MathGenie: Generating Synthetic Data with Question Back-translation for Enhancing Mathematical Reasoning of LLMs

Zimu Lu[*], **Aojun Zhou**[*], **Houxing Ren, Ke Wang, Weikang Shi**
Junting Pan, **Mingjie Zhan**[†], **Hongsheng Li**[†]
Multimedia Laboratory (MMLab), The Chinese University of Hong Kong
luzimu@mail.ustc.edu.cn   {aojunzhou, zmjdll}@gmail.com
hsli@ee.cuhk.edu.hk

Abstract

Large language models (LLMs) have exhibited great potential in mathematical reasoning. However, there remains a performance gap in this area between existing open-source models and closed-source models such as GPT-4. In this paper, we introduce **MathGenie**, a novel method for generating diverse and reliable math problems from a small-scale problem-solution dataset (denoted as *seed data*). We augment the ground-truth solutions of our seed data and train a back-translation model to translate the augmented solutions back into new questions. Subsequently, we generate code-integrated solutions for the new questions. To ensure the correctness of the code-integrated solutions, we employ rationale-based strategy for solution verification. Various pretrained models, ranging from 7B to 70B, are trained on the newly curated data to test the effectiveness of the proposed augmentation technique, resulting in a family of models known as *MathGenieLM*. These models consistently outperform previous open-source models across five representative mathematical reasoning datasets, achieving state-of-the-art performance. In particular, MathGenieLM-InternLM2 achieves an accuracy of 87.7% on GSM8K and 55.7% on MATH, securing the best overall score among open-source language models.

1 Introduction

Large language models (LLMs), such as GPT-4 (OpenAI, 2023), and others (Touvron et al., 2023; Yue et al., 2023; Gou et al., 2023; Wang et al., 2023), have demonstrated outstanding capabilities in mathematical reasoning. Existing methods explored three main types of solution formats for tackling the math-solving problem: text-only Chain-of-Thought (CoT) solutions (Wei et al., 2022), code-only Program-of-Thought (PoT) solutions (Chen

et al., 2022), and code-integrated solutions (Zhou et al., 2023). Among these, code-integrated solutions demonstrate superior performance over both CoT and PoT solutions (Gou et al., 2023; Wang et al., 2023; Zhou et al., 2023), indicating their effectiveness in enhancing problem-solving abilities.

In this paper, we focus on creating diverse and reliable augmented questions and ensuring the reliability of the generated code-integrated solutions, thereby better finetuning pretrained base models. Existing works, such as ToRA (Gou et al., 2023) and MathCoder (Wang et al., 2023), build code-integrated solutions and augment math questions using off-the-shelf GPT-4. However, scaling up the training data with GPT-4 becomes prohibitively expensive.

Consequently, developing a free open-source model to generate large-scale synthetic data presents a promising alternative, offering scalability and cost-effectiveness (Yuan et al., 2023; Singh et al., 2023). To effectively scale-up math problem-solving data, we focus on handling two critical challenges: (1) how to generate high-quality and diverse math problems to aid in generalization, and (2) how to generate accurate and reliable solutions for the augmented problems without human-annotated ground truth, preferably in a code-integrated format. A unified framework, *MathGenie*, is proposed, which consists of three components as shown in Fig. 1 to tackle the above-mentioned challenges: *Iterative Solution Augmentation*, *Question Back-translation* and *Verification-Based Solution Filtering*.

Iterative Solution Augmentation and Question Back-translation aims to generate diverse and reliable math problems. Unlike direct question augmentation (Yu et al., 2023), the proposed math problem back-translation leverages the constraints and logical relationships inherent in mathematical solutions to create a diverse and high-quality set of new math problems. Specifically, we iteratively

---

[*]Equal contribution   [†]Corresponding author

Step 1 : Iterative Solution Augmentation

Solution Augmentation Model (M_{text})

Iterative Solution Augmentation

Seed Solutions (S^0) → Solutions 1 (S^1) → Solutions 2 (S^2) → ... → Solutions K (S^K)

Step 2: Question Backtranslation

Backtranslation Model ($M_{backtrans}$)

Augmented Solutions (S^{Aug}) — Question Backtranslation → Augmented Questions (Q^{Aug})

Step 3: Verification-Based Solution Filtering

Augmented Questions (Q^{Aug}) — Code-Integrated Solution Generation → Augmented Questions and Code Integrated Solutions (Q^{Aug}, S_{code}^{Aug}) — Verification-based Filtering → Filtered Augmented Questions and Code Integrated Solutions (Q^{Aug}, S_{code}^{Aug})

Verification

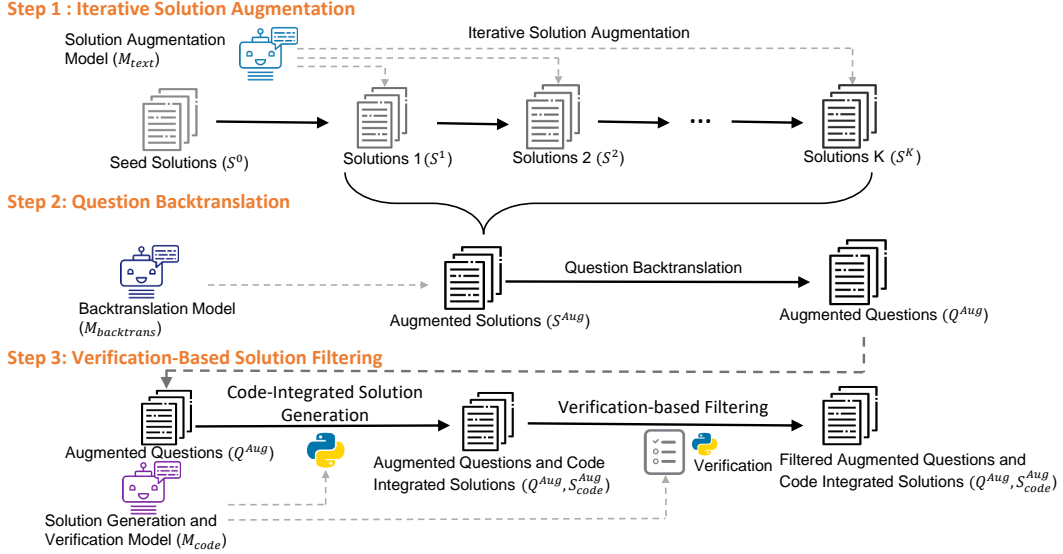Solution Generation and Verification Model (M_{code})

Figure 1: Framework of MathGenie. Iterative Solution Augmentation augments human-annotated solutions in GSM8K and MATH to create new solutions, as shown in Step 1. These solutions are then back-translated to new questions using Question Back-translation, demonstrated in Step 2. Then reliable code-integrated solutions are curated using Verification-Based Solution Filtering, by generating solutions and filtering them using verification rationales, as shown in Step 3.

augment the human-annotated solutions from the relatively small training sets of MATH (Hendrycks et al., 2021) and GSM8K (Cobbe et al., 2021), generating a large-scale collection of augmented new solutions, as shown in Step 1 of Fig. 1. These solutions are then processed by a math back-translation model, $M_{\text{backtrans}}$, to back-translate the augmented solutions into their corresponding math questions, as demonstrated in Step 2 of Fig. 1. This method draws inspiration from Instruction Back-translation (Li et al., 2023b), which back-translates instructions from texts in a web corpus. However, the key difference is that our source solutions for back-translation are augmented from existing ones to ensure the reliability and solvability of the augmented questions.

These newly generated math problems lack reliable ground-truth solutions, which necessitates the proposed Verification-Based Solution Filtering. We first build a model, M_{code}, capable of generating code-integrated solutions and verifying these solutions. Then, code-integrated solutions of the new questions are generated with this model. To enhance the reliability of these code-integrated solutions, we use M_{code} to verify the model-generated solutions by generating verification rationales for them, as demonstrated in Step 3 of Fig. 1. The verification rationales use interleaved natural language and code to verify the correctness of the solutions,

as shown in Tab. 13 and Tab. 14 in Appendix G. Only the solutions verified to be correct are retained, thus improving the accuracy and quality of the generated data.

Based on the proposed MathGenie framework, we obtain a large-scale model-generated math problem-solution dataset, *MathGenieData*, featuring diverse augmented questions and reliable code-integrated solutions.

To evaluate the effectiveness of the question-solution augmentation framework MathGenie, we finetune various state-of-the-art pretrained models, ranging from 7B to 70B. This results in **Math-GenieLM**, a new family of math models with excellent performance. Our models demonstrate high accuracy across five diverse and representative math datasets: MATH, GSM8K, SVAMP, Simuleq, and Mathematics. In particular, MathGenieLM-InternLM2 achieves an accuracy of 87.7% on GSM8K and 55.7% on MATH, achieving the best overall score. When enhanced by majority voting, MathGenieLM-Llama-2-70B attains a 10-path accuracy rate of 91.7% on GSM8K and 63.3% on MATH.

The main contributions of this paper are summarized as follows: (1) We propose the Math-Genie pipeline, which is designed to enhance the scale, diversity, and quality of synthetic math questions, as well as to improve the accuracy of the
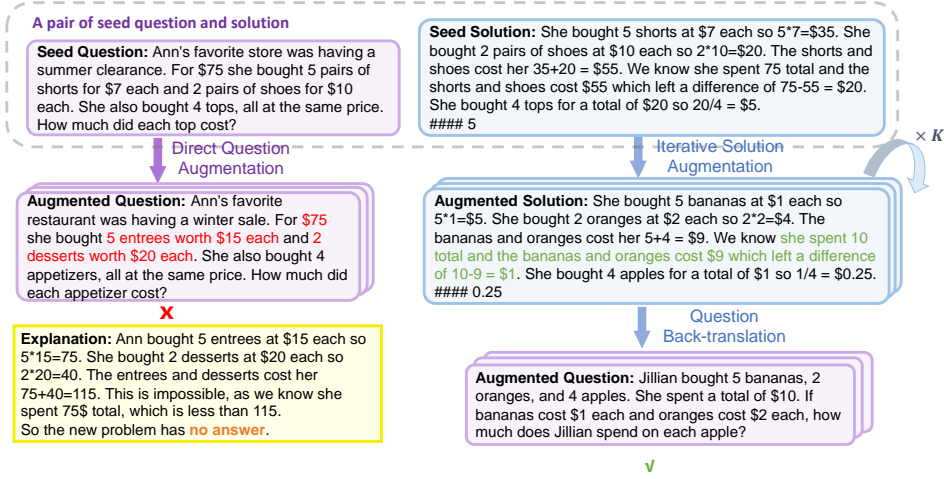
Figure 2: Comparison between Direct Question Augmentation (left) and Iterative Solution Augmentation and Question Back-translation (right). Direct Question Augmentation damages the hidden constraints between the conditions (cost of part of the things bought must not be more than the total cost), thus producing a question with no answer. Question Back-translation considers the solution, and correctly augments the question.

code-integrated solutions generated for them. (2) We conduct extensive experiments on various pre-trained language models, demonstrating consistently superior performance across multiple math datasets.

2 MathGenie

In this section, we introduce MathGenie, a pipeline for creating diverse and reliable math problems through back-translation, and curating high-quality code-integrated solutions through verification. We begin by introducing the seed data and solution generator model. Next, we present the proposed Math-Genie pipeline, which consists of three key steps, as shown in Fig. 1: Iterative Solution Augmentation, Question Back-translation, and Verification-Based Solution Filtering.

Seed Data. The seed data consists of two parts: (1) The first part is used for data augmentation, consisting of 15K math problems and human-annotated solutions from the training sets of GSM8K and MATH. We denote it as $\mathcal{D}_{\text{text}} = \{(q^i, s^i)\}_{i=1}^{n}$, where q^i is the i-th question, s^i is its natural-language solution, and n is the total number of cases. (2) The second part is used for training our candidate solution generator model, which serves to generate candidate solutions for the augmented questions. We denote this part of the seed data as $\mathcal{D}_{\text{code}} = \{(q^i, s_{\text{code}}^i, v_{\text{code}}^i)\}_{i=1}^{N}$, where q^i is the question, s_{code}^i is its code-integrated solution, and v_{code}^i is the code-integrated verification

rationales for the question-solution pair. It contains 80K samples of code-integrated solutions for problems in GSM8K and MATH, as well as code-integrated verification rationales for these solutions. Multiple different solutions are collected for each question. We acquire these solutions and verification rationales using the GPT-4 Code Interpreter, which consist of interleaved natural language and code.

Candidate Solution Generator. The candidate solution generator is a Llama-2 70B model trained with $\mathcal{D}_{\text{code}}$ and denoted as M_{code}. The code-integrated solutions in $\mathcal{D}_{\text{code}}$ enables M_{code} to output candidate code-integrated solutions for given math problems, similar to (Wang et al., 2023). It has an accuracy of 86.4% on GSM8K and 49.5% on MATH. The verification rationales in $\mathcal{D}_{\text{code}}$ enable M_{code} to effectively verify the solutions with code-integrated rationales. The training method of data in the code-integrated format is as described in Wang et al. (2023).

Iterative Solution Augmentation. Different from previous works that directly augment math questions (Luo et al., 2023), we propose to augment solutions first and then back-translate the augmented solutions to their corresponding questions to better constrain the question generation process and enhance the reliability of machine-generated questions. The proposed strategy is also different from the previous Instruction Back-translation method (Li et al., 2023b), which leverages large

amounts of texts in a web corpus. As existing solutions are limited in number and already have corresponding questions, it is necessary to augment them before the back-translation.

To augment the solutions in $\mathcal{D}_{\text{text}}$ into related new ones, we develop a solution augmentation model, M_{text}, by finetuning the LLaMA-2 70B model on high-quality instructional datasets, including OpenOrca[1] and Alpaca-GPT4[2]. M_{text} takes in a solution and a prompt instructing the model to augment it, and outputs an augmented solution. The prompts are shown in Tab. 1. The augmentations are carefully constrained and reliable. We iteratively augment each solution in $\mathcal{D}_{\text{text}}$. For convenience, the set of human-annotated solutions in $\mathcal{D}_{\text{text}}$ are denoted as $\mathcal{S}^0$. The solutions in $\mathcal{S}^0$ are augmented by M_{text} to create $\mathcal{S}^1$. As shown in Step 1 of Fig. 1, this process is repeated on the previously generated solutions, with $\mathcal{S}^2$ being created from $\mathcal{S}^1$, and so on. After K rounds, the final set of augmented solutions, denoted as $\mathcal{S}^{\text{Aug}}$, is created by taking the union of $\mathcal{S}^1, \mathcal{S}^2, \ldots, \mathcal{S}^K$:

$$\mathcal{S}^{\text{Aug}} = \mathcal{S}^1 \cup \mathcal{S}^2 \cup \cdots \cup \mathcal{S}^K. \quad (1)$$

Through iterative solution augmentation, each round produces a set of solutions that differs from the previous, making the solutions gradually deviate more from the original solutions. Consequently, the diversity of the augmented solutions is ensured, which leads to diverse augmented question-solution pairs as mentioned in the following sections. The iteration process is beneficial to the final performance, as demonstrated in Tab. 9 of Appendix B

Question Back-translation. We introduce Question Back-translation to translate the solutions in $\mathcal{S}^{\text{Aug}}$ back to their corresponding math problems. To enhance the accuracy of the translation, we build a Question Back-translation Model $M_{\text{backtrans}}$ by finetuning Llama-2 70B on the reversed pairs of question and solution in $\mathcal{D}_{\text{text}}$. The format of each sample in the finetuning data can be denoted as (s, q), where s is a solution, serving as input, and q is its corresponding question. Then, we present the solutions in $\mathcal{S}^{\text{Aug}}$ to $M_{\text{backtrans}}$, thereby translating them back to a set of new questions, denoted as $\mathcal{Q}^{\text{Aug}}$. An example of Question Back-translation is presented in Tab. 2.

Prompt
Replace the objects and verbs in Solution. Solution: {solution} You must replace the objects and verbs in the Solution with objects and verbs very different from before to create a new solution.
Add an extra step to the solution so that it is more complicated. Solution: {solution} You must add an extra step to the Solution to create a new solution.
Change the numbers in Solution. Solution: {solution} You must change the numbers in the Solution to create a new solution.
Replace the numbers and variables in Solution with different variables and numbers you pick randomly. Solution: {solution} You must replace the numbers and variables in the Solution with different numbers and variables you pick randomly to create a new solution.

Table 1: Different prompts for solution augmentation. "{solution}" is replaced with the solution to be augmented.

Input
Create a New Problem based on the Solution: For the soccer practice, Jack needs 3 days * 10 soccer balls/day = 30 soccer balls. Adding the soccer balls for the Gym class, he needs to give a total of 30 soccer balls + 10 soccer balls = 40 soccer balls.
Output
New Problem: Jack is in charge of the equipment room at his school. He needs to gather 10 soccer balls for the gym class and also give 3 days worth of soccer balls for the soccer team's practice. If each practice uses 10 soccer balls, how many soccer balls does Jack need to give out?

Table 2: An example of the proposed Question Back-translation. The solution is input to the question back-translation model, and the model outputs its corresponding question.

Question Back-translation operates on $\mathcal{S}^{\text{Aug}}$, which is less unpredictable than the texts from the web corpus used in Instruction Back-translation (Li et al., 2023b). By leveraging the constraints shown in the solutions, it is possible to create new questions more reliable than what direct question augmentation can produce, as validated in experiments.

Verification-based Solution Filtering. Existing open-source models such as MathCoder (Wang et al., 2023) only has the ability to solve math problems but are unable to effectively verify their solutions. We enhance M_{code}'s ability to verify the solutions by adding code-integrated verification rationales to the finetuning data. The training samples of verification rationales in the seed data are in the format of $(q, s_{\text{code}}, v_{\text{code}})$, where q and s_{code} are a pair of question and code-integrated solution, and v_{code} is the code-integrated verification. The (q, s_{code}) pairs are the input, while the model is trained to output v_{code}. In this way, the baseline math problem-solving model M_{code} acquires the ability to verify its solutions with rationales made

of interleaved natural language and code. This ability not only facilitates Verification-Based Solution Filtering, but can also play a role in enhancing inference accuracy.

To perform the proposed Verification-Based Solution Filtering, we first generate code-integrated solutions for each question in $\mathcal{Q}^{\text{Aug}}$. Initial filtering is performed using answer consistency (Wang et al., 2022), removing a question if its solutions reach different answers. We then present each question-solution pair to M_{code}, prompting it to output a code-integrated verification rationale, from which we can determine whether the solution is verified as correct or wrong. Examples of the verification process are shown in Appendix G. Candidate solutions that are verified to be wrong are abandoned. The process is demonstrated in Step 3 of Fig. 1.

The pipeline proposed above results in 170K samples of question and code-integrated solution pairs, denoted as AugData. AugData consists of two parts: the 110K samples augmented from GSM8K dataset, denoted as AugGSM8K, and the 60K samples augmented from MATH dataset, denoted as AugMATH. We denote the above mentioned seed data for training M_{code} as SeedData. Combining SeedData and AugData, we present the final dataset, **MathGenieData**, which can be used to finetune various pretrained models, such as Llama-2 (Touvron et al., 2023) and CodeLlama (Roziere et al., 2023), enhancing their problem-solving ability and solution verification skills. The resulting family of mathematical reasoning models is named MathGenieLM.

3 Experiments

3.1 Experimental Setup

Datasets. We evaluate our models on two in-domain datasets: GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021), whose training sets are used for finetuning. Additionally, we evaluate the final models on three out-of-domain datasets: SVAMP (Patel et al., 2021), Simuleq (Koncel-Kedziorski et al., 2016), and Mathematics (Davies et al., 2021), to evaluate the generalization ability of our proposed method.

Models. We perform full-parameter finetuning on various pretrained models, including Llama-2 7B, 13B, and 70B (Touvron et al., 2023), CodeLlama 7B, 13B, and 34B (Roziere et al., 2023), Llemma 7B and 34B (Azerbayev et al., 2023), Mistral 7B (Jiang et al., 2023), Mixtral-8x7B (Jiang

et al., 2024), and InternLM2 20B (Team, 2023). Finetuning details are described in Appendix E.

Compared methods. We compare MathGenieLM with closed-source models such as ChatGPT-3.5 (Brown et al., 2020), GPT-4 (OpenAI, 2023), and PaLM-2 (Anil et al., 2023), as well as open-source models such as Mammoth (Yue et al., 2023), MathCoder (Wang et al., 2023), ToRA (Gou et al., 2023), and WizardMath (Luo et al., 2023).

3.2 Main Results

Tab. 3 shows the accuracy of MathGenieLM across five datasets. Based on the results, we make the following observations: (1) For open-source models with parameters ranging from 7B to 70B, MathGenieLM achieves state-of-the-art performance. (2) MathGenieLM demonstrates particularly high performance on the three out-of-domain datasets compared to previous open-source models, showcasing the superior generalization capability of our method. (3) MathGenieLM's accuracy exceeds that of ChatGPT-3.5 and PaLM-2. However, there remains a noticeable gap when compared to GPT-4's performance. (4) MathGenieLM-Llemma-34B and MathGenieLM-InternLM2-20B reach over 55% accuracy on the challenging MATH dataset. This might be attributed to the high-quality math-related data they used in pretraining. (5) Mixtral-8x7B achieves excellent performance, demonstrating the potential of Mixture of Experts (MoE) models. The results in Tab. 3 are all obtained using greedy decoding.

Apart from the results obtained with greedy decoding, we also report the results of majority voting using multiple sampled paths (Wang et al., 2022), conducted on MathGenieLM-Llama-2-70B, compared with ToRA-Llama-2-70B. The results are shown in Tab. 4, where "k" represents the number of solutions generated for majority voting. We observe that, with $k = 10$, majority voting significantly increases the accuracy across all five datasets, yielding an average gain of 7.9%. Specifically, at $k = 10$, MathGenieLM-Llama-2-70B achieves an accuracy of 91.5% on GSM8K and 63.3% on MATH, significantly outperforming ToRA-70B at $k = 50$. This demonstrates the superior performance of our model.

3.3 Ablation Study

The following are some ablation studies. All finetuning in the ablation studies has been conducted using Mistral-7B as the base model.

Model	Base	Size	In-Domain		Out-of-Domain			Average
			GSM8K	MATH	SVAMP	Simuleq	Mathematics	
Colsed-Source Models								
ChatGPT-3.5	-	-	80.8	35.5	83.0	-	-	-
GPT-4	-	-	92.0	42.5	97.0	-	-	-
GPT-4 Code	-	-	97.0	69.7	-	-	-	-
PaLM-2	-	-	80.7	34.3	-	-	-	-
Open-Source Models								
Mammoth	CodeLlama	7B	59.4	33.4	71.4	45.9	55.4	53.1
MathCoder	CodeLlama	7B	67.8	30.2	70.7	49.6	55.8	54.8
ToRA	CodeLlama	7B	72.6	44.6	70.4	36.0	68.1	58.3
MathGenieLM	CodeLlama	7B	71.5	39.7	80.2	69.1	69.5	66.0
Mammoth	Llama-2	7B	53.6	31.5	67.7	41.2	46.3	48.1
MathCoder	Llama-2	7B	64.2	23.3	71.5	47.5	46.9	50.7
ToRA	Llama-2	7B	68.8	40.1	68.2	29.2	58.3	52.9
MathGenieLM	Llama-2	7B	71.7	33.0	78.5	61.4	65.0	61.9
MathGenieLM	Llemma	7B	<u>76.0</u>	**48.3**	<u>81.3</u>	**85.0**	**76.6**	**73.4**
MathGenieLM	Mistral	7B	**80.5**	<u>45.1</u>	**83.3**	<u>79.4</u>	<u>71.8</u>	<u>72.0</u>
Mammoth	CodeLlama	13B	64.7	36.3	73.7	47.1	61.5	56.7
MathCoder	CodeLlama	13B	74.1	35.9	78.0	60.7	62.5	62.2
ToRA	CodeLlama	13B	75.8	**48.1**	75.7	37.9	70.3	61.6
MathGenieLM	CodeLlama	13B	<u>78.5</u>	40.3	**84.5**	<u>76.7</u>	65.7	69.1
Mammoth	Llama-2	13B	62.0	34.2	72.4	43.2	49.2	52.2
MathCoder	Llama-2	13B	72.6	29.9	76.9	62.3	54.7	59.3
ToRA	Llama-2	13B	72.7	43.0	72.9	45.7	62.6	59.4
MathGenieLM	Llama-2	13B	**80.4**	<u>43.8</u>	<u>83.5</u>	**78.4**	**72.7**	**71.8**
Mammoth	CodeLlama	34B	72.7	43.6	84.3	51.8	65.4	63.6
MathCoder	CodeLlama	34B	81.7	45.2	82.5	65.8	75.9	70.2
ToRA	CodeLlama	34B	80.7	50.8	80.5	50.2	77.9	68.0
MathGenieLM	CodeLlama	34B	**86.2**	<u>51.4</u>	86.9	<u>85.8</u>	78.4	<u>77.7</u>
MathGenieLM	Llemma	34B	<u>84.1</u>	**55.1**	87.4	89.3	82.9	79.8
WizardMath	Llama-2	70B	81.6	22.7	-	-	-	-
Mammoth	Llama-2	70B	76.9	41.8	82.4	51.4	55.6	61.6
MathCoder	Llama-2	70B	83.9	45.1	<u>84.9</u>	<u>77.0</u>	<u>74.4</u>	<u>73.1</u>
ToRA	Llama-2	70B	<u>84.3</u>	<u>49.7</u>	82.7	73.3	72.6	72.5
MathGenieLM	Llama-2	70B	**88.4**	**51.2**	**87.7**	**89.1**	**76.0**	**78.5**
MathGenieLM	Mixtral	8x7B	87.0	53.7	88.9	89.9	81.7	80.2
MathGenieLM	InternLM2	20B	87.7	55.7	87.3	88.5	85.1	80.9

Table 3: Results of MathGenieLM, compared to various open-source and closed-source models on 2 in-domain datasets (GSM8K, MATH), and 3 out-of-domain datasets (SVAMP, Simuleq, Mathematics). The results of closed-source models are from Yue et al. (2023) and Gou et al. (2023).

Model	Base	Voting	k-path	In-Domain		Out-of-Domain			Average
				GSM8K	MATH	SVAMP	Simuleq	Mathematics	
ToRA	Llama-2 70B	✓	50	88.3	56.9	-	-	-	-
MathGenieLM	Llama-2 70B	✗	-	88.4	51.2	87.7	89.1	76.0	78.5
MathGenieLM	Llama-2 70B	✓	10	91.7(+3.3)	63.3(+12.1)	94.0(+6.3)	95.9(+6.8)	87.2(+11.2)	86.4(+7.9)

Table 4: Results of naive majority voting. k-path represents the number of solutions generated for majority voting.

Analysis of different data composition. We analyze the effect of adding and subtracting different parts of our training data to observe the impact of each component. As shown in the upper half of Tab. 5, when only AugGSM8K is added, the performance on GSM8K, SVAMP, and Simuleq improves, while adding AugMATH leads to more notable improvements in MATH and Mathematics. This is consistent with the types of questions in

each dataset: GSM8K, SVAMP, and Simuleq contain grade-school level math word problems with relatively easy calculations, whereas MATH and Mathematics feature more complex math computations. When both AugGSM8K and AugMATH are added, the improvements in the datasets are compounded as well, which shows the effectiveness of our augmented data.

Analysis of different amounts of augmented

Seed	Data Verification	AugGSM8K	AugMATH	In-Domain GSM8K	MATH	Out-of-Domain SVAMP	Simuleq	Mathematics	Average
✓	✗	✗	✗	73.5	41.8	73.1	68.5	66.6	64.7
✓	✓	✗	✗	74.2	42.2	76.5	71.0	65.5	65.9
✓	✓	✓	✗	78.2	42.2	84.3	78.8	69.1	70.5
✓	✓	✗	✓	74.9	43.5	81.6	73.3	73.2	69.3
✓	✓	✓	✓	80.5	45.1	83.3	79.4	71.8	72.0
Seed + Verification				74.2	42.2	76.5	71.0	65.5	65.9
Seed + Verification + $\frac{1}{8}$ (AugGSM8K + AugMATH)				75.6	41.6	80.7	75.3	67.7	68.2
Seed + Verification + $\frac{1}{4}$ (AugGSM8K + AugMATH)				77.9	42.7	82.6	77.0	67.8	69.6
Seed + Verification + $\frac{1}{2}$ (AugGSM8K + AugMATH)				79.2	44.0	82.1	80.2	68.8	70.9
Seed + Verification + (AugGSM8K + AugMATH)				80.5	45.1	83.3	79.4	71.8	72.0

Table 5: Effect of different data composition and amounts of augmented data with Mistral-7B as the base model.

Data	GSM8K	MATH	Average
w/ verification filtering	79.3	43.8	61.6
w/o verificatioin filtering	80.5(+1.2)	45.1(+1.3)	62.8(+1.2)

Table 6: Effect of using or not using code-integrated verification rationales to filter the training data.

data. We analyze the scaling quality of the augmented data we generated by training a model with $\{0, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\}$ times the amount of augmented data. The results, as shown in the bottom half of Tab. 5 and Fig. 3, indicate that, with an increase in the amount of augmented data, the performance on all five datasets consistently improves, with very few exceptions. This demonstrates the high scaling quality of our data.

Analysis of Verification-Based Solution Filtering. We analyze the effectiveness of Verification-Based Solution Filtering by using data before and after filtering with the help of verification to finetune the model. As demonstrated in Tab. 6, finetuning the model with augmented question-solution pairs filtered by verification results in noticeable accuracy increases in both GSM8K and MATH, indicating the superior quality of the augmented data after filtering and the effectiveness of Verification-Based Solution Filtering. Further analysis of the verification ability of M_{code} is shown in Tab. 11 of Appendix D.

Comparison with other question augmentation methods. We compare our method with three other question augmentation methods: Meta-Math (Yu et al., 2023), direct question augmentation without solution, and direct question augmentation with solution. The two direct question augmentation methods both utilize M_{text} as the question augmentation model. The former presents only the seed question to the model during ques-
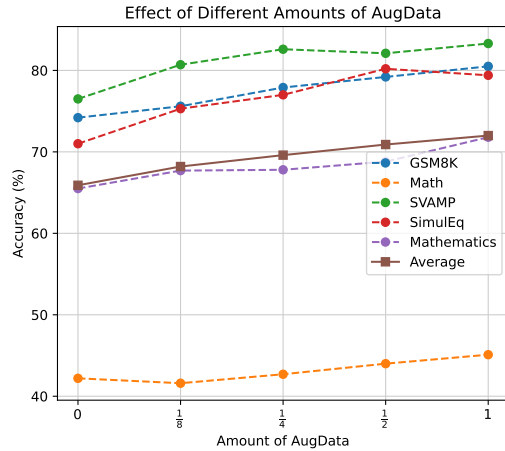

Figure 3: Performance of the Mistral 7B model finetuned with $\{0, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1\}$ times the amount of augmented data.

tion augmentation, while the latter presents both the question and its solution. The results, as shown in Tab. 7, indicate that our augmented solution-to-question back-translation method yields better performance than existing augmentation methods.

3.4 Accuracy of Verified Inference

Our models have the ability to verify its own solutions when presented with prompts as shown in Tab. 10. This represents a mathematical reasoning ability that can be applied during inference.

A simple way to do this is to verify the solutions generated and solve the problem again if the solution is verified to be incorrect. We limit the number of verification to two times. As shown in Tab. 8, applying verification twice consistently enhances accuracy across all five datasets, with notable improvements in the MATH and Mathematics datasets. The average generation (N×) presented in Tab. 8 measures the cost of verified inference, which is 2.3× on average. When compared to 3-

Augmentation Method	In-Domain		Out-of-Domain			Average
	GSM8K	MATH	SVAMP	Simuleq	Mathematics	
MetaMath	79.5	44.6	78.4	79.6	67.9	70.0
Direct question augmentation (w/o sol.)	78.8	43.0	84.0	77.0	70.2	70.6
Direct question augmentation (w/ sol.)	79.2	44.2	83.6	72.0	68.6	69.5
MathGenie (ours)	**80.5**	**45.1**	**83.3**	**79.4**	**71.8**	**72.0**

Table 7: Comparison of different question augmentation methods. **Direct question augmentation (w/o sol.)** presents M_{text} with only the seed questions to generate new questions. **Direct question augmentation (w/ sol.)** presents M_{text} with pairs of question-solution to generate new questions.

		In-Domain		Out-of-Domain			Average
		GSM8K	MATH	SVAMP	Simuleq	Mathematics	
Baseline	Accuracy	88.4	51.2	87.7	89.1	76.0	78.5
	N×	1×	1×	1×	1×	1×	1×
Verify (twice)	Accuracy	88.6	55.8	88.7	91.2	81.1	81.1
	N×	2.1×	2.8×	2.1×	2.1×	2.3×	2.3×
Voting (3-path)	Accuracy	88.6	53.8	92.0	90.9	81.5	81.4
	N×	3×	3×	3×	3×	3×	3×

Table 8: Result of MathGenieLM-Llama-2-70B using verified inference. **Verify (twice)** means that, when testing the model, the solutions are verified and those verified as incorrect are solved again. This process is repeated twice. Each verification or solution is counted as 1 generation, and N× is the average generation count of each question.

path majority voting, verified inference achieves almost identical accuracy but at a significantly reduced cost. The results of more rounds of verification are analyzed in Tab. 12 of Appendix F.

4 Related Works

Large Language Models for Mathematical Reasoning. LLMs have demonstrated remarkable performance in mathematical reasoning tasks. CoT (Wei et al., 2022) enhances LLMs' capability for multistep reasoning. Self-Consistency (Wang et al., 2022) selects the final answer through majority voting. CSV (Zhou et al., 2023) introduces code-based self-verification. Other research efforts focus on pretraining or finetuning LLMs, thereby producing math-specific LLMs, such as Llemma (Azerbayev et al., 2023), WizardMath (Luo et al., 2023), Mammoth (Yue et al., 2023), ToRA (Gou et al., 2023), and MathCoder (Wang et al., 2023). It is noted that code-integrated models (Wang et al., 2023; Gou et al., 2023) have shown superior capabilities over CoT-style models. This paper develops synthetic math problems and solutions using free models to enhance mathematical reasoning.

Instruction-following Datasets for LLMs. Recent studies (Taori et al., 2023; Peng et al., 2023; Mukherjee et al., 2023; Li et al., 2023b) have begun to utilize synthetic instructions generated by LLMs, such as GPT-4 or GPT-3.5, to distill into smaller models. WizardLM (Xu et al., 2023) proposes complex instructions to enrich the seed data for general chat models. This paper, however, focuses on math problem augmentation, particularly for code-integrated math-specific models.

Data Augmentation for Mathematical Reasoning. To upscale the number of math problems, various works (Yu et al., 2023; Liu and Yao, 2024; Li et al., 2023a) directly augment existing problems. Differing from these approaches, our method utilizes information in the solutions through math question back-translation, thereby enhancing the reliability of the augmented questions. We also create code-integrated solutions for the questions and use verification rationales to filter the solutions.

5 Conclusion

In this paper, we propose a coordinated pipeline consisting of *Iterative Solution Augmentation* and *Question Back-translation* to produce large-scale synthetic math questions, and *Verification-Based Solution Filtering* to filter the generated code-integrated solutions. Combined, these three components effectively create new questions and ensure the reliability of the corresponding code-integrated solutions. Experiments show that MathGenieLM achieves superior performance across five math

problem-solving benchmarks and on six different pretrained base models, offering insights into the development of math problem-solving models and providing hope for extension to other reasoning tasks.

Limitations

Our method requires significant GPU resources, involving the full-parameter finetuning of large language models with up to 70B parameters. Therefore, it is crucial for future studies to explore ways to reduce the required resources. Another limitation is that our models cannot process images as input, and thus lack the ability to solve problems that involve images, as discussed in (Lu et al., 2023). Additionally, our models are constrained by a limited context length, having been finetuned with a context length of 4096. These limitations are significant and merit further investigation.

Ethics Statement

Our work, by enhancing the mathematical abilities of language models, can potentially contribute to the cause of math education. Still, our models can output untrue hallucinations, just like any language model. We have utilized various open-source models such as LLaMA-2, CodeLLaMA, Mistral, and Mixtral-8x7B, as well as open-source software such as Hugging Face and PyTorch. We adhere to the policies and licenses of these resources and acknowledge the role they have played in our work.

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*.

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*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

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, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Alex Davies, Petar Velickovic, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter W. Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis, and Pushmeet Kohli. 2021. Advancing mathematics by guiding human intuition with ai. *Nature*, 600:70 – 74.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*.

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. *arXiv preprint arXiv:2103.03874*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *North American Chapter of the Association for Computational Linguistics*.

Chengpeng Li, Zheng Yuan, Guanting Dong, Keming Lu, Jiancan Wu, Chuanqi Tan, Xiang Wang, and Chang Zhou. 2023a. Query and response augmentation cannot help out-of-domain math reasoning generalization. *arXiv preprint arXiv:2310.05506*.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. 2023b. Self-alignment with instruction backtranslation. *arXiv preprint arXiv:2308.06259*.

Haoxiong Liu and Andrew Chi-Chih Yao. 2024. Augmenting math word problems via iterative question composing. *arXiv preprint arXiv:2401.09003*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2023. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*.

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*.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, abs/2303.08774.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*.

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*.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, et al. 2023. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

InternLM Team. 2023. Internlm: A multilingual language model with progressively enhanced capabilities. https://github.com/InternLM/InternLM-techreport.

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. 2023. Mathcoder:

Seamless code integration in llms for enhanced mathematical reasoning. In *International Conference on Learning Representations*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

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.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

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. *arXiv preprint arXiv:2309.12284*.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.

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, and Hongsheng Li. 2023. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. In *International Conference on Learning Representations*.

A Example of Iterative Solution Augmentation and Question Back-translation

Fig. 4 (b) shows an example of three rounds of Iterative Solution Augmentation and Question Back-translation. The seed solution is iteratively augmented, and the augmented solutions are back-translated into new questions. Compared to Fig. 4 (a), where augmentation is conducted directly on the question, Iterative Solution Back-translation demonstrates greater diversity in question phrasing, as the original question is not directly provided to the model.

| | **In-Domain** | | **Out-of-Domain** | | | |
	GSM8K	MATH	SVAMP	Simuleq	Mathematics	Average
w/ iteration	80.5	45.1	83.3	79.4	71.8	72.0
w/o iteration	78.7(-1.8)	45.0(-0.1)	82.9(-0.4)	76.1(-3.3)	70.9(-0.9)	70.7(-1.3)

Table 9: Comparison between with iteration or without iteration when conducting solution augmentation. The models are finetuned on Mistral 7B.

Prompt
Question: {question} **Solution**: {solution} Above is a math problem and its solution. Please use code to verify the solution above.

Table 10: Prompt used for back-translation. {question} is replaced with a math question, while {solution} is replaced with the its code-integrated solution.

B Analysis of Iterative Solution Augmentation

To effectively demonstrate the impact of iteration on enhancing solution quality, we generated an equal number of augmented solutions without employing iteration, by directly augmenting new solutions from the original set. As illustrated in Table 9, the outcomes of the experiment conducted without iteration in solution augmentation were significantly inferior compared to those with iteration. This underscores the beneficial role of iteration in solution augmentation, primarily attributed to its potential to enhance the diversity of the solutions.

C Prompt of Verification

Tab. 10 presents the prompt format used in finetuning and generating code-integrated verification rationales.

D Analysis of Code-integrated Verification

To understand the reason behind the improved quality of the data, we quantify the ability of M_{code} to conduct code-integrated verification by testing it on the solutions generated by M_{code} across the five testing datasets. We utilize these testing datasets because they contain ground truth, which enables us to assess the actual correctness of the solutions.

We define two metrics below to demonstrate the M_{code}'s ability to verify its solutions: Precision and Recall.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{TN}}$$

TP represents the cases in which the verification proves the solution correct and the solution's answer is actually correct. FP represents the cases in which the verification proves the solution correct, yet the solution's answer is actually wrong. TN represents the cases in which the verification proves the solution wrong, yet the solution's answer is actually correct. In short, Precision answers the question, "What proportion of verified TRUE answers are actually correct?", while Recall answers the question, "What proportion of actual correct answers were verified TRUE?". Given these definitions, Precision reflects the reliability of the retained code-integrated solutions, while Recall reflects the efficiency of the filtering step.

Tab. 11 shows that Precision is significantly higher than Accuracy across all datasets, underscoring the effectiveness and generalization capabilities of code-integrated verification.

E Finetuning Details

In this work, we finetune all models using the HuggingFace library. We employ a cosine weight scheduler with a learning rate of $2e^{-5}$, designating the first 50 steps as warm-up steps. All models are optimized using AdamW (Loshchilov and Hutter, 2017) with a batch size of 64. The 70B and 34B models are fine-tuned on 32 NVIDIA A800 80GB GPUs. Mistral-8x7B is fine-tuned on 16 NVIDIA A800 80GB GPUs, while the 7B, 13B, and 20B models are all fine-tuned on 8 NVIDIA A800 80GB GPUs.

F Analysis of Verification Rounds in Verified Inference

In verified inference, we verify the solutions of the test questions and re-solve only those questions whose solutions are verified as incorrect. Consequently, the number of questions that need solving decreases with each round. Theoretically, this process can continue until all questions have solutions

Metrics	GSM8K	MATH	SVAMP	Simuleq	Mathematics	Average
Accuracy	86.4	49.5	86.0	88.1	73.6	76.7
Precision	89.3(+2.9)	64.5(+15.0)	88.3(+2.3)	90.8(+2.7)	79.9(+3.2)	82.6(+5.9)
Recall	98.5	94.7	97.4	98.2	97.1	97.2

Table 11: Accuracy, Precision and Recall of M_{code} with and without verification. Accuracy is the percent of solutions that are correct. Precision is the percent of solutions actually correct among the solutions verified to be correct. Recall is the percent of solutions verified to be correct among the solutions actually correct.

		GSM8K	MATH	SVAMP	Simuleq	Mathematics	Average
Baseline	Accuracy	88.4	51.2	87.7	89.1	76.0	78.5
	N×	1×	1×	1×	1×	1×	1×
Verify (k=1)	Accuracy	88.4	54.2	88.4	90.9	79.2	80.2
	N×	2.0×	2.3×	2.0×	2.0×	2.1×	2.1×
Verify (k=2)	Accuracy	88.6	55.8	88.7	91.2	81.1	81.1
	N×	2.1×	2.8×	2.1×	2.1×	2.3×	2.3×
Verify (k=3)	Accuracy	88.6	56.3	88.8	91.2	81.3	81.2
	N×	2.1×	3.1×	2.1×	2.1×	2.3×	2.3×
Verify (k=4)	Accuracy	88.6	56.3	89.0	91.4	81.4	81.3
	N×	2.1×	3.3×	2.1×	2.1×	2.4×	2.4×
Verify (k=5)	Accuracy	88.6	56.5	88.9	91.4	81.4	81.4
	N×	2.1×	3.4×	2.1×	2.1×	2.4×	2.4×
Verify (k=6)	Accuracy	88.6	56.8	88.9	91.4	81.4	81.4
	N×	2.1×	3.6×	2.1×	2.1×	2.4×	2.5×
Verify (k=7)	Accuracy	88.6	56.8	88.9	91.4	81.4	81.4
	N×	2.1×	3.7×	2.1×	2.1×	2.4×	2.5×
Verify (k=8)	Accuracy	88.6	57.1	88.9	91.4	81.4	81.5
	N×	2.1×	3.7×	2.1×	2.1×	2.4×	2.5×
Verify (k=9)	Accuracy	88.6	57.1	88.9	91.4	81.4	81.5
	N×	2.1×	3.8×	2.1×	2.1×	2.4×	2.5×

Table 12: Results of MathGenieLM-Llama-2-70B with varying numbers of verification rounds during inference. Here, "k" represents the maximum number of verification rounds.

verified as correct. However, in practice, verifying too many rounds can lead to additional costs without any improvement in accuracy, as some questions may be beyond the model's problem-solving and verification capabilities. To determine the trade-off between cost and accuracy, we increased the number of verification rounds to 9. The results are shown in Tab. 12. As can be seen, the increase in average accuracy becomes small after 2 rounds of verification.

G Examples of Code-Integrated Verification Rationales

Two examples of code-integrated verification rationales are presented in Tab. 13 and Tab. 14. In Tab. 13, the solution is verified as correct by using the answer to calculate the condition and comparing it with the actual condition. In Tab. 14, the solution is verified as incorrect by solving the question through an alternative method and comparing the answers.
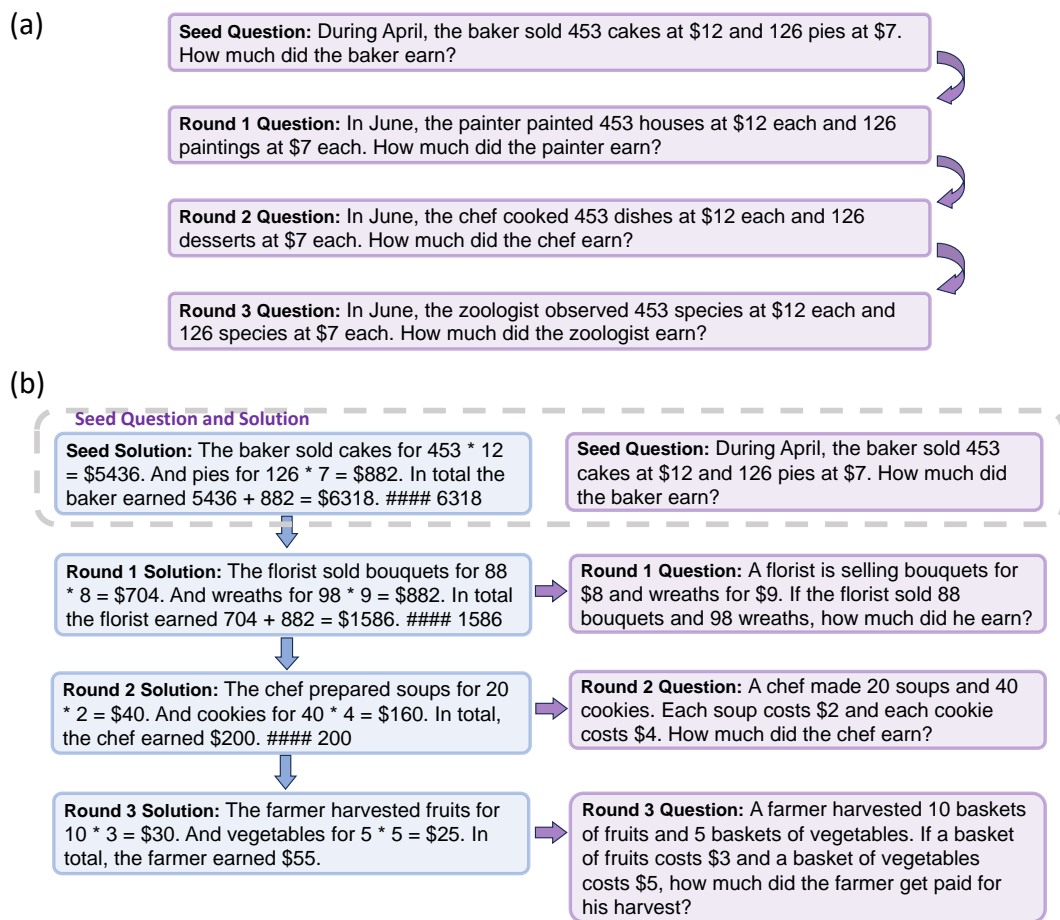
(a)

Seed Question: During April, the baker sold 453 cakes at $12 and 126 pies at $7. How much did the baker earn?

Round 1 Question: In June, the painter painted 453 houses at $12 each and 126 paintings at $7 each. How much did the painter earn?

Round 2 Question: In June, the chef cooked 453 dishes at $12 each and 126 desserts at $7 each. How much did the chef earn?

Round 3 Question: In June, the zoologist observed 453 species at $12 each and 126 species at $7 each. How much did the zoologist earn?

(b)

Seed Question and Solution

Seed Solution: The baker sold cakes for 453 * 12 = $5436. And pies for 126 * 7 = $882. In total the baker earned 5436 + 882 = $6318. #### 6318

Seed Question: During April, the baker sold 453 cakes at $12 and 126 pies at $7. How much did the baker earn?

Round 1 Solution: The florist sold bouquets for 88 * 8 = $704. And wreaths for 98 * 9 = $882. In total the florist earned 704 + 882 = $1586. #### 1586

Round 1 Question: A florist is selling bouquets for $8 and wreaths for $9. If the florist sold 88 bouquets and 98 wreaths, how much did he earn?

Round 2 Solution: The chef prepared soups for 20 * 2 = $40. And cookies for 40 * 4 = $160. In total, the chef earned $200. #### 200

Round 2 Question: A chef made 20 soups and 40 cookies. Each soup costs $2 and each cookie costs $4. How much did the chef earn?

Round 3 Solution: The farmer harvested fruits for 10 * 3 = $30. And vegetables for 5 * 5 = $25. In total, the farmer earned $55.

Round 3 Question: A farmer harvested 10 baskets of fruits and 5 baskets of vegetables. If a basket of fruits costs $3 and a basket of vegetables costs $5, how much did the farmer get paid for his harvest?

Figure 4: An example of three iterations of (a) Direct Question Augmentation and (b) Iterative Solution Augmentation and Question Back-translation.

Input:

Question:

Express $0.\overline{54}$ as a fraction in lowest terms.

Solution:

To express the repeating decimal $0.\overline{54}$ as a fraction in lowest terms, we can use algebraic manipulation.

Let $x = 0.\overline{54}$.

Step 1: Multiply x by 100 to shift the decimal point.

......

Therefore, $0.\overline{54}$ can be expressed as $\frac{6}{11}$ in lowest terms.

Above is a math problem and its solution. Please use code to verify the solution above.

Output:

Now, let's verify the answer by converting $\frac{6}{11}$ back to a decimal.

```
# Convert the fraction 6/11 to a decimal
decimal_value = float(simplified_fraction)

decimal_value
```

Result:

0.5454545454545454

The decimal representation of $\frac{6}{11}$ is approximately 0.5454545454545454, which matches the repeating decimal $0.\overline{54}$

Thus, our answer is verified and correct.

Table 13: An example of code-integrated verification rationale in which the solution is verified as correct.

Input:

Question:

What is $2121_3 - 212_3$? Express your answer in base 3.

Solution:

Alright, let's solve this step by step.

To subtract these numbers, we'll start from the rightmost digit (least significant) and move to the left, just like we do in base 10 subtraction.

......

The result of $2121_3 - 212_3$ is $\boxed{222_3}$.

Above is a math problem and its solution. Please use code to verify the solution above.

Output:

Now, let's verify our answer by converting both numbers to base 10, performing the subtraction, and then converting the result back to base 3 to see if it matches our answer.

```python
# Convert the numbers from base 3 to base 10
def base3_to_base10(num_str):
    num_str = num_str[::-1]  # reverse the string for easy processing
    base10_value = 0
    for i, digit in enumerate(num_str):
        base10_value += int(digit) * (3 ** i)
    ......

# Convert 2121_3 and 212_3 to base 10
num1_base10 = base3_to_base10("2121")
num2_base10 = base3_to_base10("212")

# Perform the subtraction in base 10
result_base10 = num1_base10 - num2_base10

# Convert the result back to base 3
result_base3 = base10_to_base3(result_base10)

result_base3
```

Result:

1202

It seems there was an error in our calculations. The correct result for $2121_3 - 212_3$ is 1202_3, not 222_3. I apologize for the oversight.

Table 14: An example of code-integrated verification rationale in which the solution is verified as wrong.