to the sides of the triangle. If these three segments are of an equal length d, find d.

Generated Solution

Let’s use sympy to solve this problem.

```
<llm-code>
from sympy import symbols, Eq, solve

# Define the variables
AB, BC, AC, d = symbols(’AB BC AC d’)

# Define the equations
eq1 = Eq(AB, 425)
eq2 = Eq(BC, 450)
eq3 = Eq(AC, 510)
eq4 = Eq(AB + BC, 2 * d)
eq5 = Eq(BC + AC, 2 * d)
eq6 = Eq(AC + AB, 2 * d)

# Solve the equations
solutions = solve((eq1, eq2, eq3, eq4, eq5, eq6), (AB, BC, AC, d))
solutions
</llm-code>
<llm-code-output>
[]
</llm-code-output>
```

The solutions are empty, so we can’t solve this problem using sympy.
Let’s try to solve it manually.
We can see that the sum of any two sides of a triangle is greater than the third side.
So we can write the following inequalities:
$AB + BC > AC$
$BC + AC > AB$
$AC + AB > BC$

Let’s rewrite them using the given values:
$425 + 450 > 510$
$450 + 510 > 425$
$510 + 425 > 450$

We can solve these inequalities using sympy:
. . .

Figure 17: The model generates the first code block, which is logically incorrect and results in an empty output. The following text block identifies the problem with the output but performs incorrect reasoning leading up to the generation of the second code block. The model fails to recover from these errors and ends up generating code-text blocks that are verbatim copies of the first code-text block. The generation stops at the sixth code block, which is our limit on the number of code-block executions during inference.

Code Execution Error

Question

The area of a triangle is 600 square feet. Find the altitude, in feet, of the triangle if the length of the corresponding base is 30 feet.

Generated Solution

Let’s use sympy to solve this problem.

```
<llm-code>
from sympy import symbols, Eq, solve

# define the variables
base, altitude = symbols(’base altitude’)

# area of the triangle
area_eq = Eq(base * altitude / 2, 600)

# length of the corresponding base
base_eq = Eq(base, 30)

# solve the equations
solutions = solve((area_eq, base_eq), (base, altitude))

# print the altitude
print(solutions[altitude])
</llm-code>
<llm-code-output>
[0;31m---------------------------[0m
[0;31mTypeError[0m Traceback (most recent call last)
File [0;32m<ipython-input-1-f95732badac7>:16[0m
[1;32m 13[0m solutions [38;5;241m=[39m solve((area_eq, base_eq), (base, altitude)
)
[1;32m 15[0m [38;5;66;03m# print the altitude[39;00m
[0;32m---> 16[0m [38;5;28mprint[39m([43msolutions[49m[43m[[49m[43maltitude[49m[43
m][49m)

[0;31mTypeError[0m: list indices must be integers or slices, not Symbol
</llm-code-output>
```

So the altitude is $\boxed{20}$.

Figure 18: Sample solution where the model generated code runs into execution error. Nevertheless, the model still generates an *incorrect* answer to the question.

Listing 1: Few-shot prompt for GSM8K.

```
Here are some examples of questions and solutions followed by a new question
 that you need to solve.
Make sure to put the answer (and only answer) inside \boxed{}.

Question:
Missy had a giant piggy bank in her bedroom. Every day she would search the
house for change to put in her bank. After 4 years, the bank was opened and
it contained $450 in change. If the second, third, and fourth-year she
doubled the amount of money she put in the bank from the amount she had put
in the previous year, how much money, in dollars, did she put in the bank
the first year?

My solution:
Missy started with some amount of money in the first year.
She was then doubling the amount of money every year for 4 years and ended
up with $450.
Let’s write down an equation for this problem and solve it using sympy.
<llm-code>
from sympy import solve, symbols
first_year_money = symbols(’first_year_money’)
second_year_money = 2 * first_year_money
third_year_money = 2 * second_year_money
fourth_year_money = 2 * third_year_money
# Solve for first year money given that the combined saving is 450
result = solve(first_year_money + second_year_money + third_year_money +
fourth_year_money - 450, first_year_money)[0]
result
</llm-code>
<llm-code-output>
30
</llm-code-output>
Thus Missy put \boxed{30} dollars in the bank the first year.




Question:
Pete has to take a 10-minute walk down to the train station and then board a
 1hr 20-minute train to LA. When should he leave if he cannot get to LA
later than 0900 hours? (24-hr time)

My solution:
Since Pete needs to take a 10 minutes walk and then a 1 hour 20 minutes
train ride, he will spend a total of 1 hour and 30 minutes.
This means that he needs to leave 1 hour and 30 minutes earlier than 09:00
hours.
Subtracting 1 hour and 30 minutes from 09:00 hours we get \boxed{07:30}
hours.




Question:
Mark deposited $88 in a bank. Bryan deposited $40 less than five times as
much as Mark. How much did Bryan deposit in the bank?

My solution:
Let’s solve this problem using Python code.
<llm-code>
mark_deposit = 88
five_times_mark_deposit = 5 * mark_deposit
```

```
bryan_deposit = five_times_mark_deposit - 40
bryan_deposit
</llm-code>
<llm-code-output>
400
</llm-code-output>
So Bryan deposited \boxed{400} dollars in the bank.
```

```
Question:
A department store displays a 20% discount on all fixtures. What will be the
 new price of a 25 cm high bedside lamp that was worth $120?

My solution:
Let’s solve this problem using Python code.
<llm-code>
discount_percent = 20
price_before_discount = 120
discount_portion = discount_percent / 100
discount_amount = price_before_discount * discount_portion
price_after_discount = price_before_discount - discount_amount
price_after_discount
</llm-code>
<llm-code-output>
96.0
</llm-code-output>
So the new price of the lamp is \boxed{96} dollars.
```

```
Question:
James opens up a flower shop. He needs to pay rent of $1200 a week with an
additional 20% of rent to pay for utilities and he has 2 employees per shift
 with the store open 16 hours a day for 5 days a week. If he pays each
employee $12.50 an hour, what are his weekly expenses to run the store?

My solution:
The cost consists of rent, utilities, and employee salaries. Let’s compute
each of them separately and then add them up.
<llm-code>
# rent cost
rent_per_week = 1200
# utility cost
utility_per_week = rent_per_week * 20 / 100
# employee cost
employee_work_hours = 16
work_days_per_week = 5
employee_work_hours_per_week = work_days_per_week * employee_work_hours
number_of_employees = 2
employee_cost_per_hour = 12.5
employees_cost_per_week = number_of_employees * employee_work_hours_per_week
 * employee_cost_per_hour
# add the three to get total cost
cost_per_week = rent_per_week + utility_per_week + employees_cost_per_week
cost_per_week
</llm-code>
<llm-code-output>
3440.0
</llm-code-output>
Thus James’s weekly expences add up to \boxed{3440} dollars.
```

Question:
Tim has 30 less apples than Martha, and Harry has half as many apples as Tim
. If Martha has 68 apples, how many apples does Harry have?

My solution:

Listing 2: Few-shot prompt for MATH.

```
Here are some examples of questions and solutions followed by a new question
 that you need to solve.
Make sure to put the answer (and only answer) inside \boxed{}.

Question:
A parabola with equation $y=x^2+bx+c$ passes through the points $(-1,-11)$
and $(3,17)$. What is $c$?

My solution:
Let’s write down an equation for the parabola and solve for $c$ using sympy.
<llm-code>
import sympy as sp

# define the symbols
x, y, b, c = sp.symbols(’x y b c’)

# define the parabola equation
parabola_eq = sp.Eq(y, x**2 + b*x + c)

# the parabola passes through the points (-1,-11) and (3,17)
# so we substitute these points into the parabola equation
point_1 = parabola_eq.subs({x: -1, y: -11})
point_2 = parabola_eq.subs({x: 3, y: 17})

# we now have two equations and two unknowns (b and c)
# we will solve for b and c
solutions = sp.solve((point_1,point_2), (b, c))
solutions[c]
</llm-code>
<llm-code-output>
-7
</llm-code-output>
So c is \boxed{-7}




Question:
Let $f(x)$ be an odd function. Is $f(f(x))$ even, odd, or neither?

Enter "odd", "even", or "neither".

My solution:
The definition of an odd function is that $f(-x) = -f(x)$.
Applying this to $f(f(-x))$ we get $f(f(-x)) = f(-f(x)) = -f(f(x))$.
Thus, $f(f(x))$ is an \boxed{odd} function.




Question:
At the 2007 Math Olympics, Team Canada won $17$ out of a possible $100$
medals. Which one of the following is closest to the fraction of medals that
 they won? $$
\frac{1}{4} \qquad \frac{1}{5} \qquad \frac{1}{6} \qquad \frac{1}{7} \qquad
\frac{1}{8}
$$

My solution:
Let’s use sympy to print out the differences between the fraction of medals
that Canada won and each of the options.
```

```
<llm-code>
from sympy import Rational, Abs

# team Canada won 17 out of 100 medals
medal_frac = Rational(17, 100)

# list of options
options = [Rational(1, 4), Rational(1, 5), Rational(1, 6), Rational(1, 7),
Rational(1, 8)]

# let’s print out the differences
[Abs(medal_frac - frac_option) for frac_option in options]
</llm-code>
<llm-code-output>
[2/25, 3/100, 1/300, 19/700, 9/200]
</llm-code-output>
Let’s now check which difference is the smallest.
<llm-code>
import numpy as np

# Calculate the idx of the closest option
min_idx = np.argmin([2/25, 3/100, 1/300, 19/700, 9/200])

# Print the closest option
print(options[min_idx])
</llm-code>
<llm-code-output>
1/6
</llm-code-output>
So the answer is \boxed{1/6}.
```

```
Question:
A rectangular box $P$ is inscribed in a sphere of radius $r$. The surface
area of $P$ is 384, and the sum of the lengths of its 12 edges is 112. What
is $r$?

My solution:
Let the dimensions of $P$ be $x$, $y$, and $z$.
The sum of the box’s edges is $4(x + y + z)$ and the surface area is $2xy +
2yz + 2xz$.
The diameter of the sphere is $2r$ and it’s equal to the diagonal of the box
.
Let’s now write down the equations based on the above information and solve
them using sympy.
<llm-code>
from sympy import symbols, Eq, solve

# define the variables
x, y, z, r = symbols(’x y z r’)

# equations based on the given information
eq1 = Eq(2 * (x*y + x*z + y*z), 384)
eq2 = Eq(4 * (x + y + z), 112)

# the diagonal of the box is the diameter of the sphere
diagonal_eq = Eq(x**2 + y**2 + z**2, (2*r)**2)

# solve the equations
solutions = solve((eq1, eq2, diagonal_eq), (x, y, z, r))

# let’s see which values of r we get
```

```
[solution[-1] for solution in solutions]
</llm-code>
<llm-code-output>
[-10, -10, 10, 10]
</llm-code-output>
Since the radius of the sphere is positive, we get $r = \boxed{10}$.




Question:
A bee starts flying from point $P_0$. She flies $1$ inch due east to point
$P_1$. For $j \ge 1$, once the bee reaches point $P_j$, she turns $30^{\circ
}$ counterclockwise and then flies $j+1$ inches straight to point $P_{j+1}$.
 When the bee reaches $P_{2015},$ how far from $P_0$ is she, in inches?

My solution:
We can represent the rotation action via $\omega = e^{\pi i/6}$.
Let’s assume the bee starts at the origin, so $P_{2015}$ is at the point \[z
 = 1 + 2 \omega + 3 \omega^2 + 4 \omega^3 + \dots + 2015 \omega^{2014}.\]
This is an arithmetic-geometric series which we can solve by simplifying the
 expression.
Alternatively, we can solve for |z| using sympy using the following code.
<llm-code>
from sympy import I, pi, exp, sqrt

# rotation action of 30 degrees
omega = exp(I * pi/6)

position = 0

for i in range(2015):
    delta = (i + 1) * omega**(i)
    position += delta

real, img = (position.as_real_imag())
# Distance from origin i.e. starting point
dist = sqrt(real**2 + img**2)
print(dist)
</llm-code>
<llm-code-output>
sqrt(2)*(1008 + 1008*sqrt(3))
</llm-code-output>
So the bee is $\boxed{1008\sqrt{2} + 1008\sqrt{6}}$ far from the starting
point.




Question:
If $f (x) = x^2 - 1$, what is the value of $f (-1)$?

My solution:
```