MuMath-Code: Combining Tool-Use Large Language Models with Multi-perspective Data Augmentation for Mathematical Reasoning

Anonymous ACL submission

Abstract

The tool-use Large Language Models (LLMs) that integrate with external Python interpreters have significantly enhanced mathematical reasoning capabilities for open-source LLMs, while tool-free methods chose another track: augmenting math reasoning data. However, a great method to integrate the above two research paths and combine their advantages remains to be explored. In this work, we firstly include new math questions via **mu**lti-perspective data augmenting methods and then synthesize **code**-nested solutions to them. The open LLMs (i.e., Llama-2) are finetuned on the augmented dataset to get the resulting models, **MuMath-Code** (μ-Math-Code). During the inference phase, our MuMath-Code generates code and interacts with the external python interpreter to get the execution results. Therefore, MuMath-Code leverages the advantages of both the external tool and data augmentation. To fully leverage the advantages of our augmented data, we propose a two-stage training strategy: In Stage-1, we finetune Llama-2 on pure CoT data to get an intermediate model, which then is trained on the code-nested data in Stage-2 to get the resulting MuMath-Code. Our MuMath-Code-7B achieves 83.8 on GSM8K and 52.4 on MATH, while MuMath-Code-70B model achieves new state-of-the-art performance among open methods—achieving 90.7% on GSM8K and 55.1% on MATH. Extensive experiments validate the combination of tool use and data augmentation, as well as our two-stage training strategy. We release the proposed dataset along with the associated code for public use.

1 Introduction

In Natural Language Processing (NLP), Large Language Models (LLMs) (Radford et al., 2019; Brown et al., 2020; Raffel et al., 2023) especially the proprietary ones such as GPT-4 (OpenAI, 2023a) and Claud-3 (Anthropic, 2024) have demonstrated superiority in a variety of tasks, e.g., text classification (Wang et al., 2018; Devlin et al., 2019; Min et al., 2022; Jiang et al., 2023b), automated coding (Chen et al., 2021; Luo et al., 2023b), instructions following (Longpre et al., 2023), and math problem solving (Chowdhery et al., 2022; Lewkowycz et al., 2022; Anil et al., 2023; Fu et al., 2023a). Among these tasks, the capability to handle math problems stands as a typical and critical criterion for the evaluation of different LLMs. However, a significant performance disparity is ob-

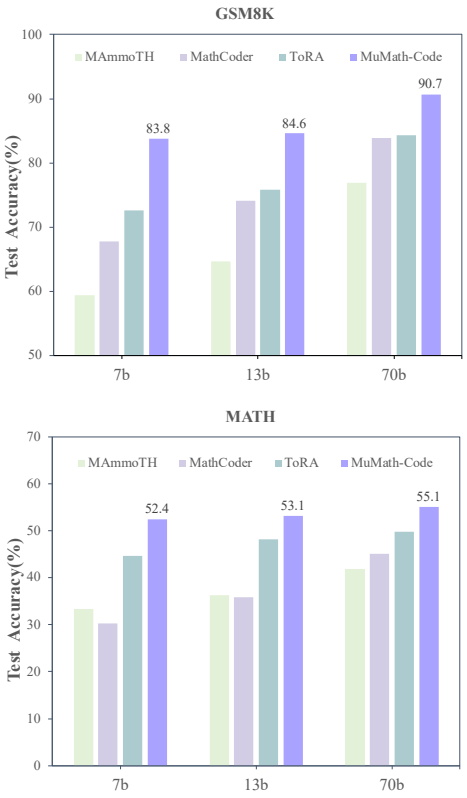


Figure 1: The comparison between our MuMath-Code and other state-of-the-art tool-use LLMs. MuMath-Code exhibits a substantial improvement in performance on both GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021a), relative to the previous approaches.

served between open-source LLMs, for instance, LLaMA (Touvron et al., 2023a,b), and their proprietary counterparts, when it comes to mathematical reasoning ability.

In recent years, many scholarly publications have been directed towards improving the mathematical proficiency of LLMs, which can be categorized into two distinct research trajectories: those that purely rely on natural language reasoning and those that incorporate external tools. The former methods are tool-free, mainly depends on data augmentation to enhance the models' mathematical reasoning capability, while the second trajectory (namely tool-use LLMs) are often coupled with external Python interpreters. From the perspective of knowledge distillation (Huang et al., 2022; Li et al., 2022; Magister et al., 2023; Ho et al., 2023; Fu et al., 2023b; Shridhar et al., 2023), both mainstream approaches transfer math reasoning abilities from the powerful teacher models (for instance, GPT-4) to the inferior open foundation models.

The tool-free methods synthesize a large number of new math problems and corresponding solutions, taking the original training math QA pairs as the initial data seeds. Scaling law theoretically provides the basis for the ongoing improvement of LLMs' performance by constantly incorporating new training data. Representative approaches are RFT (Yuan et al., 2023), MetaMath (Yu et al., 2023), WizardMath (Luo et al., 2023a), MuggleMath (Li et al., 2023), MuMath (You et al., 2024), etc. As for the second trajectory, code executors substantially supplant LLMs in particularly challenging computational and logical tasks, thereby alleviating the problem-solving burden on them. This tool-use category is exemplified by PAL (Gao et al., 2023), PoT (Chen et al., 2023), MAmmoTH (Yue et al., 2023), ToRA (Gou et al., 2023) and MathCoder (Wang et al., 2023).

Although the aforementioned research paths have been individually successful, to date, few methods have been developed that amalgamate their respective advantages. In this paper, we propose a novel method that integrates tool usage with data augmentation to synthesize a large amount of **mu**lti-perspective **math**ematical questions and solutions (we employ the augmenting methods introduced in a previous work MuMath (You et al., 2024)). Specifically, we utilize proprietary LLMs (like GPT-4) to generate Python **code** while synthesizing new solutions to math problems, and then fine-tune the open-source models (e.g., LLaMA) on the augmented dataset. The resulting model, **MuMath-Code**, is thus equipped with the ability to write code for math problem solving. During the inference phase, our MuMath-Code can generates both CoT (Wei et al., 2022) reasoning texts and Python code blocks. These code blocks are then extracted and executed by an external Python interpreter, and the execution results are returned to MuMath-Code for subsequent rounds of CoT reasoning or code generation until the final result is obtained or the maximum number of execution rounds is reached.

The multi-perspective mathematical question set comprises questions augmented via rephrasing (Yu et al., 2023), alteration (Li et al., 2023; You et al., 2024), FOBAR (Jiang et al., 2023a), BF-Trans (You et al., 2024), besides those from the original training sets. Regarding the solutions nested with Python code, we leverage a general pattern like the ones used in ToRA (Gou et al., 2023) and MathCoder (Wang et al., 2023): CoT-PoT interleaving. However, we propose prefix CoT, code debugging and pseudo-answer guidance filtering to improve the consistency and quality of our augmented solutions. The prefix CoT is a thoughtful analysis in pure natural language before code generation, making the LLMs consider this analysis while generating all the subsequent content, which thus are helpful for the models to learn the whole solution. Besides, we prompt GPT-4 to debug and correct the inexecutable code when requesting the solutions, and we keep the faulty code since this process of verification and correction can help boost the models' coding proficiency. Furthermore, for those synthesized questions via alteration, which lack ground truth answers as filtering guidance, we choose the majority-voting answers as the pseudo-answers. This process can increase the correctness of the generated solutions and thus improve the data quality generally. We name the proposed dataset as **MuMath-Code-Data** and denote it as $\mathcal{D}_{\mu\text{-}code}$.

Moreover, previous tool-use LLMs for math are derived by directly finetuning on code-nested data, which thus fail to fully harness the intrinsic natural language reasoning capability of the LLMs themselves. Different from the other tool-use methods, we design a two-stage training strategy to better combine the advantages of data augmentation and external code execution. The first stage is to enhance the models' pure language mathematical reasoning, where the largest (751K) dataset

proposed in MuMath (here called **MuMath-Data** and denoted as $\mathcal{D}_\mu$) is utilized to finetune LLaMA, and get an intermediate model, **MuMath**. In the second stage, we continue finetuning MuMath on MuMath-Code-Data to equip the model with the ability to write code for solving math problems. The resulting model, **MuMath-Code**, is thus can be prompted to leverage the Python interpreter to execute its generated code for securing the desirable outputs at inference time.

Our contributions are summarized as follows:

- We construct a multi-perspective augmentation dataset with code-nested solutions for math problem solving, called MuMath-Code-Data.
- We design a two-stage training strategy to equip the open LLMs with pure language reasoning and math related code generation capabilities, respectively.
- The obtained model, MuMath-Code, achieves new state-of-the-art performance among open LLMs across the in-domain math reasoning datasets as well as the out-of-domain ones. MuMath-Code-7B have 83.8 on GSM8K and 52.4 on MATH, while MuMath-Code-70B has achieved 90.7% on GSM8K and 55.1% on MATH.

2 Related Work

2.1 Tool-Free LLMs for Math

Rejection Sampling-based Fine-Tuning (RFT, Yuan et al., 2023) only augments the solutions via rejection sampling to collect a variety of different reasoning paths. Since RFT does not introduce new math questions, the diversity of the augmented dataset is quite low, which limits the performance improvement of the finetuned models. With the aim of incorporating a broader spectrum of questions, MetaMath (Yu et al., 2023) employs rephrasing, Self-Verification (SV, Weng et al., 2023) and FOBAR (Jiang et al., 2023a) to generate new questions. Ideally speaking, like the original questions, there are also ground truth answers for filtering solutions to these augmented questions. To bring in more diverse data, WizardMath (Xu et al., 2023; Luo et al., 2023a) and MuggleMath (Li et al., 2023) choose to create totally new questions via evolution or directional modification (changing numbers, adding conditions, increasing complexity, etc.) based on the seed questions. These altered questions have no ground truth answers, thus lacking a criterion to filter their corresponding synthesized solutions.

Furthermore, MuMath (You et al., 2024) leverages some of the aforementioned methods, and additionally proposes BF-Trans and expression replacement (etc.) to perform comprehensive augmentation, thus constructing a multi-perspective math question set with much greater diversity. For improving data quality, majority sampling serves as the filtering rule for the synthesized solutions to those new questions without deterministically known answers. Instead of solution filtering, a contemporary work, Xwin-Math (Li et al., 2024), employs verification with solution requesting during question synthesis, thereby improving the solvability of the questions and the correctness of the answers. Since there is no restriction on the direction of question modification, Xwin-Math theoretically offers a wider variety of diverse synthesized data. Balancing the efficacy and the ease of replication, in this paper the proposed MuMath-Code opts to employ the question augmentation from MuMath, although it is orthogonal to any other augmentation methods.

Nevertheless, as probabilistic models, LLMs inherently have limitations in logical reasoning and numerical computation. Thus, to improve the accuracy of mathematical problem-solving while relying solely on the capabilities of LLMs necessitates the utilization of a substantially larger dataset compared to tool-use methods.

2.2 Tool-Use LLMs for Math

Another research trajectory highlights the synergy between LLMs and external tools. Pioneering efforts along this include the Program-aided Language model (PAL, Gao et al., 2023) and Program of Thought (PoT, Chen et al., 2023). Moreover, MAmmoTH (Yue et al., 2023) integrates both CoT and PoT in a coarse-grained fashion (each sample corresponds to only one of these two possible solution types), enabling flexible inference where the finetuned models may adopt different methods for different questions. Different from MAmmoTH, ToRA (Gou et al., 2023) interleaves python code blocks and natural language reasoning parts over multiple turns for a same solution, which offers a more flexible combination of CoT and PoT. However, neither MAmmoTH nor ToRA employs query augmentation, thereby narrowing the range of math questions, which in effect, limits the problem-solving capabilities that can be ac-

quired. Wang et al. propose a contemporaneous work with ToRA, MathCoder (Wang et al., 2023), where each solution is also organized in an interleaved manner. Besides, they introduce interpolation problems to mitigate the disparity in difficulty level between GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021b). Hence, like our MuMath-Code, MathCoder is also an amalgamation of tool usage and math question augmentation, although the new questions it introduces are comparatively narrow in scope and limited in diversity.

Similar to ToRA and MathCoder, we also construct such solutions that intertwine Python code with pure language reasoning text to adaptably combine LLMs with external code executing tools. However, we propose prefix CoT, code debugging, and pseudo-answer guidance filtering to further enrich the solutions and improve their correctness. Additionally, different from MathCoder, the question augmentation we utilize are multi-perspective, thus offering greater diversity and exposing the model to a broader scope of novel questions, thereby significantly enhancing the model's generalization capabilities.

3 Methodology

We employ the augmented questions from MuMath (You et al., 2024), detailed in Appendix A, and synthesize code-nested solutions to them. To help the models better learn such solutions with multi-turn code generation, code execution and pure natural language reasoning, we propose prefix CoT, code debugging, and pseudo-answer guidance filtering to augment the quality of the synthetic data, as well as a two-stage training strategy. Figure 2 delineates the overall pipeline.

3.1 MuMath-Code-Data

To facilitate the interaction with the python interpreter, we synthesize the code-nested solutions for the models to learn, each consisting of multi-turn code generation, code execution and pure natural language reasoning.

Specifically, for each question from $\mathcal{Q}$, we prompt proprietary LLMs to request solutions each with at least one block of code, which is then extracted and passed to the external interpreter for execution. Every execution result is appended to the preceding content, right after the corresponding code block. If the code execution fails, we append a prompt to actively debug, using all the previous content as a whole new prompt to request the corrected code, which we then extract and execute again. By iterating this process multiple times, we obtain a reasoning path comprising code, execution outcomes and natural language analysis. This reasoning path is similar to that of MathCoder (Wang et al., 2023) and ToRA (Gou et al., 2023), but the differences lie in the use of our proposed prefix CoT, code debugging, and pseudo-answer guidance filtering, which will be elaborated on in this section. We marked MuMath-Data-Code as $\mathcal{D}_{\mu\text{-}code}$.

Prefix CoT We have observed that before generating code, a thorough pure natural language analysis is helpful for the models' performance. Therefore, we deliberately add a thoughtful CoT reasoning before code writing. The request prompt used is "*Analyze the question; list some knowledge points related to the question and beneficial for problem solving*".

Code Debugging Several research studies have shown that the use of error correction and verification data can improve the mathematical reasoning capabilities of LLMs. Therefore, we introduce an error correction process for our augmented dataset. Specifically, while constructing a solution, if the generated code fails to execute, we append a prompt "*The code above has encountered a problem. Now point out its mistakes and then correct them.*" for GPT-4 to debug the code and write new code until the executable code is obtained, or the maximum number of requests is reached. The failing code and error information are kept to equip the finetuned models with debugging ability, and thus enhance their coding proficiency for solving math problems.

Pseudo-Answer Guidance Filtering In MuMath-Data, we employ majority sampling to filter solutions. This provides us with pseudo-answers for the augmented questions corresponding no reference answers, which can also be employed for MuMath-Code-Data to select solutions. This approach improve the correctness of the synthesized solutions, thereby leading to an enhancement in the overall quality of the augmented data.

To sum up, we mark the i-th CoT (pure natural language reasoning) part as c_i; the i-th python code part is marked as p_i, which always begins with ```` ```python ```` and ends with ```` ``` ````; the i-th code execution output is denoted as o_i, beginning with

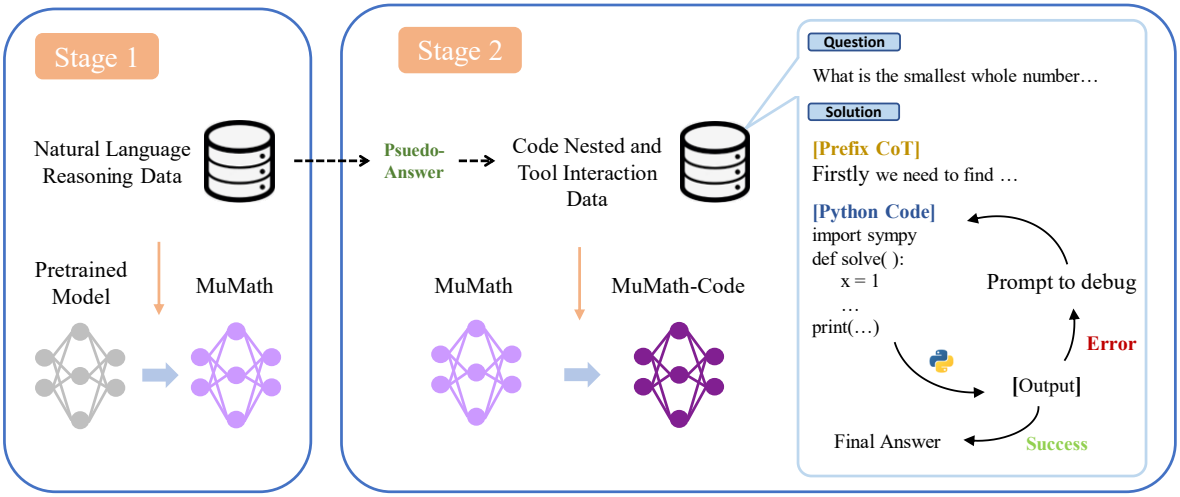


Figure 2: Illustration of our proposed method. The foundation model is first trained through an initial stage, resulting in an intermediary model that possesses more powerful math reasoning capability. This intermediary model is then further trained on the proposed dataset to learn code generation and tool interaction, leading to the final model, MuMath-Code.

```output and ending with ```. To formalize, one resulting solution s is defined as follows:

$$s = \left(\bigoplus_{i=1}^{n-1} c_i p_i o_i\right) c_n = c_1 p_1 o_1 c_2 p_2 o_2 ... c_{n-1} p_{n-1} o_{n-1} c_n, \quad (1)$$

where $\bigoplus$ stands for the concatenation of all the turns, and n is the number of CoT parts. See Appendix C for an example.

3.2 Two-Stage Training

Stage-1 The first stage training is on MuMath-Data (see Appendix B), where the models concentrate on learning the capability of pure CoT math reasoning. The learning target is as follows:

$$\mathcal{L}_1 = -\mathbb{E}_{q,s\sim\mathcal{D}_\mu}\left[\sum_{t=1}^{l} \log P\left(x_t | q, x_{<t}; \boldsymbol{\theta}\right)\right], \quad (2)$$

where the solution $s = (x_1, x_2, ..., x_l)$ contains l tokens, and $\boldsymbol{\theta}$ is the model parameter.

This training stage endow the models with a fairly strong mathematical reasoning capability, which can be seen as an preliminary task for the second stage learning.

Stage-2 The second stage training is on MuMath-Code-Data, where the models concentrate on PoT-CoT interleaved data to learn how to interact with an external tool (i.e., the Python interpreter). We mask the loss of the outputs from the code execution, which should not be learned by the models. The learning target is:

$$\mathcal{L}_2 = -\mathbb{E}_{q,s\sim\mathcal{D}_{\mu\text{-code}}}\left[\sum_{i=1}^{n} \log P\left(c_i p_i | q, \bigoplus_{j=1}^{i-1} c_j p_j o_j; \boldsymbol{\theta}\right)\right], \quad (3)$$

where $p_n = \varnothing$. The training process at Stage-2 is consistent with the inference, so we do not need to consider the issue of catastrophic forgetting (regarding the natural language reasoning in Stage-1). At inference time, after being given a mathematical problem, the finetuned model needs to generate code for problem solving, and then an external interpreter executes the code and returns the result for the model to continue generating. Therefore, Stage-2 training simulates the above inference process by masking out the losses of the execution outputs.

4 Experiments

4.1 Experimental Setup

Datasets Our seed datasets for synthesis are the training sets of two popular math reasoning benchmarks: GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021a). GSM8K contains elementary school math problems, comprising 7,473 training instances and 1,319 test instances;
```

while MATH encompasses math competition problems at the high school level with 7,500 training samples and 5,000 for test.

We take the MuMath (You et al., 2024) dataset (750K) as our $\mathcal{D}_{\mu}$ for Stage-1 training, and the MuMath augmented question set $\mathcal{Q}$ are utilized to construct $\mathcal{D}_{\mu\text{-}code}$ for Stage-2; in $\mathcal{Q}$, we request 15 solutions for each question that originates from GSM8K and 30 for MATH-related ones, and then perform filtering to get 30K samples for each question subset, making 600K in total.

Implementation Details Our study utilizes LLaMA-2 (7B, 13B and 70B) (Touvron et al., 2023b) and CodeLlama (7B, 13B, 34B, and 70B) (Rozière et al., 2023) as the foundation models for full-parameter finetuning, corresponding to MuMath-Code-L and MuMath-Code-CL as the resulting models. We employ AdamW as the optimizer and a cosine learning rate scheduler with a 0.03 warmup ratio. Across all the models and both stages, we train 3 epochs with a 128 global batch size. All the models except for LLaMA-70B and CodeLlama-70B are trained using the Deepspeed framework, while those two 70B models are trained using Megatron for the sake of speed. The hardware we use are NVIDIA H800 GPUs.

4.2 Comparison Results

As shown in Table 1, the comparison experiment of our models with the current state-of-the-art demonstrates that our approach consistently achieves superior performance across all scales of open-source models on all the datasets. Notably, our MuMath-Code-L 7B model has attained a test accuracy of 83.8 on the GSM8K, and MuMath-Code-CL 7B has reached a score of 52.4 on MATH. These outcomes surpass many 70B open-source baselines and even some proprietary LLMs. Additionally, our MuMath-Code-CL 34B and 70B achieve 55.0+ on MATH, two impressive results considering that they are accomplished by leveraging data augmentation techniques based on the original training set without the incorporation of extensive additional mathematical corpora for pre-training.

There are some noteworthy findings from the experimental statistics presented in the table, such as the performance of MuMath-Code-CL 13B on MATH, registering at 53.1, which is only marginally higher than that of MuMath-Code-CL 7B, which stands at 52.4. Moreover, the MuMath-Code-CL 34B's performance on MATH, scoring at 55.0, is very close to that of the MuMath-Code-CL 70B, which records a score of 55.1. We speculate that this may be attributed to the phenomenon where, beyond a certain threshold of data volume, the advantages conferred by increased model size may be diminished or even offset by the benefits derived from the expanded dataset. Additionally, variations in the training frameworks may also contribute to the observed discrepancy between the performances of MuMath-Code-CL 34B and 70B.

4.3 Effectiveness of the Two-Stage Training Strategy

MuMath-Code is derived from a two-stage training process that enhances the model's pure natural language reasoning capabilities and the ability to generate code and interact with external tools. In this section, we validate the efficacy of this bifurcated training strategy. Unless otherwise specified, all ablation experiments presented in this paper are conducted on 7B models, for the sake of time efficiency. We have designed a comparative evaluation of model performances for two-stage and one-stage training strategies. The two-stage training referred to here is as described in Section 3.2, which involves continuing training from the checkpoints of the first stage (the MuMath models). The one-stage training, directly applies the second stage of training on the base models. Table 2 illustrates the performance comparison of models derived from both strategies across different data volumes, revealing that training solely on $\mathcal{D}_{\mu\text{-}code}$ is worse than the two-stage training. Furthermore, by merging the training data from both stages into a single dataset for one-stage training, we observe that the outcomes are still not as favorable as those obtained from two separate training stages.

To further validate the effectiveness of our two-stage training strategy, we select MetaMath (Yu et al., 2023) and Xwin-Math (Team, 2023) 7B models as the initial checkpoints for Stage-2 training, emulating the scenario where relevant datasets were employed during the first stage (Given the the unavailability of the most recent models and dataset proposed in (Li et al., 2024), we opt to utilize Xwin-Math-7B-V1.0 detailed in the corresponding GitHub repository). Table 2 illustrates that models fine-tuned from MetaMath and Xwin-Math checkpoints on $\mathcal{D}_{\mu\text{-}code}$ (two-stage) outperform the one directly trained from Llama (single-stage), verifying the efficacy of a two-stage training strategy as well as the compatibility of our $\mathcal{D}_{\mu\text{-}code}$

Model	GSM8K	MATH	GSM-Hard	SVAMP	TabMWP	ASDiv	MAWPS
colsed-source LLMs							
Claud-3 Opus (Anthropic, 2024)	95.0	60.1	-	-	-	-	
GPT-4 (OpenAI, 2023b)	92.0	42.5	64.7	93.1	67.1	91.3	97.6
GPT-4 (PAL)	94.2	51.8	77.6	94.8	95.9	92.6	97.7
GPT-3.5 (OpenAI, 2023a)	80.8	35.5	55.9	83.0	69.1	87.3	94.6
GPT-3.5 (PAL)	78.6	38.7	67.6	77.8	79.9	81.0	89.4
tool-free open LLMs							
7B							
LLaMA-2 (Touvron et al., 2023b)	13.3	4.1	7.8	38.0	31.1	50.7	60.9
LLaMA-2 SFT (Touvron et al., 2023b)	41.3	7.2	16.1	31.9	27.8	47.4	60.0
WizardMath (Luo et al., 2023a)	54.9	10.7	20.6	57.3	38.1	59.1	73.7
MetaMath (Yu et al., 2023)	66.5	19.8	-	-	-	-	-
MuggleMath (Li et al., 2023)	68.4	-	-	-	-	-	-
MuMath (You et al., 2024)	70.9	22.0	-	76.8	-	93.6	87.3
13B							
LLaMA-2 (Touvron et al., 2023b)	24.3	6.3	13.6	43.1	39.5	56.3	70.4
LLaMA-2 SFT (Touvron et al., 2023b)	51.1	9.2	22.3	46.3	35.8	58.6	75.0
WizardMath (Luo et al., 2023a)	63.9	14.0	28.4	64.3	46.7	65.8	79.7
MetaMath (Yu et al., 2023)	72.3	22.4	-	-	-	-	-
MuggleMath (Li et al., 2023)	74	-	-	-	-	-	-
MuMath (You et al., 2024)	76.4	25.3	-	-	-	-	-
70B							
LLaMA-2 (Touvron et al., 2023b)	57.8	14.4	36.0	73.6	57.5	76.0	92.4
LLaMA-2 SFT (Touvron et al., 2023b)	69.3	14.9	39.0	64.0	53.0	71.3	84.8
WizardMath (Luo et al., 2023a)	81.6	22.7	50.3	80.0	49.8	76.2	86.2
MetaMath(Yu et al., 2023)	82.3	26.6	-	-	-	-	-
MuggleMath (Li et al., 2023)	82.3	-	-	-	-	-	-
MuMath (You et al., 2024)	84.5	32.2	-	87.6	-	96.6	92.0
tool-use open LLMs							
7B							
MAmmoTH (Yue et al., 2023)	53.6	31.5	-	67.7	-	-	-
MAmmoTH-Coder	59.4	33.4	-	71.4	-	-	-
CodeLLama (PAL) (Rozière et al., 2023)	34.0	16.6	33.6	59.0	47.3	61.4	79.6
MathCoder-L (Wang et al., 2023)	64.2	23.3	-	71.5	-	-	-
MathCoder-CL (Wang et al., 2023)	67.8	30.2	-	70.7	-	-	-
ToRA (Gou et al., 2023)	68.8	40.1	54.6	68.2	42.4	73.9	88.8
ToRA-Code (Gou et al., 2023)	72.6	44.6	56.0	70.4	51.6	78.7	91.3
MuMath-Code-L	**83.8**	48.8	70.5	87.6	65.6	86.2	94.7
MuMath-Code-CL	82.6	**52.4**	**70.6**	**88.1**	**66.9**	**87.4**	**95.3**
13B							
MAmmoTH (Yue et al., 2023)	62.0	34.2	-	72.4	-	-	-
MAmmoTH-Coder(Yue et al., 2023)	64.7	36.3	-	73.7	-	-	-
CodeLlama (PAL) (Rozière et al., 2023)	39.9	19.9	39.0	62.4	59.5	65.3	86.0
MathCoder-L (Wang et al., 2023)	72.6	29.9	-	76.9	-	-	-
MathCoder-CL (Wang et al., 2023)	74.1	35.9	-	78.0	-	-	-
ToRA (Gou et al., 2023)	72.7	43.0	57.3	72.9	47.2	77.2	91.3
ToRA-Code (Gou et al., 2023)	75.8	48.1	60.5	75.7	65.4	81.4	92.5
MuMath-Code-L	84.3	49.9	70.6	**87.9**	64.9	**86.4**	94.9
MuMath-Code-CL	**84.6**	**53.1**	**70.8**	86.8	**67.2**	85.2	**95**
34B							
CodeLLaMa (PAL) (Rozière et al., 2023)	53.3	23.9	49.4	71.0	63.1	72.4	91.5
MAmmoTH-Coder (Yue et al., 2023)	72.7	43.6	-	84.3	-	-	-
MathCoder-CL (Wang et al., 2023)	81.7	45.2	-	82.5	-	-	-
ToRA (Gou et al., 2023)	80.7	50.8	63.7	80.5	70.5	84.2	**93.3**
MuMath-Code-CL	**87.6**	**55.0**	**68.8**	**91.4**	**74.9**	**87.9**	92.9
70B							
LLaMA-2 (PAL)	55.2	18.3	50.0	74.6	59.5	71.9	92.8
MAmmoTH (Yue et al., 2023)	76.9	41.8	-	82.4	-	-	-
MathCoder-L (Wang et al., 2023)	83.9	45.1	-	84.9	-	-	-
ToRA (Gou et al., 2023)	84.3	49.7	67.2	82.7	74.0	86.8	93.8
MuMath-Code-L	**90.7**	52.8	68.6	**93**	74	**88.4**	**95.4**
MuMath-Code-CL	89.5	**55.1**	**70.1**	92.9	**77.4**	87.9	94.7

Table 1: Comparison of the state-of-the-art methods on various datasets. For the *tool-use open LLMs*, the best results are bolded and the second best underlined among the same scale models tested on the same datasets.

Inference	Training Strategy	LLaMA		CodeLlama	
		GSM8K	MATH	GSM8K	MATH
Tool free	$\mathcal{D}_{meta}$	66.5	19.8	-	-
	$\mathcal{D}_{xwin}$	66.6	17.4	-	-
Tool use	$\mathcal{D}_{\mu\text{-}code}$	81.2	46.2	81	49.8
	$\mathcal{D}_{\mu} + \mathcal{D}_{\mu\text{-}code}$	82.7	47.1	81.3	49.1
	$\mathcal{D}_{meta} \rightarrow \mathcal{D}_{\mu\text{-}code}$	82.3	47.4	-	-
	$\mathcal{D}_{xwin} \rightarrow \mathcal{D}_{\mu\text{-}code}$	82.0	47.2	-	-
	$\mathcal{D}_{\mu} \rightarrow \mathcal{D}_{\mu\text{-}code}$	83.8	48.8	82.6	52.4

Table 2: A two-stage training strategy improves the models' performance, as opposed to a single-stage training.

Sythesized Solutions	LLaMA		CodeLlama	
	GSM8K	MATH	GSM8K	MATH
w all	83.8	48.8	82.6	52.4
w/o prefix CoT	81.3	47.5	81.8	49.4
w/o code debugging	82	47.1	82.1	52.1
w/o either	81.0	46.8	81.3	49.0

Table 3: Ablation study for prefix CoT and code debugging.

with different first-stage CoT datasets.

4.4 Ablation Studies

To verify our proposed prefix CoT and code debugging, we respectively modify the solutions in $\mathcal{D}_{\mu\text{-}code}$ via two distinct approaches: the first approach involves the removal of the prefix CoT, thereby eliminating the detailed preliminary analysis and directly begining with code writing; the second approach consists of retaining only the final and successfully executed code and omitting all the other inexecutable code before as well as the corresponding debugging process. The results of this ablation study are presented in Table 3, which demonstrates that the exclusion of either the prefix CoT or code debugging leads to a decline in the models' test accuracy. This emphatically underscores the significance of a thorough analysis prior to code writing and the code mistake correction process for the models' learning.

Moreover, we conduct another ablation experiment on pseudo-answer guidance filtering. In Section 3.1, we note that pseudo-answers are suitable for synthetic questions that lack a definitive correct answer, namely those in Q_{alter} and $Q_{replace}$. In MuMath, majority voting is utilized to assign pseudo-answers to these questions. These pseudo-answers are then also employed to filter the data for $\mathcal{D}_{\mu\text{-}code}$ in the second training stage. As illustrated in Table 4, fine-tuning the model with data filtered through this pseudo-answer technique proves to be more beneficial than solutions obtained through directly random sampling. This trend holds across data volumes ranging from 30K to 180K.

Data Size	Pseudo-Answer	LLaMA		CodeLlama	
		GSM8K	MATH	GSM8K	MATH
30K	w	65.4	33.4	67.3	38.5
	w/o	65.9	32.4	67.6	36.9
60K	w	71.4	37.6	73.4	42.7
	w/o	71.2	36.6	72.3	40.4
90K	w	75.1	39.3	75.2	44.5
	w/o	74.8	37.9	74.2	41.9
120K	w	76.1	40.7	76.9	45.7
	w/o	74.6	40.3	75.9	43.6
150K	w	77.8	42.7	76.8	46
	w/o	75.8	41.5	76.4	45
180K	w	77.3	43.5	78.5	47.3
	w/o	76.7	42.7	77.7	46.5

Table 4: Ablation study for pseudo-answer guidance filtering.

5 Conclusion

In this paper, we propose a multi-perspective and code integrated math reasoning dataset called MuMath-Code-Data, where each solution contains multi-turn code generation, code execution and pure natural language analysis (CoT). Through a two-stage training strategy, our MuMath-Code models outperforms the state-of-the-art open methods and even some powerful proprietary ones across different scales on the in-domain reasoning datasets (i.e., GSM8K and MATH) as well as those out-of-domain ones. Additionally, ablation studies demonstrates the effectiveness of our three novel methods for the data synthesis: prefix CoT, code debugging and pseudo-answer guidance filtering. Our work represents a new attempt at integrating mathematical question augmentation (tool-free) with code generation and execution (tool-use) to enhance the mathematical reasoning capabilities of LLMs, and we hope it can inspire subsequent research endeavors.

6 Limitations

Our work is limited by the capabilities of the LLMs used to synthesize new math reasoning data.

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*.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. https://www.anthropic.com/news/claude-3-family.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *ArXiv*, abs/2005.14165.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yao Fu, Litu Ou, Mingyu Chen, Yuhao Wan, Hao Peng, and Tushar Khot. 2023a. Chain-of-thought hub: A continuous effort to measure large language models' reasoning performance.

Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. 2023b. Specializing smaller language models towards multi-step reasoning. *arXiv preprint arXiv:2301.12726*.

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

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. 2021a. Measuring mathematical problem solving with the MATH dataset.

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

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2023. Large language models are reasoning teachers.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2022. Large language models can self-improve.

Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James T. Kwok. 2023a. Forward-backward reasoning in large language models for mathematical verification.

Weisen Jiang, Yu Zhang, and James Kwok. 2023b. Effective structured prompting by meta-learning and representative verbalizer. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 15186–15199. PMLR.

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.

Chen Li, Weiqi Wang, Jingcheng Hu, Yixuan Wei, Nanning Zheng, Han Hu, Zheng Zhang, and Houwen Peng. 2024. Common 7b language models already possess strong math capabilities. *arXiv preprint arXiv:2403.04706*.

Chengpeng Li, Zheng Yuan, Hongyi Yuan, Guanting Dong, Keming Lu, Jiancan Wu, Chuanqi Tan, Xiang Wang, and Chang Zhou. 2023. Query and response augmentation cannot help out-of-domain math reasoning generalization.

Shiyang Li, Jianshu Chen, Yelong Shen, Zhiyu Chen, Xinlu Zhang, Zekun Li, Hong Wang, Jing Qian, Baolin Peng, Yi Mao, Wenhu Chen, and Xifeng Yan. 2022. Explanations from large language models make small reasoners better.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023a. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023b. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. 2023. Teaching small language models to reason.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2022. MetaICL: Learning to learn in context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2791–2809, Seattle, United States. Association for Computational Linguistics.

OpenAI. 2023a. Chatgpt: Optimizing language models for dialogue. https://openai.com/blog/chatgpt.

OpenAI. 2023b. Gpt-4 technical report.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, , and I. Sutskever. 2019. Language models are unsupervised multitask learners. *Technical Report*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer.

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.

Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. 2023. Distilling reasoning capabilities into smaller language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 7059–7073, Toronto, Canada. Association for Computational Linguistics.

Xwin-Math Team. 2023. Xwin-math.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models.

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. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

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

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.

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.

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.

Weihao You, Shuo Yin, Zhilong Ji, Xudong Zhao, Guoqiang Zhong, and Jinfeng Bai. 2024. Mumath: Multi-perspective data augmentation for mathematical reasoning in large language models. In *2024 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

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

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.

A Question Synthesis

A.1 MuMath Augmented Questions

The original questions from the training sets of GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021a) are taken as the seed question set $Q_{original}$. The question augmenting methods employed in MuMath are conducted on this seed set, which are briefly concluded as follows:

(1) Rephrasing Rewrite a text while keeping the original meaning unchanged. Based on the fact that a rephrased question holds the same meaning as the original one, the final answer of it should also be the same. We denote the rephrased question set as $Q_{rephrase}$.

(2) Question Alteration There are five manners to alter the original questions, like changing numbers and adding more conditions, concluded in MuggleMath (Li et al., 2023). The resultant question set created via alteration is referred to as $Q_{alter} = Q_{alter1} \cup Q_{alter2} \cup Q_{alter3} \cup Q_{alter4} \cup Q_{alter5}$. Besides, Expression Replacement, proposed in MuMath, firstly get the expressions of the solution to an original question, then change the calculation operators within them. Based on the changed expressions, a new question is asked to generate. $Q_{replace}$ represents the question set produced by this augmentation technique. Note that Q_{alter} and $Q_{replace}$ correspond no definitely correct answers due to modifications in the questions' intrinsic meanings.

(3) FOBAR Following Jiang et al. (2023a), we mask a certain condition in an initial question by substituting it with "X", and meanwhile give the answer to the original question as a new condition, thereby creating a reverse question that seeks to determine the value of the unknown X. Q_{fobar} is utilized to mark the FOBAR question set.

(4) BF-Trans Backward-Forward Transformation (BF-Trans), proposed in MuMath, aims to construct such backward questions that can be answered through direct reasoning, bypassing the necessity of solving equations to find the unknown variables (thus resemble the data sampled from the original distribution). For a certain question-answer pair, BF-Trans firstly utilize FOBAR to transform the original question into a backward one; secondly, we rephrase the FOBAR question into a new form where the masked value is requested directly instead of employing an unknown variable X, resulting in a "secondary forward" question which we called BF-Trans question. The set of these BF-Trans questions is marked as Q_{bf}.

To sum up, all the 10 aforementioned subsets (5 in Q_{alter}) constitute the resulting question set $\mathcal{Q} = Q_{original} \cup Q_{rephrase} \cup Q_{alter} \cup Q_{replace} \cup Q_{fobar} \cup Q_{bf}$. Based on $\mathcal{Q}$, we generate 2 datasets called **MuMath-Data** and **MuMath-Code-Data**, emphasizing pure natural language mathematical reasoning and tool interaction via code generation, respectively.

B MuMath-Data

MuMath-Data (denoted as $\mathcal{D}_\mu$) is just the largest dataset from MuMath, which contains about 750K samples with pure CoT reasoning solutions to questions in $\mathcal{Q}$.

Majority Sampling As is introduced in the paper of MuMath, for Q_{alter} and $Q_{replace}$ whose each question has no reference answer, majority sampling is utilized to filter all the randomly generated solutions and only those solutions with the majority answers are kept. In other words, each majority answer serves as a pseudo-answer to the corresponding question.

C Examples

Example C.1: MuMath-Code-Data QA Pair

Question:
What is the smallest whole number that has a remainder of 1 when divided by 4, a remainder of 1 when divided by 3, and a remainder of 2 when divided by 5?

Solution
We need to find a positive integer solution for the system of linear congruences. We can use the Chinese Remainder Theorem to solve this system. The Chinese Remainder Theorem states that for any system of equations like this, there exists a solution. Let's find the solution using sympy's solve_congruence function.

```python

from sympy.ntheory.modular import solve_congruence
def smallest_number():
    congruences = [('1', '4'), ('1', '3'), ('2', '5')]
    result = solve_congruence(*congruences)
    return result[0]

result = smallest_number()
print(result)
```
```output
ValueError: 1 is not an integer
```
The code above has encountered a problem. Now point out its mistakes and then correct them.
Mistake:
The error is in the format of the input to the function 'solve_congruence'. The function is expecting tuples of the format '(remainder, modulus)', but the inputs need to be integers, not strings.

Correction:
Let's modify the code to convert the strings to integers.

```python
def smallest_number():
    congruences = [(1, 4), (1, 3), (2, 5)]
    result = solve_congruence(*congruences)
    return result[0]

result = smallest_number()
print(result)
```
```output
NameError: name 'solve_congruence' is not defined
```
The code above has encountered a problem. Now point out its mistakes and then correct them.
Mistake:
The error is that the 'solve_congruence' function is not defined. It should be imported from the 'sympy' module before it can be used.

Correction:
Let's modify the code to import the 'solve_congruence' function from the 'sympy' module.

```python
from sympy.ntheory.modular import solve_congruence

def smallest_number():
    congruences = [(1, 4), (1, 3), (2, 5)]
    result = solve_congruence(*congruences)
    return result[0]

result = smallest_number()
print(result)
```
```output
37
```
The smallest whole number that has a remainder of 1 when divided by 4, a remainder of 1 when divided by 3, and a remainder of 2 when divided by 5 is $\\boxed{37}$.

D Additional Experimental Results

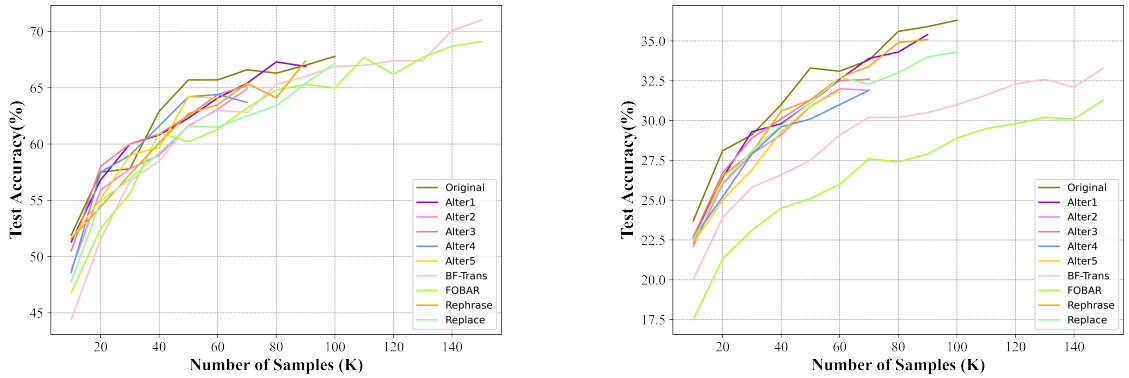
D.1 Two-Stage Training

$\mathcal{D}_{\mu\text{-}code}$	Training Strategy	LLaMA		CodeLlama	
		GSM8K	MATH	GSM8K	MATH
100K	$\mathcal{D}_{\mu\text{-}code}$	69.7	36.1	72.4	41
	$\mathcal{D}_{\mu} \rightarrow \mathcal{D}_{\mu\text{-}code}$	77.1	41.5	80.3	46.1
200K	$\mathcal{D}_{\mu\text{-}code}$	76.2	41.4	78.3	44.2
	$\mathcal{D}_{\mu} \rightarrow \mathcal{D}_{\mu\text{-}code}$	80.4	46	80.7	49.1
300K	$\mathcal{D}_{\mu\text{-}code}$	77.1	43.7	78.2	46.8
	$\mathcal{D}_{\mu} \rightarrow \mathcal{D}_{\mu\text{-}code}$	79.5	46.4	83.2	50.2
400K	$\mathcal{D}_{\mu\text{-}code}$	78	44.3	79	47.8
	$\mathcal{D}_{\mu} \rightarrow \mathcal{D}_{\mu\text{-}code}$	81.6	48.5	81.9	50.9
500K	$\mathcal{D}_{\mu\text{-}code}$	79.8	45.7	80.2	48.9
	$\mathcal{D}_{\mu} \rightarrow \mathcal{D}_{\mu\text{-}code}$	82.8	48.7	82.6	52.2
600K	$\mathcal{D}_{\mu\text{-}code}$	81.2	46.2	81	49.8
	$\mathcal{D}_{\mu} \rightarrow \mathcal{D}_{\mu\text{-}code}$	83.8	48.8	82.6	52.4

Table 5: We vary the data volumes of $\mathcal{D}_{\mu\text{-}code}$. It is observed that training solely on $\mathcal{D}_{\mu\text{-}code}$ is consistently inferior to the two-stage training across all data volumes.

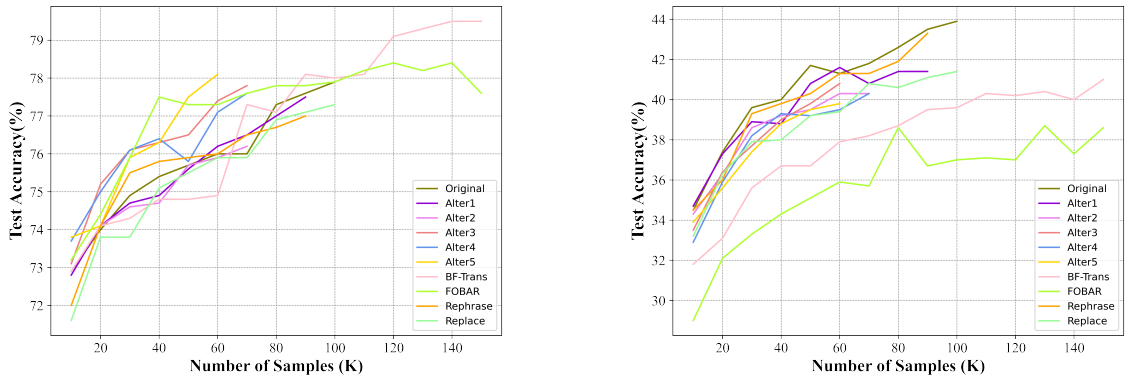
D.2 Scaling Study

The scaling experiments for various subsets of the MuMath-Code-Data are depicted in Figure 3. These curves represent the performance changes of models trained on different data subsets with respect to the



(a) Test on GSM8K (b) Test on MATH

Figure 3: Scaling all the subsets of MuMath-Code-Data. The models undergo a single stage (only Stage-2) of training.



(a) Test on GSM8K (b) Test on MATH

Figure 4: Scaling all the subsets of MuMath-Code-Data. Initially, the model has already been finetuned on MuMath-Data (thus two-stage training results). It is observable that the curves show very similar trends to those in Figure 3.

number of samples. The base model is LLaMA 7B and it is directly trained on the subsets of $\mathcal{D}_{\mu\text{-}code}$ (single-stage training). It is evident that with the increase in data volume, all subsets continuously contribute to the enhancement of the models' performance, and the curves still do not show saturation. This indicates that employing our methodology allows for the continued addition of data to further improve the LLMs' mathematical reasoning capabilities. For the two-stage scenario where the initial model is an intermediate checkpoint from Stage-1, please see Figure 4.