OpenMathInstruct-1: A 1.8 Million Math Instruction Tuning Dataset

Anonymous ACL submission

Abstract

Recent work has shown the immense potential of synthetically generated datasets for training large language models (LLMs), especially for acquiring targeted skills. Current largescale math instruction tuning datasets such as MetaMathQA (Yu et al., 2024) and MAmmoTH (Yue et al., 2024) 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 closedsource LLMs, such as GPT-4, and the best opensource LLMs. Building on the recent progress in open-source LLMs, our proposed prompting novelty, and some brute-force scaling, we construct OpenMathInstruct-1, a 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. We will release our code, models, and the OpenMathInstruct-1 dataset under a commercially permissive license.

1 Introduction

001

002

007

011

015

017

019

027

042

The huge development and inference costs associated with general-purpose large language models (LLMs) have led to the rise of smaller, taskspecific LLMs. Recent work has proposed creating these domain/task-specific LLMs by generating *high-quality synthetic data* using powerful closedsource models such as GPT-3.5/4 (OpenAI et al., 2023) and training smaller models on the generated *distillation* data (Eldan and Li, 2023; Gunasekar et al., 2023; Li et al., 2023). For mathematical reasoning, our task of interest, all the current state-of-



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 Gou et al. (2024).

043

045

051

054

055

059

060

061

062

063

064

the-art open-source models are *gpt-distilled* (Wang et al., 2024; Yue et al., 2024; Gou et al., 2024; Liao et al., 2024). 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,¹ (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 (Chen et al., 2023a).

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 Mixtral 8x7B model (Jiang et al., 2024), currently one of the best open-source LLMs at mathematical reasoning, by generating *code-interpreter* style solutions for two popular mathematical reasoning benchmarks, namely GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). We use the metric *training set coverage (TSC)* to compare the models, where TSC measures the number of

¹https://openai.com/policies/terms-of-use

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, Meta-MathQA, 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 (Mishra et al., 2022)	272K	- (🗸)
MathInstruct (Yue et al., 2024)	262K	GPT-4 (X)
MetaMathQA (Yu et al., 2024)	395K	GPT-3.5 (X)
MathCodeInstruct (Wang et al., 2024)	80K	GPT-4 + Self (X)
WizardMath* (Luo et al., 2023)	96K	GPT-3.5 (X)
ToRA* (Gou et al., 2024)	16K	GPT-4 (X)
OpenMathInstruct-1 (Ours)	1.8M	Mixtral (✔)

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.

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.² 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 text solutions accompany mathematical benchmarks such

as MATH and GSM8K. These text solutions can aid the synthesis of code-interpreter style solutions. We show that using the text solution in our fewshot prompt with a slight modification substantially increases the coverage and, consequently, the performance of the fine-tuned model (Section 2.2.3).

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

Our solution synthesis experiments result in OpenMathInstruct-1, a collection of 1.8M problemsolution 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 a range of models based on Mistral-7B (Jiang et al., 2023), Llama 2 (Touvron et al., 2023), and CodeLlama (Rozière et al., 2023). In particular, the CodeLlama-70B model fine-tuned on a subset of OpenMathInstruct-1 achieves a score of 84.6% on GSM8K and 50.7% on MATH. These scores are competitive with the best gpt-distilled models. Finally, to support the open-source efforts in this direction, we will publicly release all our fine-tuned models, code, and the OpenMathInstruct-1 dataset along with a further 6.6M incorrect solutions.³

2 Training Data Synthesis

2.1 Overview

Setup. Let $\mathcal{X} = \{(q_1, a_1), \dots, (q_N, a_N)\}$ be a typical mathematical reasoning training dataset, where q_i and a_i denote the *i*th question and answer respectively. Optionally, the training data may include text solution t_i , which illustrates a trajectory from q_i to a_i using mathematical principles.⁴ 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 (Zelikman et al., 2022; Huang et al., 2023). Recent work has also attempted augmenting training set problems (Yue et al., 2024; Yu et al., 2024), 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 (Figure 2). The code-interpreter format interweaves natural

³The incorrect solution trajectories can be used to train verifier models (Cobbe et al., 2021; Lightman et al., 2023).

⁴Both GSM8K and MATH have these text solutions.

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.

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

So the new price of the lamp is 96 dollars.

Figure 2: Code-Interpreter style solution for a training set problem from GSM8K.

language reasoning with Python code blocks. It 148 thus combines the computation precision of cod-149 ing environments with the expressiveness of natu-150 ral language reasoning, which is particularly suit-151 152 able for mathematical tasks (Zhou et al., 2024; Gou et al., 2024). To demarcate the start and end of a code block, we use the strings $\langle 11m-code \rangle$ 154 and $\langle /11m-code \rangle$. A code block is followed 155 by its execution block, which is demarcated by 156 $\langle 1lm-code-output \rangle$ and $\langle /llm-code-output \rangle$. During inference, the model invokes the Python 158 interpreter to run the preceding code block after 159 generating $\langle /11m-code \rangle$, appends the execution re-160 sult in between the $\langle 11m-code-output \rangle$ separators, 161 162 and resumes the autoregressive model inference.⁵

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 163 task, $\{q_1, \cdots, q_K\}$ represent K problems represen-164 tative of the dataset, $\{c_1, \dots, c_K\}$ represent their 165 respective solutions in the code-interpreter format, 166 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 169 to the correct answer for the question q', we add 170 the pair (q', c') to our fine-tuning set. For all our 171

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. Complete instruction is in Table 12 in Appendix.

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.⁶

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.

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 codebased solutions, text-based solutions, and a combination of both. The prompts used for the two datasets are shown in Appendix B.7.

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 (Gou et al., 2024; Liao et al., 2024).

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,

216

217

218

219

172

⁵During training, we don't mask the code execution output surrounded by $\langle 11m$ -code-output \rangle separators.

⁶https://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	# Samples	MATH # Unique Solns.	Coverage (in %)	# Samples	GSM8K # Unique Solns.	Coverage (in %)
Default + Subj	224 224	177K 191K	80.1 80.1	128	434K -	99.1
Mask-Text + Subj	224 224	192K 227K	85.9 87.5	128	602K -	99.9
Total	896	787K	93.0	256	1036K	99.9

and probability (See Table 10 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 subjectspecific 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 $32 \times 7 = 224$ solutions per problem. However, even with this finegrained prompting, we only find a negligible gain in the training set coverage, though the total number of correct solutions increases by 14K (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. But achieving this coverage required sampling almost 450 solutions per problem (224 + 224 = 448). *Can we make the solution pipeline more efficient?*

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

2.2.3 Masked Text Solution Prompting

GSM8K and MATH benchmarks come with groundtruth 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:

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

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 we ran into many creative ways in which the generated solution was cheating (see Figure 9 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 3. 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 $\{t_1^{\text{mask}}, \dots, t_K^{\text{mask}}\}$ represent masked text solutions corresponding to $\{t_1, \dots, t_K\}$. 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 $80.1\% \rightarrow 85.9\%$ for the single *default* prompt, and $80.1\% \rightarrow 87.5\%$ for the subject-specific prompts.

227

229

235

239

252

253

254

255

256

257

258

259

260

261

263



Figure 4: Histogram of the number of solutions for problems in a 64K downsampled subset of MATH instances in OpenMathInstruct-1.

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 described in Appendix B.6.

2.3 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.

2.3.1 Fair vs Naive Downsampling

For a dataset like MATH, where problems can have very different 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.3 for a detailed discussion on solution count). A naive strategy for downsampling treats every instance, i.e., 292 problem-solution pair, as an equal. This problem-293 agnostic sampling perpetuates the imbalance of the original corpus, as seen in Figure 4(a). We propose a *fair* sampling alternate in which we iterate 297 over all the problems round-robin and sample from unpicked solutions for each problem. This problemdependent sampling ensures a more balanced representation of each problem in the downsampled dataset (see Figure 4(b)). Experimental results show 301

that *fair* downsampling outperforms *naive* down-sampling (Section 4.1.1).

302

303

304

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

330

331

332

333

334

335

336

337

338

339

340

341

343

344

345

346

347

348

2.3.2 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 textbased solutions is 2%, but for MATH, their representation is 35.1%.7 While natural language reasoning is more expressive, it lacks the precision of code-based solutions (Gao et al., 2023). Suppose for a problem q there are a total of N_{total} correct solutions in the corpus, out of which $N_{\rm code}$ 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: 1. *Majority-Code*: If $N_{\text{code}} > N_{\text{text}}$, remove all the text-based solutions. 2. 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 codepreferential data selection (Section 4.1.3).

3 Experimental Setup

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-2 (Loshchilov and Hutter, 2019) and dropout (Hinton et al., 2012) 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 (Kuchaiev et al., 2019). Appendix B.1 details the full set of hyperparameters.

Evaluation Setup. We evaluate our models on popular math reasoning benchmarks, namely GSM8K, MATH, GSM-Hard (Gao et al., 2023), SVAMP (Patel et al., 2021), TabMWP (Lu et al., 2023), ASDiv (Miao et al., 2020), and MAWPS (Koncel-Kedziorski et al., 2016). For ablation studies 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 (Wang et al., 2023) for evaluation. For majority

⁷We detect the presence of code by searching for $\langle 11m-code \rangle$ 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
	Llama-2	WizardMath MataMath	54.9	10.7	-	36.1	-	-	-
			50 7 -	$-\frac{19.4}{22.4}$		- 717 -			
			72.6	<i>11.</i>	56.0	71.4	51.6	- 78 7	01.3
	CodeLlama	+ SC (k=50)	76.8	52.5	-	-	51.0	-	-
	Couchiania	OpenMath-CodeLlama	75.9	43.6	60.1	79.6	56.0	77 7	93 5
7B		+ SC (k=50)	84.8	55.6	-	-	-	-	-
		MetaMath-Mistral-7B	-7777	$-\frac{33.0}{282}$					
		MAmmoTH-7B-Mistral	75.0	40.0	_	-	_	_	_
	Mistral	WizardMath	83.2	33.0	-	-	-	-	-
	monu	OpenMath-Mistral-7B	80.2	44 5	63.7	82.4	70.0	82.7	95.4
		+ SC (k=50)	86.9	57.2	-	-	-	-	-
Llama-2	Lloma 2	WizardMath	63.9	14.0	-	51.9	-	-	-
	Liaina-2	MetaMath	72.3	22.4	-	-	-	-	-
		MĀmmoTH	64.7	36.3		73.7			
13B		ToRA	<u>75.8</u>	48.1	<u>60.5</u>	<u>75.7</u>	65.4	81.4	<u>92.5</u>
	CodeLlama	+ SC (k=50)	80.4	55.1	-	-	-	-	-
		OpenMath-CodeLlama	78.8	<u>45.5</u>	61.9	78.8	<u>59.7</u>	81.2	93.6
	+ SC (k=50)	86.8	57.6	-	-	-	-	-	
	MAmmoTH	72.7	43.6	-	84.3	-	-	-	
		ToRA	80.7	51.0	63.7	80.5	70.5	84.2	93.3
34B	CodeLlama	+ 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	-	-	-	-	-
		WizardMath	81.6	22.7	-	71.8	-	-	-
Llama-2 70B		MetaMath	82.3	26.6	-	-	-	-	-
		MAmmoTH	76.9	41.8	-	82.4	-	-	-
	Llama-2	ToRA	84.3	49.7	67.2	82.7	<u>74.0</u>	86.8	93.8
		+ SC (k=50)	88.3	56.9	-	-	-	-	-
		OpenMath-Llama2	84.7	46.3	65.7	<u>85.0</u>	70.8	84.3	<u>95.6</u>
		+ SC (k=50)	90.1	58.3	-	-	-	-	-
	CodeL lama	OpenMath-CodeLlama	84.6	50.7	66.6	87.8	74.2	84.7	<u>9</u> 5.7
Couelianna	+ SC (k=50)	90.8	60.4	-	-	-	-	-	

voting, we found that using the 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 answering questions after code execution errors.

4 **Results**

351

352

355

361

362

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 2.3).⁸ Thus, the total finetuning corpus size is roughly 1.2M. We will justify the data selection choice 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.⁹ Our models easily outperform both MAmmoTH and MetaMath, 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 MAmmoTH and Meta-Math substantially. Compared to ToRA, with

364

- 372 373
- 374 375 376
- 377

⁹Our 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.

⁸The actual number of MATH instances is 511,677.

Table 4: Comparison of performance of fair vs naive sampling on our validation subset of GSM8K and MATH.

Prompt	GSM8K	MATH
Naive	74.3	35.0
Fair	75.3	37.0

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 gains with SC
can be attributed to our diverse 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.

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 Impact of Fine-Tuning Dataset Size

Table 5: 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

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 subsets of GSM8K and MATH. Table 5 shows that the performance increases on both GSM8K and MATH with the increase in the fine-tuning dataset size. We didn't find benefit from training the models for more steps, so the performance gain is attributable to the increased data size.

4.1.3 MATH-only Ablations

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

Table 6: Comparison of default vs subject-wise promptperformance on our MATH validation subset.

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

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 subjectwise 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. 414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

Table 6 compares the finetuning performance on these two splits on our MATH validation subset. While the model trained on the *subject-specific* subset trails the model trained on the *default* subset for greedy decoding; the trend is decisively reversed for self-consistent decoding with four samples. This suggests that the subset generated with subject-specific prompts has a higher diversity than the ones generated using a single prompt.

Table 7: 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

Code-Preferential Subsets. In this ablation, we determine the impact of code-preferential solution selection strategies proposed in Section 2.3.2. Table 7 shows that code-preferential solution strategies aid the greedy decoding performance. However, the reduction in solution diversity arguably results in decreased performance with self-consistency decoding (text-based solutions are only 1/3rd of the original corpus to begin with). 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 in our finetuning data blend.

5 Analysis

We analyze the performance of the ablation model trained on 512K instances from Section 4.1.2. We focus our discussion on the MATH benchmark where this model scores 41.6% on our MATH validation subset.

408

409

410

411

412

413



(a) Subject-wise performance (b) Level-wise performance

Figure 5: Performance split by subjects and levels on our MATH validation subset.

Performance-split by Subjects and Levels. Figure 5 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 (Zhou et al., 2024). We see a monotonic decrease in performance with the increase in hardness level which is to be expected (Zhou et al., 2024).

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

Error Analysis. Table 8 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 solutions could have been avoided by preferring code-based solutions; see Figure 15 in the Appendix for a sample solution. This analysis provides another support for our proposal and use of code-preferred solutions from Section 2.3.2.

Table 9 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. Sample solutions from these error types are presented in Appendix B.3.

6 Related Work

Recently, a plethora of work has been done on enhancing the mathematical reasoning capabilities of LLMs. Inference techniques such as Chain-of-Thought (Wei et al., 2022), its programmatic counterpart, Program of Thought (Gao et al., 2023; Chen et al., 2023b), Self-Consistency (Wang et al., 2023),

Table 8: Performance	split	based	on	solution	format.
----------------------	-------	-------	----	----------	---------

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

Table 9: 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

and Self-Verification (Zhou et al., 2024) have been shown to significantly improve the reasoning capabilities of LLMs without any further training. 483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

Pretraining language models on math-heavy content has resulted in foundation LLMs such as Minerva (Lewkowycz et al., 2022), Galactica (Taylor et al., 2022), and Llemma (Azerbayev et al., 2023) 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 (Yuan et al., 2023), ToRA (Gou et al., 2024), MAmmoTH (Yue et al., 2024), MetaMath (Yu et al., 2024) and Math-Coder (Wang et al., 2024). 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.

7 Conclusion

We introduce OpenMathInstruct-1, a math instruction tuning dataset with 1.8M problem-solution pairs with a commercially permissive license. Compared to previous work, OpenMathInstruct-1 is at least four times bigger. The problems are taken from the training set of GSM8K and MATH benchmarks, and the solutions are synthesized by fewshot prompting the Mixtral model. With our proposed prompting novelty of using masked text so*lutions* 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 will publicly release all our fine-tuned models, code, and the OpenMathInstruct-1 along with a further 6.6M incorrect sampled solutions.

527

530

531

534

536

541

542

543

544

546

547

548

552

553

556

557

558

560

561

562

563

564

566

567

569

570

8 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 11 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

- Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An Open Language Model For Mathematics.
- Lingjiao Chen, Matei Zaharia, and James Zou. 2023a. How is ChatGPT's behavior changing over time?
- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023b. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *TMLR*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems.
- Ronen Eldan and Yuanzhi Li. 2023. TinyStories: How Small Can Language Models Be and Still Speak Coherent English? *arXiv*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided Language Models. In *ICML*, pages 10764–10799. PMLR. 571

572

573

574

575

576

577

578

579

580

582

583

584

585

586

587

589

590

591

592

593

594

595

596

597

598

599

600

601

602

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

- Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving. In *ICLR*.
- 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. 2023. Textbooks Are All You Need. *arXiv*.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. *NeurIPS*.
- Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing coadaptation of feature detectors. *arXiv preprint arXiv:1207.0580.*
- Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large Language Models Can Self-Improve. In *EMNLP*.
- 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. 2023. Mistral 7B. *arXiv*.
- 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. 2024. Mixtral of Experts.
- Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A Math Word Problem Repository. In *NAACL-HLT*.
- O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook, et al. 2019. NeMo: a toolkit for building AI applications using neural modules. In Systems for ML Workshop, NeurIPS.
- Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem

627

679

Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving Quantitative Reasoning Problems with Language Models. In NeurIPS.

- Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023. Textbooks Are All You Need II: phi-1.5 technical report. arXiv.
- Minpeng Liao, Wei Luo, Chengxi Li, Jing Wu, and Kai Fan. 2024. MARIO: MAth Reasoning with code Interpreter Output - A Reproducible Pipeline.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's Verify Step by Step. arXiv.
- Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. 2023. TinyGSM: achieving> 80% on GSM8k with small language models. arXiv preprint arXiv:2312.09241.
- Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. arXiv.
- Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023. Dynamic Prompt Learning via Policy Gradient for Semi-structured Mathematical Reasoning. In ICLR.
- Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. WizardMath: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct. arXiv preprint arXiv:2308.09583.
- Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers. In ACL.
- Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. 2022. Lila: A Unified Benchmark for Mathematical Reasoning. In EMNLP.
- Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive Learning from Complex Explanation Traces of GPT-4.
- 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, 682 Chelsea Carlson, Rory Carmichael, Brooke Chan, 683 Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, 685 Chester Cho, Casey Chu, Hyung Won Chung, Dave 686 Cummings, Jeremiah Currier, Yunxing Dai, Cory 687 Decareaux, Thomas Degry, Noah Deutsch, Damien 688 Deville, Arka Dhar, David Dohan, Steve Dowling, 689 Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna 690 Eloundou, David Farhi, Liam Fedus, Niko Felix, 691 Simón Posada Fishman, Juston Forte, Isabella Ful-692 ford, Leo Gao, Elie Georges, Christian Gibson, Vik 693 Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-694 Lopes, Jonathan Gordon, Morgan Grafstein, Scott 695 Gray, Ryan Greene, Joshua Gross, Shixiang Shane 696 Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, 697 Yuchen He, Mike Heaton, Johannes Heidecke, Chris 698 Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, 699 Brandon Houghton, Kenny Hsu, Shengli Hu, Xin 700 Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun 702 Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo 703 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, 707 Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, 709 Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai 710 Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, 711 Chak Ming Li, Rachel Lim, Molly Lin, Stephanie 712 Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, 713 Patricia Lue, Anna Makanju, Kim Malfacini, Sam 714 Manning, Todor Markov, Yaniv Markovski, Bianca 715 Martin, Katie Mayer, Andrew Mayne, Bob McGrew, 716 Scott Mayer McKinney, Christine McLeavey, Paul 717 McMillan, Jake McNeil, David Medina, Aalok Mehta, 718 Jacob Menick, Luke Metz, Andrey Mishchenko, 719 Pamela Mishkin, Vinnie Monaco, Evan Morikawa, 720 Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, 721 David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev 722 Nayak, Arvind Neelakantan, Richard Ngo, Hyeon-723 woo Noh, Long Ouyang, Cullen O'Keefe, Jakub 724 Pachocki, Alex Paino, Joe Palermo, Ashley Pantu-725 liano, Giambattista Parascandolo, Joel Parish, Emy 726 Parparita, Alex Passos, Mikhail Pavlov, Andrew 727 Peng, Adam Perelman, Filipe de Avila Belbute Peres, 728 Michael Petrov, Henrique Ponde de Oliveira Pinto, 729 Michael, Pokorny, Michelle Pokrass, Vitchyr Pong, 730 Tolly Powell, Alethea Power, Boris Power, Eliza-731 beth Proehl, Raul Puri, Alec Radford, Jack Rae, 732 Aditya Ramesh, Cameron Raymond, Francis Real, 733 Kendra Rimbach, Carl Ross, Bob Rotsted, Henri 734 Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, 735 Shibani Santurkar, Girish Sastry, Heather Schmidt, 736 David Schnurr, John Schulman, Daniel Selsam, Kyla 737 Sheppard, Toki Sherbakov, Jessica Shieh, Sarah 738 Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, 739 Maddie Simens, Jordan Sitkin, Katarina Slama, Ian 740 Sohl, Benjamin Sokolowsky, Yang Song, Natalie 741 Staudacher, Felipe Petroski Such, Natalie Summers, 742 Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine 743 Thompson, Phil Tillet, Amin Tootoonchian, Eliz-744 abeth Tseng, Preston Tuggle, Nick Turley, Jerry 745

746

- 801

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, Jiavi 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. 2023. GPT-4 Technical Report.

- Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP Models really able to Solve Simple Math Word Problems? In NAACL-HLT.
- 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. 2023. Code Llama: Open Foundation Models for Code. arXiv.
- Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A Large Language Model for Science.
- 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. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models.
 - Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2024. MathCoder: Seamless Code Integration in LLMs for Enhanced Mathematical Reasoning. In ICLR.
 - Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In ICLR.

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

806

807

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

- Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. WizardLM: Empowering Large Language Models to Follow Complex Instructions. arXiv.
- Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Meta-Math: Bootstrap Your Own Mathematical Questions for Large Language Models. In ICLR.
- Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling Relationship on Learning Mathematical Reasoning with Large Language Models.
- Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2024. MAmmoTH: Building math generalist models through hybrid instruction tuning. In ICLR.
- Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. STaR: Bootstrapping Reasoning With Reasoning. In NeurIPS.
- Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2024. Solving Challenging Math Word Problems Using GPT-4 Code In Interpreter with Code-based Self-Verification. ICLR.

A Data Statistics

A.1 MATH Training Data Split by Subjects

Table 10: 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

A.2 Code-Block Count Frequencies



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

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. 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.3 Frequency of solutions per problem in OpenMathInstruct-1

Figure 12 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-

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 = -5/2x = 1</llm-code-output> So d = -5/2 and e = 1. Let's calculate the answer: <llm-code> d = -5/2e = 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 |-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.

law curve, with most problems having very few solutions and very few problems with many solutions. Concretely, 19% problems (1324 out of 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

857

858

859

860

861

862

863

864

865

838

839

Cod	e +	Text	Reas	oning
				$\overline{\boldsymbol{\mathcal{U}}}$

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 $a \le x \le b$. 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 < 00)) | ((-00 < 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 8: 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.

sampled solutions), that are *valid*.

B Miscellaneous

867

870

B.1 Training Hyperparameters

Table 11: Details of training hyperparameters for finetuning the different base models. LR=Learning rate, TP=Tensor Parallel, PP=Pipeline Parallel.

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

Table 11 details the hyperparameters used for finetuning the different base models. Due to com-

Question	
Jolene and P birthday. Th years ago. T one year after to their third child. Two y gave birth to fourth child?	hil have four children, each with the same ey gave birth to their first child exactly 15 hey gave birth to their second child exactly r the birth of their first child. They gave birth child on the fourth birthday of their second ears after the birth of their third child, they their fourth child. How old, in years, is their
Reference	Solution
years old. Their 2nd ch child, and the Their 3rd chi child, and the Their 4th chi child, and the	d was born 15 years ago, and therefore is 15 iild was born 1 year after their 15-year-old erefore is 15-1=14 years old. ild was born 4 years after their 14-year-old erefore is 14-4=10 years old. ild was born 2 years after their 10-year-old erefore is 10-2=8 years old.
Generated	Solution
<pre><li< td=""><td>irst', 'second', 'third', 'fourth'] [15, 14, 10, 8] children = len(children) ildren[:-1] # let's get rid of the youngest already know that one dict(zip(children, child_age)) butput> econd': 14, 'third': 10} output></td></li<></pre>	irst', 'second', 'third', 'fourth'] [15, 14, 10, 8] children = len(children) ildren[:-1] # let's get rid of the youngest already know that one dict(zip(children, child_age)) butput> econd': 14, 'third': 10} output>

Shortcut Solution

Figure 9: 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.

pute 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 (Gou et al., 2024). For the Mistral-7B model, we use a lower learning rate of 1e-6 based on our hyperparameter search. 871

872

873

874

875

876

877

878

879

B.2 Sample Solutions

13

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

• Figure 8 shows a sample solution that utilizes the strength of the code-interpreter solution 882

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) = 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]

Trimmed Solution

</llm-code>

894

895

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) = 66.

Figure 10: Sample solution where the model generates unrelated text after answering the question. During postprocessing, we trim the solution by removing text beyond the solution line with the answer.

format with reasoning in both code and natural language.

Figure 9 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 11 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 (Gu-

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 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 5:6

Figure 11: 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.

nasekar et al., 2023). We leave the work of developing such semantic filters for future work.

• Figure 10 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 12: Histogram of the number of solutions for problems in GSM8K and MATH.

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

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921 922

924

925

926

927

928

929

930

931

932

933

934

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

- Figure 13 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.
- 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 ends up generating a new answer.
- Figure 15 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 choice of preferring code-based solutions.

B.4 Instructions for Few-shot Data Generation

Table 12 details the instructions used for the different generation tasks.

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.

Table 12: Instructions for prompting the model.

Task	Instruction
Few-shot prompt (<i>I</i>)	Here are some examples of questions and solutions followed by a new ques- tion that you need to solve. Make sure to put the answer (and only answer) inside .
Few-shot prompt text masking (\mathcal{I}_{mask})	Here are some examples of questions, solutions, and their masked solutions followed by a new question and solu- tion that you need to mask. The goal is to ensure that the masked solution doesn't have any of the numerical val- ues 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 model	System: You're an expert Python pro- grammer and mathematician. Help the user to solve this problem using code when necessary. Make sure to put the answer (and only answer) in- side .

• Filter out solutions that have the final answer. This ensures that our masked solutions are at least masking the final answer.

935

936

937

938

939

940

941

942

• 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.

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])
</lin-code>
(lim-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([43
msolutions[49m[43m][49m[43maltitude[49m[43m][49
m)

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

So the altitude is 20.

Figure 13: Sample solution where the model generated code runs into execution error. Nevertheless, the model still generates an *incorrect* answer to the question.

B.6 Post-processing

943

945

947

951

953

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

- The solution has multiple answers as it has multiple \boxed{} blocks. We remove such solutions.
- The solution has the (llm-code) string but not the (/llm-code) string. We remove such solutions.
- The solution continues even after generating

Last	Step	Mishap
		1.11011000

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>

</llm-code-output>

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



Figure 15: Sample solution where the model performs the right text reasoning but makes calculation mistake.

the answer, i.e., the \boxed{} block. While in some cases, this continuation merely concludes the answer, we noticed that continuations that went beyond two lines were almost always gibberish generated by the LLM. We removed the text in the lines beyond the solution line with the answer. See Figure 10 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 (Cobbe et al., 2021), 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 11 in the Appendix for a sample *semantically noisy* solution.

971B.7Few-shot prompts used for GSM8K and972MATH

Listing 1: Few-shot prompt for GSM8K. Here are some examples of questions and solutions followed by a new question that you need to 975 solve. 976 Make sure to put the answer (and only answer) inside \boxed{}. 977 978 Ouestion: Missy had a giant piggy bank in her bedroom. Every day she would search the house for change 979 to put in her bank. After 4 years, the bank was opened and it contained \$450 in change. If 981 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 983 bank the first year? 984 985 My solution: 986 Missy started with some amount of money in the first year. 987 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') 992 second_year_money = 2 * first_year_money 993 third_year_money = 2 * second_year_money fourth_year_money = 2 * third_year_money 994 995 # Solve for first year money given that the combined saving is 450 996 result = solve(first_year_money + second_year_money + third_year_money + fourth_year_money -997 450, first_year_money)[0] 998 result </llm-code> 999 <llm-code-output> 1001 30 1002 </llm-code-output> 1003 Thus Missy put \boxed{30} dollars in the bank the first year. 1004 1005 1006 1007 1008 1009 Ouestion: 1010 Pete has to take a 10-minute walk down to the train station and then board a 1hr 20-minute 1011 train to LA. When should he leave if he cannot get to LA later than 0900 hours? (24-hr time) 1012 1013 My solution: Since Pete needs to take a 10 minutes walk and then a 1 hour 20 minutes train ride, he will 1014 1015 spend a total of 1 hour and 30 minutes. 1016 This means that he needs to leave 1 hour and 30 minutes earlier than 09:00 hours. 1017 Subtracting 1 hour and 30 minutes from 09:00 hours we get \boxed{07:30} hours. 1018 1019 1020 1021 1023 Ouestion: 1024 Mark deposited \$88 in a bank. Bryan deposited \$40 less than five times as much as Mark. How 1025 much did Bryan deposit in the bank? 1026 1027 My solution: 1028 Let's solve this problem using Python code. <llm-code> 1030 mark_deposit = 88 1031 five_times_mark_deposit = 5 * mark_deposit 1032 bryan_deposit = five_times_mark_deposit - 40 1033 bryan_deposit 1034 </llm-code> 1035 <llm-code-output> 1036 400 1037 </llm-code-output> So Bryan deposited \boxed{400} dollars in the bank. 1039 1040 1041

```
1043
Question:
                                                                                                            1044
A department store displays a 20% discount on all fixtures. What will be the new price of a
                                                                                                            1045
                                                                                                            1046
25 cm high bedside lamp that was worth $120?
                                                                                                            1047
My solution:
                                                                                                            1048
Let's solve this problem using Python code.
                                                                                                            1049
<llm-code>
                                                                                                            1050
discount_percent = 20
                                                                                                            1051
price_before_discount = 120
                                                                                                            1052
discount_portion = discount_percent / 100
                                                                                                            1053
discount_amount = price_before_discount * discount_portion
                                                                                                            1054
price_after_discount = price_before_discount - discount_amount
                                                                                                            1055
price_after_discount
                                                                                                            1056
</llm-code>
                                                                                                            1057
<llm-code-output>
                                                                                                            1058
96.0
                                                                                                            1059
</llm-code-output>
                                                                                                            1060
So the new price of the lamp is \boxed{96} dollars.
                                                                                                            1061
                                                                                                            1062
                                                                                                            1063
                                                                                                            1064
                                                                                                            1065
                                                                                                            1066
                                                                                                            1067
Ouestion:
James opens up a flower shop. He needs to pay rent of $1200 a week with an additional 20% of
                                                                                                            1068
rent to pay for utilities and he has 2 employees per shift with the store open 16 hours a day
                                                                                                            1069
for 5 days a week. If he pays each employee $12.50 an hour, what are his weekly expenses to
                                                                                                            1070
                                                                                                            1071
run the store?
                                                                                                            1072
My solution:
                                                                                                            1073
The cost consists of rent, utilities, and employee salaries. Let's compute each of them
                                                                                                            1074
separately and then add them up.
                                                                                                            1075
<llm-code>
                                                                                                            1076
                                                                                                            1077
# rent cost
rent_per_week = 1200
                                                                                                            1078
                                                                                                            1079
# utility cost
utility_per_week = rent_per_week * 20 / 100
                                                                                                            1080
# employee cost
                                                                                                            1081
                                                                                                            1082
employee_work_hours = 16
work_days_per_week = 5
                                                                                                            1083
employee_work_hours_per_week = work_days_per_week * employee_work_hours
                                                                                                            1084
number_of_employees = 2
                                                                                                            1085
                                                                                                            1086
employee_cost_per_hour = 12.5
employees_cost_per_week = number_of_employees * employees_work_hours_per_week *
                                                                                                            1087
employee_cost_per_hour
                                                                                                            1088
# add the three to get total cost
                                                                                                            1089
cost_per_week = rent_per_week + utility_per_week + employees_cost_per_week
                                                                                                            1090
cost_per_week
                                                                                                            1091
</llm-code>
                                                                                                            1092
<llm-code-output>
                                                                                                            1093
3440.0
                                                                                                            1094
</llm-code-output>
                                                                                                            1095
Thus James's weekly expences add up to \boxed{3440} dollars.
                                                                                                            1096
                                                                                                            1097
                                                                                                            1098
                                                                                                            1099
                                                                                                            1100
                                                                                                            1101
Question:
                                                                                                            1102
Tim has 30 less apples than Martha, and Harry has half as many apples as Tim. If Martha has
                                                                                                            1103
68 apples, how many apples does Harry have?
                                                                                                            1104
                                                                                                            1105
My solution:
                                                                                                            1109
```

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 $boxed{-7}$ Ouestion: 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? \$\$ $frac{1}{4} \qquad frac{1}{5} \qquad frac{1}{6} \qquad frac{1}{7} \qquad frac{1}{8}$ \$\$ Mv 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

1108 1109

1110

1111 1112 1113

1114 1115

1116 1117

1118

1119

1120

1121

1122 1123

1124 1125

1126

1127 1128

1129

1130

1131 1132 1133

1134

1135

1136

1137

1138

1139

1140

1141

1148

1149 1150

1151 1152

1153

1154

1155

1162

1163

1164 1165

1166 1167

1168

1169

1170

1171

1172

1173

options = [Rational(1, 4), Rational(1, 5), Rational(1, 6), Rational(1, 7), Rational(1, 8)] 1177 1178 # let's print out the differences 1179 [Abs(medal_frac - frac_option) for frac_option in options] 1180 </lim-code> 1181 <llm-code-output> 1182 [2/25, 3/100, 1/300, 19/700, 9/200] 1183 </llm-code-output> 1184 Let's now check which difference is the smallest. 1185 <llm-code> 1186 import numpy as np 1187 1188 # Calculate the idx of the closest option 1189 min_idx = np.argmin([2/25, 3/100, 1/300, 19/700, 9/200]) 1190 1191 # Print the closest option 1192 print(options[min_idx]) 1193 1194 </llm-code> <llm-code-output> 1195 1196 1/6 </llm-code-output> 1197 So the answer is $boxed{1/6}$. 1198 1199 1200 1201 1202 1203 Question: 1204 A rectangular box \$P\$ is inscribed in a sphere of radius \$r\$. The surface area of \$P\$ is 384, 1205 and the sum of the lengths of its 12 edges is 112. What is \$r\$? 1206 1207 My solution: 1208 Let the dimensions of P be x, y, and z. 1209 The sum of the box's edges is 4(x + y + z) and the surface area is 2xy + 2yz + 2xz. 1210 The diameter of the sphere is \$2r\$ and it's equal to the diagonal of the box. 1211 Let's now write down the equations based on the above information and solve them using sympy. 1212 <llm-code> 1213 from sympy import symbols, Eq, solve 1214 1215 # define the variables 1216 x, y, z, r = symbols('x y z r')1217 1218 1219 # equations based on the given information eq1 = Eq(2 * (x*y + x*z + y*z), 384)1220 eq2 = Eq(4 * (x + y + z), 112)1221 1222 # the diagonal of the box is the diameter of the sphere 1223 diagonal_eq = Eq(x**2 + y**2 + z**2, (2*r)**2)1224 1225 # solve the equations 1226 solutions = solve((eq1, eq2, diagonal_eq), (x, y, z, r)) 1227 1228 # let's see which values of r we get 1229 [solution[-1] for solution in solutions] 1230 </llm-code> 1231 <llm-code-output> 1232 [-10, -10, 10, 10] 1233 </llm-code-output> 1234 Since the radius of the sphere is positive, we get $r = \boxed{10}$. 1235 1236 1237 1238 1239 1240 1241 Ouestion: A bee starts flying from point \$P_0\$. She flies \$1\$ inch due east to point \$P_1\$. For \$j \ge 1242 1\$, once the bee reaches point P_j , she turns 30° counterclockwise and then flies 1243 1244 j+1 inches straight to point P_{j+1} . When the bee reaches P_{2015} , how far from P_0 is she, in inches? 1245 1246

```
My solution:
1247
1248
                 We can represent the rotation action via \ = e^{\pi i/6}.
1249
                 Let's assume the bee starts at the origin, so P_{2015} is at the point [z = 1 + 2 + 2 + 2]
1250
                  3 \omega^2 + 4 \omega^3 + \dots + 2015 \omega^{2014}.\]
1251
                  This is an arithmetic-geometric series which we can solve by simplifying the expression.
1252
                  Alternatively, we can solve for |z| using sympy using the following code.
1253
                  <llm-code>
1254
                  from sympy import I, pi, exp, sqrt
1255
1256
                 # rotation action of 30 degrees
1257
                 omega = exp(I * pi/6)
1258
                 position = 0
1259
1260
1261
                  for i in range(2015):
1262
                     delta = (i + 1) * omega**(i)
1263
                     position += delta
1264
1265
                 real, img = (position.as_real_imag())
1266
                  # Distance from origin i.e. starting point
                  dist = sqrt(real**2 + img**2)
1267
1268
                  print(dist)
1269
                  </llm-code>
1270
                  <llm-code-output>
1271
                  sqrt(2)*(1008 + 1008*sqrt(3))
1272
                  </llm-code-output>
                 So the bee is \boxed{1008\sqrt{2} + 1008\sqrt{6}}\ far from the starting point.
1273
1274
1275
1276
1277
1278
1279
                  Question:
1280
                 If f(x) = x^2 - 1, what is the value of f(-1)?
1281
                 My solution:
1283
```