MARIO: MAth Reasoning with code Interpreter Output – A Reproducible Pipeline

Minpeng Liao[*] and **Chengxi Li**[*] and **Wei Luo**[*] and **Jing Wu**[*] and **Kai Fan**[†]
Alibaba Group
`{minpeng.lmp,xiji.lcx,muzhuo.lw,wj334275,k.fan}@alibaba-inc.com`

Abstract

Large language models (LLMs) have significantly improved in understanding natural language but still lack in mathematical reasoning, a hurdle on the path to true artificial general intelligence. The training of large language models, based on next-token prediction, struggles to capture the precise nature of mathematical reasoning, presenting both practical and theoretical challenges. In this paper, we address this challenge by enriching the data landscape and introducing a reasonable data format, enhanced the text analysis of the LLM with a capability to utilize a Python code interpreter. This dataset is derived from GSM8K and MATH and has been further refined through a combination of GPT annotations, human review, and self-training processes. Additionally, we propose a tentative, easily replicable protocol for the fine-tuning of math-specific LLMs, which has led to a significant improvement in the performance of a 7B-parameter LLM on the GSM8K and MATH datasets. A solution generator and a value estimator are fine-tuned simultaneously in a multi-task fashion, while an outlier-free value model-based inference method is proposed to further boost the performance. We are committed to advancing the field of mathematical reasoning in LLMs. and, to that end, we will make the source code and checkpoints publicly available.[1]

1 Introduction

The Chain-of-Thought (CoT) prompting technique (Wei et al., 2022) has been empirically shown to enhance the complex reasoning capabilities of large language models (LLMs) by generating a sequence of intermediate reasoning steps. Proprietary LLMs, including GPT-4(OpenAI, 2023) and Claude-2 (Anthropic, 2023), are designed to produce CoT responses by default, leading to improved reasoning performance, as evidenced by a 50.36% accuracy rate on the MATH dataset (Hendrycks et al., 2021) with GPT-4 (Zhou et al., 2023). Moreover, when LLMs are augmented with a plugin capable of executing code snippets, their accuracy in arithmetic computations—which are typically challenging for LLMs—is further enhanced, *e.g.*, GPT-4-Code achieved a 69.69% accuracy rate on the MATH dataset (Zhou et al., 2023). This underscores the efficacy of integrating text analysis with code execution in datasets designed for math reasoning tasks.

Recently, Yang et al. (2023) demonstrated that, despite extensive fine-tuning for arithmetic operation simulation task, LLMs are still unable to achieve perfect accuracy. Consequently, integrating code for precise numerical computation has become an inevitable trend. The Program-of-Thought (PoT) framework (Chen et al., 2022) and Program-Aided Language (PAL) models (Gao et al., 2023) represent seminal efforts in generating code-centric datasets. Building on this trend, ToRA (Gou et al., 2023) advances the field by employing a proprietary annotation methodology along with GPT-4, setting a new benchmark for state-of-the-art performance in solving mathematical problems via python code. However, it is noteworthy that the solutions contained within these datasets predominantly consist of code snippets, with minimal accompanying textual analysis. An obvious shortcoming of a code-centric solution is that it can overlook common sense in math word problems, *e.g.*,, as shown in Figure 1, the PoT solutions disregard the fact that the quantity of the food taken cannot be negative.

MathCoder (Wang et al., 2023a) represents an initiative that emulates the response patterns of GPT-4, integrating a plugin proficient in both code generation and natural language reasoning. This approach harnesses the capabilities of GPT-

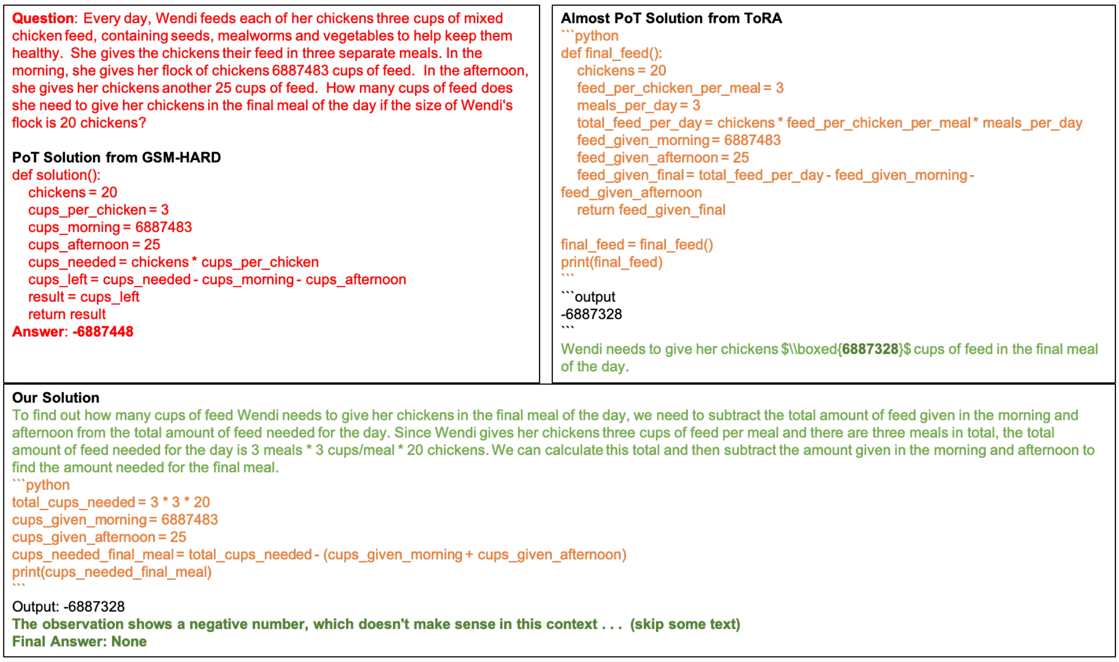
---

[*] equal contribution

[†] corresponding author

[1] https://github.com/MARIO-Math-Reasoning/MARIO

Findings of the Association for Computational Linguistics ACL 2024, pages 905–924
August 11-16, 2024 ©2024 Association for Computational Linguistics



Question: Every day, Wendi feeds each of her chickens three cups of mixed chicken feed, containing seeds, mealworms and vegetables to help keep them healthy. She gives the chickens their feed in three separate meals. In the morning, she gives her flock of chickens 6887483 cups of feed. In the afternoon, she gives her chickens another 25 cups of feed. How many cups of feed does she need to give her chickens in the final meal of the day if the size of Wendi's flock is 20 chickens?

PoT Solution from GSM-HARD

````
def solution():
    chickens = 20
    cups_per_chicken = 3
    cups_morning = 6887483
    cups_afternoon = 25
    cups_needed = chickens * cups_per_chicken
    cups_left = cups_needed - cups_morning - cups_afternoon
    result = cups_left
    return result
````

Answer: **-6887448**

Almost PoT Solution from ToRA

~~~
```python
def final_feed():
    chickens = 20
    feed_per_chicken_per_meal = 3
    meals_per_day = 3
    total_feed_per_day = chickens * feed_per_chicken_per_meal * meals_per_day
    feed_given_morning = 6887483
    feed_given_afternoon = 25
    feed_given_final = total_feed_per_day - feed_given_morning -
feed_given_afternoon
    return feed_given_final

final_feed = final_feed()
print(final_feed)
```
```output
-6887328
```
~~~

Wendi needs to give her chickens $\\boxed{**6887328**}$ cups of feed in the final meal of the day.

Our Solution

To find out how many cups of feed Wendi needs to give her chickens in the final meal of the day, we need to subtract the total amount of feed given in the morning and afternoon from the total amount of feed needed for the day. Since Wendi gives her chickens three cups of feed per meal and there are three meals in total, the total amount of feed needed for the day is 3 meals * 3 cups/meal * 20 chickens. We can calculate this total and then subtract the amount given in the morning and afternoon to find the amount needed for the final meal.

~~~
```python
total_cups_needed = 3 * 3 * 20
cups_given_morning = 6887483
cups_given_afternoon = 25
cups_needed_final_meal = total_cups_needed - (cups_given_morning + cups_given_afternoon)
print(cups_needed_final_meal)
```
~~~

Output: -6887328

The observation shows a negative number, which doesn't make sense in this context . . . (skip some text)

Final Answer: None

Figure 1: A comparison between code-centric solutions and our text-code solution. In the PoT solution provided by GSM-Hard (Gao et al., 2023), the code-centric approach yields a negative quantity of cups. The PoT solution from ToRA omits the minus sign in its concluding sentence, failing to address the impractical issue. Our solution, incorporating both text analysis and a code snippet, recognizes that a negative quantity of cups is illogical.

4, equipped with a code interpreter, to automate the annotation process, producing a blend of text analyses and code snippets, guided by appropriate instructions. Our data generation approach generally mirrors that of MathCoder, but with the enhancement of an extra layer of human verification specifically for the correction of easily rectifiable errors. Therefore, all mistakes in the original GSM8K (Cobbe et al., 2021a) training set have been corrected by hand, while the MATH dataset, containing tougher problems, are partially fixed and required professional annotators' expertise. To address this, we utilize self-training and knowledge distillation techniques to selectively identify correct solutions, in conjunction with more samplings.

Furthermore, we introduce a tentative and easily replicable protocol for the fine-tuning of math-specific LLMs. To ensure the reproducibility of our experiments, we begin with the extensively studied LLM, Llama-2 (Touvron et al., 2023), and utilize its math-oriented continual pre-training variant, Llemma (Azerbayev et al., 2023). Subsequently, we apply supervised fine-tuning to our annotated dataset to establish a baseline model. Moreover, we implement a toolkit for evaluating math answers, which allows for the comparison between the ground truth and the LLM predictions.

Unlike text generation tasks such as summarization, mathematical reasoning usually yields a unique answer, which simplifies the verification of its correctness. However, assessing the reasoning process that leads to the final answer remains challenging. To address this issue, Lightman et al. (2023) introduced a model supervised by processes and a corresponding dataset with manually labeled solution procedures. Yet, from a replication standpoint, creating these annotations is both labor-intensive and costly. As a compromise, we recommend training a straightforward outcome-supervised model as described by (Cobbe et al., 2021b; Yu et al., 2023a), to serve as an auxiliary tool for comparing and selecting the best among various solutions. Our outcome value model (OVM) is fine-tuned using the efficient LoRA training (Hu et al., 2021) in a multi-task setting, which enables the model to conserve computational resources while maintaining its generative capabilities. In summary, our main contributions are in three-fold.

1. We create a math reasoning format that integrates both text analyses and code snippets, leveraging logical reasoning and precise computation.

2. We introduce a reproducible pipeline for data generation and LLM fine-tuning in the mathematical domain.

3. Our experiments demonstrate that our approach can significantly enhance performance on math reasoning tasks. We will open-source our pipeline and model checkpoints.

2 Dataset

In this section, we detail our methodology for building the corpus, aiming for seamless integration of text analysis and code snippets. The textual content should articulate the problem-solving process, while the code snippets should perform precise computations. As shown in Figure 2, our data pipeline mainly includes three steps.

GPT Generation To create a solution in the desired format, we utilize instructions inspired by REACT (Yao et al., 2022) to ensure GPT recognizes when to employ an external tool-Python code interpreter. In addition to the REACT instruction in the prompt, we provide two manually crafted demonstration examples within the prompt for the language model to emulate. For further details on the prompt setup, please see Appendix A.1.1.

Given that the problems in GSM8K are relatively straightforward, we initially prompt both GPT-3.5 and GPT-4 to tackle each question with at maximum 5 code snippets using a temperature 0. For questions that remain unsolved after the initial two attempts, we address the potential requirement for more creative and diverse solutions by re-prompting GPT-4 with a temperature 0.6 for another two attempts. Therefore, we obtain at least one correct solution for 98.3% of the questions in GSM8K. In contrast, we exclusively utilize GPT-4 with maximum 8 code snippets allowed due to the substantially higher difficulty of the MATH questions for all the four attempts. Following this process, a mere 66.7% of the questions in MATH are provided with at least one correct solution.

Human Review Approximately 100 questions remain for which the answers generated by GPT do not align with the answers from the original GSM8K. Given the manageable number, we have conducted a manual review of the discrepancies and corrected any inaccuracies found in either the LLM-generated or the original answers. This ensures that each question within the GSM8K training set is associated with at least one correct solution in required format. For MATH dataset, we forgo manual solutions for all remaining 1,208 questions due to the significant burden it imposes. Instead, human verification is applied for the correction of easily rectifiable errors in the filed of final answers (see Appendix A.2 for details). Consequently, 83.9% of MATH questions are correctly solved, then combined with previously created GSM8K dataset into a unified dataset with size 26.9K.

Augmentation We reformatted the seed data from REACT to HTML and fine-tuned it on the pre-trained MATH LLM, Llemma-34B (Azerbayev et al., 2023). For reformatting details, see Appendix A.3. Using the fine-tuned model, we produced up to 100 samples per unsolved question at a temperature of 0.6, stopping after finding at most 4 correct solutions. This method yielded a coverage of 93.8% of the questions in MATH with correct solutions.

To further augment the scale and variety of questions in our dataset, we incorporate about 240K novel questions from MetaMath (Yu et al., 2023b), which are transformations of those found in the original GSM8K and MATH datasets. It allow us to gather a richer set of sampled solutions including both positive and negative examples. We anticipate that this strategy of data augmentation will significantly boost the model's performance and serve the training signals for outcome value model.

3 Fine-Tuning

To enhance a large language model's mathematical reasoning capabilities, we propose utilizing the foundational model Llemma (Azerbayev et al., 2023). There are two primary reasons for choosing Llemma. First, Llemma represents a continuation of the pre-training process initiated by Llama-2 (Touvron et al., 2023), extending its proficiency into both mathematical and coding domains, which aligns seamlessly with our requirements. Second, it has been demonstrated that neither Llama-2 nor Llemma exhibit excessive overfitting on the GSM8K or MATH dataset, as confirmed by (Wei et al., 2023).

3.1 Supervised Fine-Tuning with Full Parameters

The supervised fine-tuning closely mirrors the data generation step that serves to expand the coverage of the MATH dataset. During the SFT stage, we tune the entire set of parameters of the LLM using our specially curated dataset. For each given question $\mathbf{q}$ and its corresponding correct solution $\mathbf{s}^+$, we optimize the model by minimizing the following cross-entropy loss.

$$\min -\log p(\mathbf{s}^+|\mathbf{q})$$

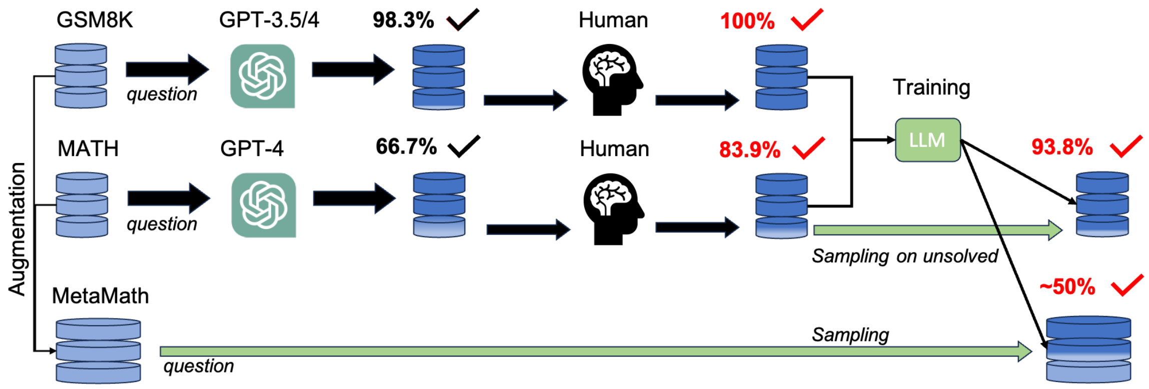


Figure 2: The data pipeline illustrates the process of data generation for the GSM8K and MATH datasets. We employ GPT to provide initial annotations, followed by human verification to fix easily rectifiable errors. For MATH dataset, an additional sampling strategy derived from a self-trained Large Language Model (LLM) is imposed.

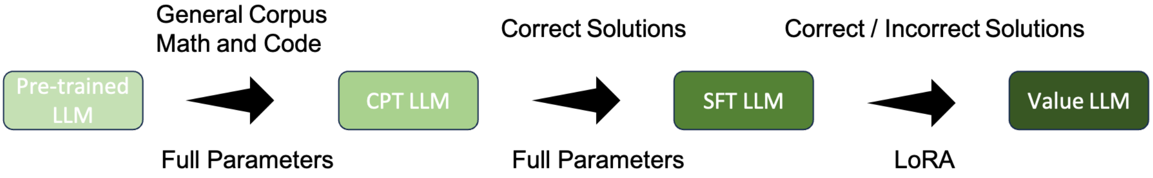


Figure 3: The training pipeline is divided into three distinct stages. First, we continue pre-training using a comprehensive corpus that encompasses both mathematical and coding domains, e.g., Llemma (Azerbayev et al., 2023). Second, it involves supervised fine-tuning in full parameters, utilizing our specially curated dataset. Finally, the model is further fine-tuned in multi-task setting.

3.2 Multi-Task Fine-Tuning with LoRA

During solution sampling, the LLM is capable of effortlessly creating both correct (positive) and incorrect (negative) samples $\mathbf{s}^-$. This duality enables us to train the LLM to discern the validity of a solution by predicting whether the final answer is accurate. To achieve this, we add a light-weighted binary classifier, alongside the existing softmax layer responsible for token prediction. We maintain a roughly equal ratio of positive to negative examples for balanced training. The overall loss follows the multi-task setting.

$$\min -\log p\left(\mathbf{s}^+, \mathbf{y}^+|\mathbf{q}\right) - \log p\left(\mathbf{y}^-|\mathbf{q}, \mathbf{s}^-\right)$$

Note that the first term can be factorized as two tasks $p(\mathbf{y}^+|\mathbf{q}, \mathbf{s}^+)p\left(\mathbf{s}^+|\mathbf{q}\right)$. The value prediction task corresponds to the sequence classification loss calculated for each token with the label as the correctness of the solution. Therefore, $\mathbf{y}$ is a vector whose length is equal to the number of tokens in the solution $\mathbf{s}$. Given the significantly larger data size required for training the Value LLM, we employ computationally efficient LoRA (Hu et al., 2021) during training.

In our multi-task setting, the Value LLM plays a dual role on generation solutions and evaluating them. The primary benefits of this feature is practicality, as it requires the deployment of only a single LLM for the entire inference process.

3.3 Outlier-Free OVM Inference

The solution-generating LLM can be paired with the Value LLM, which serves as an outcome value model (OVM). The Value LLM primarily serves to evaluate outcomes, that is, to estimate the likelihood of the final answer being correct. To improve the quality of solutions generated by the SFT LLM, one might consider re-ranking multiple solutions sampled from the SFT LLM. However, we suggest employing an outlier-free OVM selection algorithm to identify the best answer. Specifically, given K sampled solutions $\{\mathbf{s}_i\}_{i=1}^K$ resulting in k distinct final answers $\{\mathbf{a}_j\}_{j=1}^k$, the frequency of each answer is represented as n_j, such that $\sum_{j=1}^k n_j = K$. The optimal answer is selected according to the following criterion,

$$\kappa = \arg \max_{\{j|n_j>\delta_K\}} \max_{\mathbf{s}_i\in\mathbf{a}_j} \mathrm{OVM}(\mathbf{s}_i)$$

In our experiments, for $K = 20$, we set $\delta_K = 1$. This is because a small number of samples, such as

Data source	Generation method			Total solutions	# Code snippets	Trainset coverage
	GPT	Human	Self-train			
SFT (correct solutions)						
GSM8K	17,480	95	-	17,576	≤ 5	7,473 / 7,473
MATH	6,483	2,862	1,933	11,277	≤ 8	7,011 / 7,500
MetaMath	-	-	$\sim$55K	$\sim$55K	≤ 8	-
OVM (correct / incorrect solutions)						
MetaMath	-	-	$\sim$300K	$\sim$300K	≤ 8	-

Table 1: Data statistics

20 in our example, might lead to a situation where a random sample yields an anomalously high outcome prediction, making it crucial to exclude outlier solutions. In the rare case that all K sampled solutions are unique, we simply choose the solution with the highest predicted outcome value.

In addition, this generation ability of value LLM, maintained along with the prediction of token-level values, allows for straightforward modifications to the decoding algorithm used in the transformer decoder implementation. For instance, the beam search mechanism could combine the original log-likelihood with the predicted value. We will explore this potential direction in future work.

4 Experiments

4.1 Dataset Recap

We present the statistics for our positive examples used in supervised fine-tuning in Table 1. The seed data, derived from GSM8K and MATH datasets, culminates in a collection of 26.9K solutions. For the augmentation data obtained from MetaMath, which encompasses 240K new questions, we employed the augmentation method to sample one or two solutions for each question and randomly select approximately 55K question-correct solution pairs. In total, we have gathered 300K examples, both positive and negative, maintaining an approximately balanced ratio of labels.

The in-domain test sets come from the original GSM8K and MATH datasets. We also conduct evaluations on two out-of-domain (OOD) test sets: the open-source OCWCourses dataset (Lewkowycz et al., 2022) and our proprietary GaoKao2023-Math-En dataset. OCWCourses comprises a collection of 272 STEM problems aimed at the undergraduate level, requiring multi-step reasoning for most questions. The GaoKao2023-Math-En dataset consists of 385 mathematics problems from the 2023 Chinese higher education entrance examination (professionally translated into English), the 2023 American Mathematics Competitions, and the 2023 American College Testing.

4.2 Implementation Details

We train the Llemma series (Azerbayev et al., 2023) through fine-tuning with our curated corpus, resulting in the development of our SFT LLM series. During this optimization phase, we generally employed a learning rate of 5e-5, with the exception of the 7B and 34B models, for which we reduced the rate to 4e-5. We set the global batch size at 512 and used a linear learning rate scheduler that included a warm-up phase constituting 3% of the total training duration, spread over 3 epochs. Training for all models was launched with the accelerate[2] in DeepSpeed ZeRO Stage2 (Rajbhandari et al., 2021) and Flash-Attention 2 mechanism (Dao, 2023). When fine-tuning the value LLM with LoRA, we configure the hyper-parameters with a rank of 4096 and an alpha of 2048 for the attention parameters W_q and W_v. In the context of the Llama-2-7B architecture, 2B model parameters are trainable. We employ a learning rate of 5e-5, which is progressively adjusted using a cosine decay scheduler. We use 8 or 16 A100-80G GPUs for training 7B and 34B models. We also implemented a new math answer evaluation toolkit to compare the ground truth with the LLM predictions to determine if they are equivalent expressions.

Baselines We conducted comparisons with renowned proprietary and open-source LLMs such as GPT (OpenAI, 2023), Claude (Anthropic, 2023), PaLM (Anil et al., 2023), Minerva (Lewkowycz et al., 2022), Gemini (Team et al., 2023), Llama-2 (Touvron et al., 2023), CodeLlama (Roziere et al., 2023), Qwen (Bai et al., 2023), and DeepSeek (DeepSeek, 2023). We also have reported results from a variety of open-source models, most notably Llama-2, along with several SFT models derived from Llama-2, including RFT (Yuan et al., 2023), WizardMath (Luo et al., 2023), MathCoder (Wang et al., 2023a), MAmmoTH (Yue et al., 2023) and ToRA (Gou et al., 2023).

4.3 Main Results

SFT Model Table 2 demonstrates the performance of greed decoding. Our 7B model across four datasets encompasses both in-domain and out-of-domain problems when comparing with other open-sourced LLMs with similar model size and data size. In contrast, for more complex problems in the MATH dataset, or even for challenging out-

---

[2] `https://github.com/huggingface/accelerate`

Model	Size	Tool	Zero Shot	In-domain		Out-of-domain	
				GSM8K	MATH	OCW	GK2023[*]
Proprietary Models							
GPT-4	-	✗	✗	92.0	42.5	-	-
GPT-4-Code	-	✓	✗	92.9	69.7	-	-
ChatGPT	-	✗	✗	80.8	35.5	-	-
ChatGPT(PAL)	-	✓	✗	78.6	38.7	-	-
Claude-2	-	✗	✗	85.2	32.5	-	-
PaLM-2	540B	✗	✗	80.7	34.3	-	-
Minerva	540B	✗	✗	58.8	33.6	17.6	-
Gemini Ultra maj@32[†]	-	✗	✗	94.4	53.2	-	-
Open-Source Models							
Llama-2 SFT	7B	✗	✓	41.3	7.2	-	-
Llama-2 RFT	7B	✗	✓	51.2	-	-	-
Llemma	7B	✗	✗	36.4	18.0	7.7	-
Llemma(PAL)	7B	✓	✗	40.1	21.5	-	-
Qwen	7B	✗	✗	51.7	11.6	-	-
WizardMath	7B	✗	✓	54.9	10.7	-	-
DeepSeek-Coder	6.7B	✓	✗	43.2	19.2	-	-
MathCoder	7B	✓	✓	67.8	30.2	-	-
MAmmoTH-Coder	7B	✓	✗	59.4	33.4	11.0	15.3
ToRA	7B	✓	✓	68.8	40.1	2.6	19.5
ToRA-Code	7B	✓	✓	72.6	44.6	4.8	23.9
MARIO	7B	✓	✓	70.1	47.0	**21.7**	**38.2**
MARIO-OVM-7B[§]	7B	✓	✓	**74.5**	**48.3**	21.0	34.8
CodeLlama	34B	✗	✗	29.6	12.2	7.0	-
CodeLlama(PAL)	34B	✓	✗	53.3	23.9	-	-
Llemma	34B	✗	✗	51.5	25.0	11.8	-
Llemma(PAL)	34B	✓	✗	62.6	27.1	-	-
DeepSeek-Coder	33B	✓	✗	60.7	29.1	-	-
MathCoder	34B	✓	✓	**81.7**	45.2	-	-
MAmmoTH-Coder	34B	✓	✗	72.7	43.6	14.0	25.2
ToRA-Code	34B	✓	✓	80.7	50.8	5.5	31.7
MARIO[‡]	34B	✓	✓	78.8	**53.5**	**30.2**	**42.6**
DeepSeek-Chat	67B	✗	✗	84.1	32.6	-	-
WizardMath	70B	✗	✓	81.6	22.7	-	-
MathCoder	70B	✓	✓	83.9	45.1	-	-
MAmmoTH	70B	✓	✗	76.9	41.8	11.8	24.7
ToRA	70B	✓	✓	84.3	49.7	9.6	30.9
Qwen	72B	✗	✗	78.9	35.2	-	-

Table 2: Results on different datasets. The best results of open-source models are bold. [*]GK2023-ME represents Gaokao-2023-Math-En dataset. [†]maj@K means majority voting over K samples. [§]The MARIO-OVM-7B here is simply used as an SFT LLM to generate one single solution.

Model	PAL (Gao et al., 2023)	DeepSeek-Coder	MAmmoTH	ToRA-Code	MARIO	MARIO-OVM
Size	175B*	6.7B	7B	7B	7B	7B
Accuracy	61.2	40.3	56.5	56.0	50.0	53.2

Table 3: Results on GSM-Hard (Gao et al., 2023). *PAL is based on code-davinci-002.

of-domain problems, our 34B model consistently outperforms others. Fewer training data may be one reason, but the main reason should be its capability to perform text analysis, which breaks down problems into manageable code snippets, thus enhancing its problem-solving effectiveness. This is verified by the similar pattern observed with our 7B model that was trained on an 82K dataset, which is in line with past state-of-the-art (SOTA) methods. So we conclude that such a model achieves superior results on more complex problems, likely because these problems demand more than simple logic and a few arithmetic steps—scenarios where models with a code-centric approach typically have an edge.

OVM Model The experimental findings of our outlier-free OVM selection algorithm are displayed in Table 4, where we contrast our approach with the majority voting algorithm. Our findings indicate that the gain of majority voting by our approach is more significant than ToRA, because text generations allow more creative ideas for problem solving than code only solution. In addition, our outlier-free OVM inference can further push up the performance of majority voting. In Table 4, we also present a comprehensive results showcasing the OVM's performance when it takes on both the roles of solution generation and outcome evaluation. The OVM demonstrates a comparable proficiency in generating solutions; however, it exhibits a slightly reduced effectiveness on out-of-domain datasets. This outcome is to be expected, given that our OVM has been continually fine-tuned on the MetaMath questions, originating from the GSM8K and MATH datasets.

Inference method	GSM8K	MATH	OCWCourses	GK2023
ToRA-Code-7B	72.6	44.6	4.8	23.9
+maj@50	76.8 +4.2	52.5 +7.9	-	-
MARIO-7B	70.1	47.0	21.7	38.2
+maj@20	80.5 +10.4	56.7 +9.7	25.4 +3.7	41.6 +3.4
+OVM-7B@20	82.9 +12.8	59.1 +12.1	**28.3** +6.6	**45.2** +7.0
MARIO-OVM-7B	74.5	48.3	21.0	34.8
+maj@20	**83.8** +9.3	59.7 +11.4	22.1 +1.1	43.6 +8.8
+OVM-7B@20	83.6 +9.1	**60.6** +12.3	25.4 +4.4	42.9 +8.1

Table 4: Results on OVM-7B. +x indicates the increased accuracy compared with the greedy decoding.

4.4 Ablation Studies

We perform the first ablation study to examines the impact of each data source by incrementally adding more training examples, with the primary findings detailed in Table 5. Overall, the advancements in MATH are more pronounced. We ascribe this trend to three main factors. First, the GSM8K dataset, synthesized by GPT, encompasses 98.3% of the questions, in contrast to the MATH dataset's 66.7% coverage. Secondly, the selection criterion for the MATH dataset hinges on an exact match between GPT's generated final answer and the dataset's provided answer, which could lead the fine-tuned model to overfit specific questions that have straightforward answers. Human intervention has the potential to enhance the variability of the answers. Lastly, the teacher model's generated solutions concentrate more heavily on the MATH dataset. This is due to the fact that we have extracted a greater number of multi-step reasoning solutions according to MetaMath questions, which are likely better suited to the difficulty of the MATH dataset.

The second ablation study aims to investigate the impact of the foundational math LLM and the data formatting for SFT. DeepSeek-MATH-7B (Shao et al., 2024), is a specialized math-focused LLM developed through continual pre-training on the Deep-Seek-Code-7B model, which benefits from a more extensive math pre-training corpus than Llemma-7B and purposely omits any content that may relate to GSM8K and MATH datasets. Consequently, DeepSeek-MATH-7B is supposed to outperform Llemma-7B. When applying SFT on a large scale code-centric SFT dataset, DeepSeek-MATH-7B can achieve the SOTA performance as 7B LLM. The result presented in the second row of Table 6 shows the results of fine-tuning DeepSeek-MATH-7B with our dataset. Despite being only 1/30th the size of their used dataset, our hybrid format demonstrates greater data efficiency. The result presented in the last row of Table 6 suggest that SFT from a superior continue pre-trained (CPT) model enhances math reasoning capabilities.

Data used	# trainset	GSM8K	MATH
GPT	23.9K	66.3	40.2
+Human	26.9K	67.1 +0.8	43.5 +3.3
+MATH Aug	28.8K	67.4 +1.1	44.4 +4.2
+MetaMath Aug	82K	70.1 +3.8	47.0 +6.8

Table 5: Ablation study of CPT Model and Data format. #The result is sourced from (Shao et al., 2024).

CPT Model	data size	data format	GSM8K	MATH.
DeepSeek-Math-7B#	776K	code	83.7	57.4
DeepSeek-Math-7B	28.8K	text+code	78.4	56.1
Llemma-Math-7B	28.8K	text+code	67.4	44.4

Table 6: Ablation study of data usage on 7B model.

4.5 Why GSM-Hard is not a good testset for MATH LLM?

The GSM-Hard dataset, introduced by Gao et al. (2023), is akin to the original GSM8K test set, with the sole distinction being the alteration of numbers in the original problem statements. However, as illustrated in Figure 1, these modifications to the numbers do not always align with the common sense of real physical world, *e.g.*, ages cannot be negative, and the number of people cannot be fractional. Methods following the PoT paradigm tend to generate code without verifying the sensibility of their output, scarifying this ability of LLMs. In contrast, our approach incorporates a textual analysis that ensures the results derived from code execution are consistent with the constraints of the physical world. As a result, our LLM will opt not to produce an illogical final answer or to arbitrarily round fractions, even if the so-called correct answer has been computed from the code execution in our approach. This accounts for the lower accuracy of our method on this dataset, shown in Table 3. In addition, we found some solutions in the GSM-Hard remain the same as the original GSM8K, even the numbers have changed. Some representative examples are provided in Appendix A.4, which compares the solutions between ground truth provided by GSM-Hard, ToRA, and our approach. In summary, we suggest not the use GSM-Hard dataset unless the mentioned errors have been fixed.

Reformatting In Appendix A.3, we quantitatively verify the intuition by reformatting the REACT data to HTML data.

5 Related Works

Mathematical reasoning attracts more attentions because of the emergence of LLMs. Recent works (Wei et al., 2022; Kojima et al., 2023; Wang et al., 2023b; DeepSeek, 2023) on mathematical reasoning have made impressive progress empowered by LLMs. Yet exact calculations and symbolic manipulations within the reasoning process remain challenging. Some works have explored tools including calculators (Cobbe et al., 2021b; Shao et al., 2022) and code interpreters (Gao et al., 2023) to address the limitations. Further research (Wang et al., 2023a; Yue et al., 2023; Gou et al., 2023) attempt to combine tool-use and textual reasoning process to leverage the strengths of both.

Knowledge distillation (Hinton et al., 2015; Gou et al., 2021) is a commonly used approach to promote student models by transferring knowledge from teacher models to them. Utilizing teacher LLM to construct reasoning samples for student model to fine-tune proved to be effective practice of knowledge distillation(Fu et al., 2023; Ho et al., 2023). Our corpus construction includes knowledge distillation of this kind on MATH with more samplings from 34B SFT LLM.

Verification in mathematical reasoning plays a crucial role in ensuring inference performance by allowing auto-regressive models to correct already-made errors. It has been proved that LLMs can self-verify (Anonymous, 2023; Weng et al., 2023; Xie et al., 2023) and self-refine (Madaan et al., 2023) by designed prompting. A specifically trained verifier can also play a similar role by intervening the decoding process (Cobbe et al., 2021b; Khalifa et al., 2023; Yu et al., 2023a). In this paper, we use multi-task fine-tuning which is similar to the training of a simple outcome supervision model.

6 Conclusion

This paper introduces a reproducible pipeline that covers both the construction of a math-specific dataset and the fine-tuning of a large language model (LLM). Our approach demonstrates that integrating text analyses with code snippets enhances the model's capabilities for common sense reasoning and precise computation in mathematical reasoning tasks. Moreover, our fine-tuning method enhances model performance by incorporating a verifier model that requires only a negligible number of additional parameters. To the best of our knowledge, our approach sets a new state-of-the-art benchmark for LLMs with a size around 7 billion parameters on the MATH datasets, and it exhibits notable generalization ability on challenging out-of-domain math datasets.

7 Limitations

The primary limitation of this study lies in the expenses associated with generating data. To begin with, producing raw data in the REACT format necessitates using the GPT API, *e.g.*, generating a single solution for questions in the GSM8K and MATH datasets costs \$0.01 and \$0.025 respectively when utilizing GPT-4 in non-stream mode. Additionally, human intervention for error correction demands approximately 80 working hours of labor to rectify solutions. Scaling up this dataset would therefore entail a significant increase in both financial outlay and manpower.

Furthermore, our initial experimentation encountered several mistakes in the details of both the data and training pipelines, which resulted in additional, unnecessary expenditures. As a result, we have decided to release the source code for our data and training pipelines. We hope that by doing so, other researchers in this field can draw on our experience and avoid similar costly errors, thereby reducing their expenses.

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Anonymous. 2023. Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. In *Submitted to The Twelfth International Conference on Learning Representations*. Under review.

Anthropic. 2023. Model card and evaluations for claude models.

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. *arXiv preprint arXiv:2310.10631*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report.

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

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. 2021a. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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

Tri Dao. 2023. FlashAttention-2: Faster attention with better parallelism and work partitioning.

DeepSeek. 2023. Deepseek coder: Let the code write itself. `https://github.com/deepseek-ai/DeepSeek-Coder`.

Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. 2023. Specializing smaller language models towards multi-step reasoning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10421–10430. PMLR.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving.

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 Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2023. Large language models are reasoning teachers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14852–14882, Toronto, Canada. Association for Computational Linguistics.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Muhammad Khalifa, Lajanugen Logeswaran, Moontae Lee, Honglak Lee, and Lu Wang. 2023. GRACE: Discriminator-guided chain-of-thought reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 15299–15328, Singapore. Association for Computational Linguistics.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857.

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 preprint arXiv:2305.20050*.

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*.

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

R OpenAI. 2023. Gpt-4 technical report. *arXiv*, pages 2303–08774.

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA. Association for Computing Machinery.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Zhihong Shao, Fei Huang, and Minlie Huang. 2022. Chaining simultaneous thoughts for numerical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2533–2547, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Y Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023a. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

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. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Tianwen Wei, Liang Zhao, Lichang Zhang, Bo Zhu, Lijie Wang, Haihua Yang, Biye Li, Cheng Cheng, Weiwei Lü, Rui Hu, et al. 2023. Skywork: A more open bilingual foundation model. *arXiv preprint arXiv:2310.19341*.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575, Singapore. Association for Computational Linguistics.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. 2023. Decomposition enhances reasoning via self-evaluation guided decoding.

Zhen Yang, Ming Ding, Qingsong Lv, Zhihuan Jiang, Zehai He, Yuyi Guo, Jinfeng Bai, and Jie Tang. 2023. Gpt can solve mathematical problems without a calculator. *arXiv preprint arXiv:2309.03241*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Fei Yu, Anningzhe Gao, and Benyou Wang. 2023a. Outcome-supervised verifiers for planning in mathematical reasoning. *arXiv preprint arXiv:2311.09724*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023b. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

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. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, et al. 2023. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. *arXiv preprint arXiv:2308.07921*.

A Appendix

A.1 Introduce to our submitted code repository

A.1.1 Reproducing Data Generation

To generate the solution of a provided question, please refer to the following example script in our submitted code repository.

```
python gpt_react.py \
--verbose \
--dataset math \
-g gpt-4-1106-preview \
-q "Find all the roots of x^4 + 4 = 0."
```

A.1.2 Fine-tuning

Our training is mostly performed on LLaMA-Factory[3] code base. Please refer to that repository for more details.

A.1.3 Inference

Single question inference.

```
python react.py -c /path/to/checkpoint_dir -q "Compute tan(45)." --verbose
```

Large scale inference with vllm[4].

```
python batch_react.py -c /path/to/checkpoint_dir -q /path/to/question_file
```

Question file should be in `jsonl` format, where each line is a json string. The json string should at least include a key value pair for question.

A.1.4 Evaluation Toolkit

In order to evaluate the model prediction, it requires our implemented toolkit that is located in folder `math_evaluation`.

```
python eval.py -q /path/to/question_file
```

Question file should be in `jsonl` format, where each line is a json string at least containing "pred" and "answer" keys for prediction and ground truth, respectively.

A.1.5 Prompts

The REACT Instruction is as follows. For the demonstration examples for GSM8K and MATH, please refer to the file `prompts.py` in our submitted code repository.

```
You are a powerful agent with broad math knowledge and great Python programming
skills. Answer the math questions as best you can. You have access to the following
tool:

python_interpreter: A Python shell to execute python code snippet.

When solving math problem, you should think step by step, where each step includes 4
mini-steps Thought/Action/Action Input/Observation. Note that if some step requires
accurate calculation (including but not limited to symbolic simplification,
derivation, numerical calculation, solving equations or inequalities), you should
write Python code and execute it to obtain result.
The following is the required template.

Question: the input question
```

---

[3] https://github.com/hiyouga/LLaMA-Factory/)

[4] https://github.com/vllm-project/vllm

~~~
Thought: the text analysis, and list the math equations if necessary

Action: the action to take, should be python_interpreter, or None

Action Input: the Python Code in markdown format (if Action is None, it is None), e.g.,
```python
import math
theta = math.pi / 4
some_var = math.cos(theta)
print(some_var)
```

Observation: the result of the action

... (this Thought/Action/Action Input/Observation can repeat N times)

Thought: the final analysis of the answer

Final Answer: the concise answer without verbose context

The following are 2 demonstrations examples.

{examples}

Now! It's your turn.

Question: {question}

Thought:
~~~

A.2 Human Review for MATH

Our own researchers are responsible for the human review. We mainly fix the following issues in the field of final answer.

- **Redundant text** Even in the prompt we have indicated the field of final answer should only include the math expression of final answer without other text. It is inevitable that a full sentence will be generated in this field. Therefore, we will remove the redundant text, *e.g.*,

  ```
  Final Answer: John spent 25 dollars in total. => Final Answer: 25
  ```

- **Equivalent expression** Because the text analysis of LLM is based on python code snippets and the corresponding execution results, the generated final answers prefer the ‘sympy’ format, which differs from the ‘latex’ format provided in the MATH dataset. However, they are sometimes equivalent. In this case, we should consider the generated solution as correct, *e.g.*,

  ```
  \\frac{8 - 7x}{6} = 4/3 - 7x/6
  \\begin{pmatrix} 1 & 2 \\\\ 3 & 4 \\end{pmatrix} = Matrix([[1, 2], [3, 4]])
  ```

  This also motivates us to develop the math evaluation toolkit.

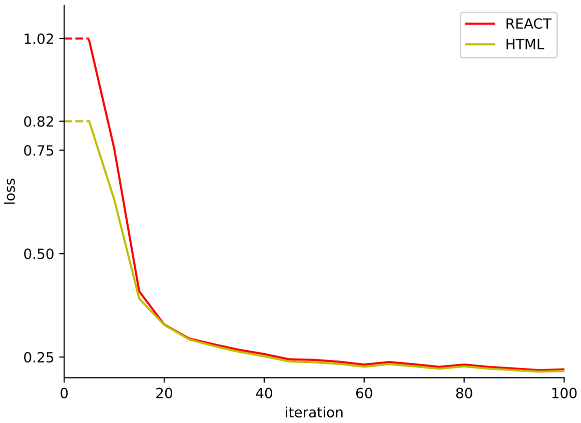


Figure 4: Train loss of different formats within first 100 iterations when fine-tuning on Llemma-34B.

A.3 Re-formatting

The data generation with REACT is depicted in A.1.5. However, the data we obtained using the REACT template's keywords was not used directly. Instead, we converted the REACT data into an HTML-like format, using `<p></p>` to encapsulate text analyses and `<code></code>` to encapsulate code snippets, as demonstrated in the Figure 5. We observed that employing REACT instructions typically yielded higher quality output from GPT models. Yet, when it comes to fine-tuning a pre-trained Large Language Model (LLM), utilizing an HTML-like format for the initial iteration results in a training loss that is, on average, approximately 20% lower. A thorough comparison is available in Section 4. Our hypothesis is that the HTML-like format may bridge the gap between the pre-training corpus and the fine-tuning corpus, leading to improved initialization performance.

The standard template for REACT examplified in previous section employs a key-value pair format represented as strings, with each step comprising elements like "`Thought:text analysis`", "`Action: tool name`", "`Action Input: code snippet`", and "`Observation: execution output`". REACT (Yao et al., 2022) is an excellent approach to empower the LLM the capability of using external tool. However, our analysis of the log-likelihood for REACT examples suggests that such data formats are rare in the corpora used for pre-training LLMs. In contrast, when converting REACT data into an HTML format, using tags such as "`<p>text analysis</p>`" and "`<code>code snippet</code>`", we noted a substantial decrease in log-likelihood. In our data pipeline of Figure 2, when training the Llemma-34B on the 26.9K dataset with different formats, this reduction was evident from the initial loss observed in the first iteration.

As Figure 4 demonstrates, the loss for fine-tuning with HTML-formatted data is 20% lower than when using the original REACT format. However, upon evaluating the performance on the test sets after 3 epochs of SFT, we observed only a marginal improvement (1%) on MATH test set. Given the substantially reduced initial loss, we hypothesize that careful hyper-parameter tuning could potentially enhance the performance of data formatted in HTML. We intend to explore this possibility in future work.

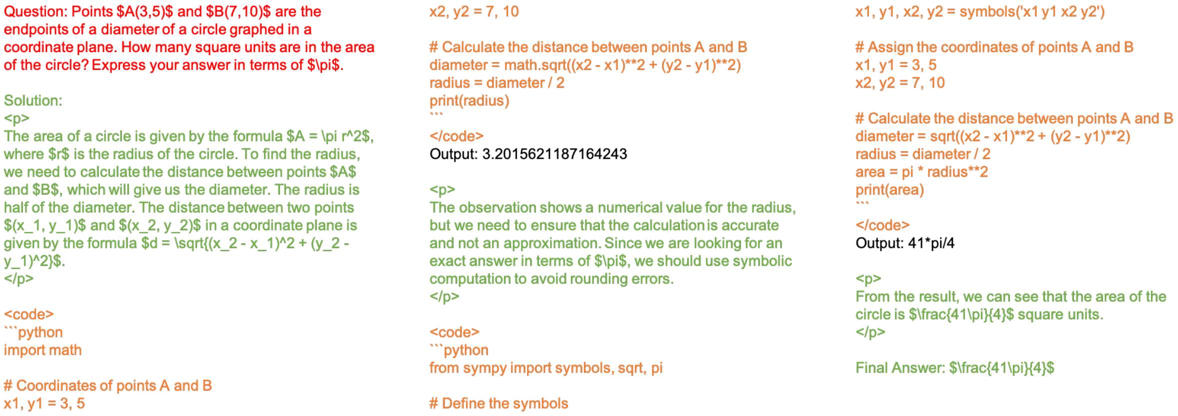
A.4 Case Study on GSM-Hard

In this case study, we present three typical examples demonstrating why many of our model's final answers are unjustifiably deemed incorrect.

A.4.1 Incorrect or unreasonable answer

In this example, the PoT solution provided by GSM-Hard remains the same as the initial answer in the GSM8K test set, yet it fails to align with the question as it does not incorporate the number 780786 in its program. Meanwhile, the ToRA solution neglects the crucial detail that the quantity of Pokemon must be an integer. In contrast, our model had previously generated the fractional answer, but its internal self-verification process detected inconsistencies, leading it to ultimately output 'None.'

```
Question: Alex, Stan, and Adelwolfe are trying to catch them all, Pokemon that is.
```



~~~~
Question: Points $A(3,5)$ and $B(7,10)$ are the
endpoints of a diameter of a circle graphed in a
coordinate plane. How many square units are in the area
of the circle? Express your answer in terms of $\pi$.

Solution:
<p>
The area of a circle is given by the formula $A = \pi r^2$,
where $r$ is the radius of the circle. To find the radius,
we need to calculate the distance between points $A$
and $B$, which will give us the diameter. The radius is
half of the diameter. The distance between two points
$(x_1, y_1)$ and $(x_2, y_2)$ in a coordinate plane is
given by the formula $d = \sqrt{(x_2 - x_1)^2 + (y_2 -
y_1)^2}$.
</p>

<code>
```python
import math

# Coordinates of points A and B
x1, y1 = 3, 5
x2, y2 = 7, 10

# Calculate the distance between points A and B
diameter = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
radius = diameter / 2
print(radius)
```
</code>
Output: 3.2015621187164243

<p>
The observation shows a numerical value for the radius,
but we need to ensure that the calculation is accurate
and not an approximation. Since we are looking for an
exact answer in terms of $\pi$, we should use symbolic
computation to avoid rounding errors.
</p>

<code>
```python
from sympy import symbols, sqrt, pi

# Define the symbols
x1, y1, x2, y2 = symbols('x1 y1 x2 y2')

# Assign the coordinates of points A and B
x1, y1 = 3, 5
x2, y2 = 7, 10

# Calculate the distance between points A and B
diameter = sqrt((x2 - x1)**2 + (y2 - y1)**2)
radius = diameter / 2
area = pi * radius**2
print(area)
```
</code>
Output: 41*pi/4

<p>
From the result, we can see that the area of the
circle is $\frac{41\pi}{4}$ square units.
</p>

Final Answer: $\frac{41\pi}{4}$
~~~~

Figure 5: A re-formatting example includes a combined display of text analyses and accompanying code snippets. Please note that the display represents a restructured format resembling HTML, from the original REACT format generated.

~~~~
Together they have caught 780786 Pokemon. Alex has caught 5 more than Stan, and Stan
has caught 13 less than 4 times as many as Adelwolfe has caught. How many Pokemon
has Stan caught?

PoT Solution from GSM-Hard:

def solution():
    stan = -1  # we don't know how many pokemon stan has caught, so we set it to -1
    total = 780786
    alex = stan + 5
    adelwolfe = (stan + 13) / 4
    stan = total - alex - adelwolfe
    # write values for alex and adelwolfe in terms of stan
    stan = total - (stan + 5) - ((stan + 13) / 4)
    # simplify
    stan + 5 + (stan + 13) / 4 + stan == 339
    4 * stan + 20 + stan + 13 + 4 * stan == 4 * total
    9 * stan + 33 == 4 * total
    9 * stan == 4 * 339 - 33
    # so we get
    stan = (4 * 339 - 33) / 9
    result = stan
    return result
Answer: 147.0


PoT Solution from ToRA:

```python
from sympy import symbols, Eq, solve

def stan_pokemon():
    # Define the symbols
    a, s, ad = symbols('a s ad')
~~~~

```
    # Define the equations
    eq1 = Eq(a + s + ad, 780786)
    eq2 = Eq(a, s + 5)
    eq3 = Eq(s, 4*ad - 13)

    # Solve the system of equations
    solution = solve((eq1, eq2, eq3), (a, s, ad))

    return solution[s]

result = stan_pokemon()
print(result)
```
```output
1041037/3
```
Stan has caught $\\boxed{\\frac{1041037}{3}}$ Pokemon.

Our Solution:

<p>
Let's assume that Adelwolfe has caught x Pokemon. According to the given information, Stan has caught 13 less than 4 times as many Pokemon as Adelwolfe, so Stan has caught (4x - 13) Pokemon. Alex has caught 5 more than Stan, so Alex has caught (4x - 13) + 5 = 4x - 8 Pokemon. Together, they have caught 780786 Pokemon, so we can set up the equation:

x + (4x - 13) + (4x - 8) = 780786

Now, I will solve this equation to find the value of x, which represents the number of Pokemon Adelwolfe has caught. Once I have x, I can calculate the number of Pokemon Stan has caught by substituting x into the expression (4x - 13).
</p>

<code>
```python
from sympy import symbols, Eq, solve

x = symbols('x')
equation = Eq(x + (4*x - 13) + (4*x - 8), 780786)
solution = solve(equation, x)
stan_pokemon = 4*solution[0] - 13
stan_pokemon
```
</code>
Output: 1041037/3

<p>
The solution to the equation is x = 1041037/3. However, since x represents the number of Pokemon Adelwolfe has caught, it must be an integer. I need to verify that the solution for x is indeed an integer.
</p>

~~~
<code>
```python
is_integer = solution[0].is_integer
is_integer
~~~
</code>
Output: False

Final Answer: None
```

### A.4.2 Incorrect or unreasonable intermediate step

In this case, the PoT solution offered by GSM-Hard calculates an intermediate variable cows as -3016191, which is not feasible. Even though the final answer representing the number of chickens is positive, the question is deemed invalid because the quantity of cows cannot be a negative value. Regarding the ToRA solution, it arrives at the same answer as GSM-Hard despite generating an incorrect program output. Our model's program produces identical intermediate results to what is considered the correct answer, yet upon textual analysis, it recognizes that the count of animals must not be negative. Consequently, the model persists in self-verification up to a predetermined maximum number of steps and ultimately yields 'None' as the output.

```
Question: Farmer Brown has 3016226 animals on his farm, all either chickens or cows.
They have a total of 70 legs, all together. How many of the animals are chickens?

PoT Solution from GSM-Hard:

def solution():
    total_animals = 3016226
    total_legs = 70
    chicken_legs = 2
    cow_legs = 4
    cows = (total_legs - (total_animals * chicken_legs)) / (cow_legs - chicken_legs)
    chickens = total_animals - cows
    result = chickens
    return result
Answer: 6032417.0

PoT Solution from ToRA:

```python
def chicken_count():
    total_animals = 3016226
    total_legs = 70
    # Let\'s denote the number of chickens as x. Then the number of cows is 3016226 - x.
    # The total number of legs is the sum of the legs of the chickens and the cows.
    # So we have the equation: 2x + 4*3016226 = 70
    # Solving this equation for x gives us the number of chickens.
    chickens = (total_legs - 4 * total_animals) / 2
    return int(chickens)

chickens = chicken_count()
print(chickens)
```

~~~~
```
```output
-6032417
```
The number of chickens is $\\boxed{6032417}$.


Our solution:

<p>
To solve this problem, we need to set up an equation based on the information given.
We know that each chicken has 2 legs and each cow has 4 legs. Let's denote the number
of chickens as $c$ and the number of cows as $w$. The total number of legs is the sum
of the legs of all the chickens and all the cows, which can be expressed as $2c + 4w = 70$.
We also know that the total number of animals is $c + w = 3016226$. We can use these
two equations to solve for $c$ and $w$.
</p>

<code>
```python
from sympy import symbols, Eq, solve

# Define the symbols
c, w = symbols('c w')

# Equation for the total number of legs
legs_eq = Eq(2*c + 4*w, 70)

# Equation for the total number of animals
animals_eq = Eq(c + w, 3016226)

# Solve the system of equations
solution = solve((legs_eq, animals_eq), (c, w))
print(solution)
```
</code>
Output: {c: 6032417, w: -3016191}

<p>
The solution to the system of equations indicates that there are 6032417 cows and
-3016191 chickens, which is not possible since the number of animals cannot be negative.
This suggests there might be an error in the interpretation of the problem or in the
equations set up. The negative number of chickens indicates that the equation for
the total number of legs is incorrect.
</p>

...(skip many verification steps)

Final Answer: None
~~~~

A.4.3 Ambiguous question

In this instance, the solutions provided by GSM-Hard and ToRA are more precise, yielding the exact figure of 377712.375. In fact, our model also generates this precise number during an intermediate step. However, the question's phrasing regarding "how many hours" is somewhat ambiguous, leading our model to round the answer to the nearest whole number to better align with the question's apparent emphasis on the duration in hours.

~~~
Question: Lori wants to buy a $320.00 pair of shoes and a matching belt that is
$3021379. Her part-time job pays her $8.00 an hour. How many hours will she have
to work before she can make her purchase?

PoT Solution from GSM-Hard:

def solution():
    shoes_cost = 320
    belt_cost = 3021379
    hourly_wage = 8
    total_cost = shoes_cost + belt_cost
    hours_worked = total_cost / hourly_wage
    result = hours_worked
    return result
Answer: 377712.375


PoT Solution from ToRA:

```python
def hours_to_work():
    shoe_cost = 320.00
    belt_cost = 3021379.00
    total_cost = shoe_cost + belt_cost
    hourly_wage = 8.00
    hours_needed = total_cost / hourly_wage
    return hours_needed

hours_needed = hours_to_work()
print(hours_needed)
```
```output
377712.375
```
Lori will have to work $\\boxed{377712.375}$ hours before she can make her purchase.


Our Solutions:

<p>
To find out how many hours Lori needs to work, we need to calculate the total cost
of the shoes and the belt, and then divide that by her hourly wage.
</p>

<code>
~~~

```python
shoes_cost = 320.00
belt_cost = 3021379
total_cost = shoes_cost + belt_cost
hourly_wage = 8.00
hours_needed = total_cost / hourly_wage\nprint(hours_needed)
```
</code>
Output: 377712.375

<p>
The calculation shows that Lori needs to work approximately 377712.375 hours to make her purchase. Since she can't work a fraction of an hour, she will need to round up to the nearest whole hour.
</p>

<p>
Since Lori can't work a fraction of an hour, and the calculation shows that she needs to work approximately 377712.375 hours, she will have to work 377713 hours to make her purchase.
</p>

Final Answer: 377713