RV-Syn: Rational and Verifiable Mathematical Reasoning Data Synthesis Based on Structured Function Library

Jiapeng Wang[1,3*], **Jinhao Jiang**[1,3*],
Zhiqiang Zhang[2], **Jun Zhou**[2], **Wayne Xin Zhao**[1,3†],
[1]Gaoling School of Artificial Intelligence, Renmin University of China [2]Ant Group
[3]Beijing Key Laboratory of Research on Large Models and Intelligent Governance
wangjp1010@ruc.edu.cn, batmanfly@gmail.com

Abstract

The advancement of reasoning capabilities in Large Language Models (LLMs) requires substantial amounts of high-quality reasoning data, particularly in mathematics. Existing data synthesis methods, such as data augmentation from annotated training sets or direct question generation based on relevant knowledge points and documents, have expanded datasets but face challenges in mastering the internal logic of the problem during generation and ensuring the verifiability of the solutions. To address these issues, we propose **RV-Syn**, a novel **R**ational and **V**erifiable mathematical **Syn**thesis approach. RV-Syn first constructs a structured library of mathematical operations and then composes them into executable computational graphs, which serve as verifiable solution blueprints. These graphs are subsequently back-translated into complex problems, enabling solution-guided, logic-aware problem generation while inherently ensuring the verifiability of the solving process. Experimental results show RV-Syn surpasses existing synthesis methods, including those involving human-crafted problems. Our method achieves a 6.3% performance gain over the previous state-of-the-art synthetic data on LLaMA-3-8B and demonstrates superior data efficiency, outperforming others with only half the training data (50k vs. 100k), enabling a more scalable and robust reasoning dataset generation framework.

1 Introduction

The development of advanced reasoning Large Language Models (LLMs) (Zhao et al., 2023; OpenAI, 2024) has markedly improved their ability to address complex tasks across domains such as mathematics, science, and coding. This highlights the importance of synthesizing complex reasoning data to drive further advancements, given the limited availability of high-quality annotated instructions (Shah et al., 2024; Yuan et al., 2023).

To address this scarcity, researchers have explored various synthesizing methods, particularly in the mathematics domain. The mainstream methods involve data augmentation based on existing annotated training sets, such as GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021), ranging from self-evolving instructions (Xu et al., 2024a; Zeng et al., 2024) and question paraphrasing (Yu et al., 2024), to solution augmentation (Lu et al., 2024). However, these methods are limited by the available training data, constraining the synthesis diversity (Li et al., 2024c, 2023). To enhance diversity, recent approaches enable LLMs to generate a large scale of questions from various mathematics-related sources, including web pages (Yue et al., 2024) and knowledge points (Tang et al., 2024) from web corpora or textbooks.

However, as shown in Figure 1, these direct problem generation methods suffer from a fundamental limitation. We argue this stems from the auto-regressive nature of LLMs, which leads to a lack of forward-looking planning during generation. When composing a math problem, a model might introduce entities and quantities without a coherent plan, frequently resulting in internal contradictions, unsolvable scenarios, or trivial questions, particularly in complex, multi-step reasoning problems. Furthermore, this approach makes it difficult to validate the correctness of the generated solutions, often compromising the quality and reliability of the training data.

To address the aforementioned challenges, we draw inspiration from how human educators create problems. Just as one cannot *directly write down an Olympic-level math problem without deep consideration and careful curation of the underlying problem-solving process*, the same holds true for models. Specifically, human educators obtain abstract independent computation goals from histori-

[*] Equal contribution.
[†] Corresponding author.

Findings of the Association for Computational Linguistics: EACL 2026, pages 1812–1827

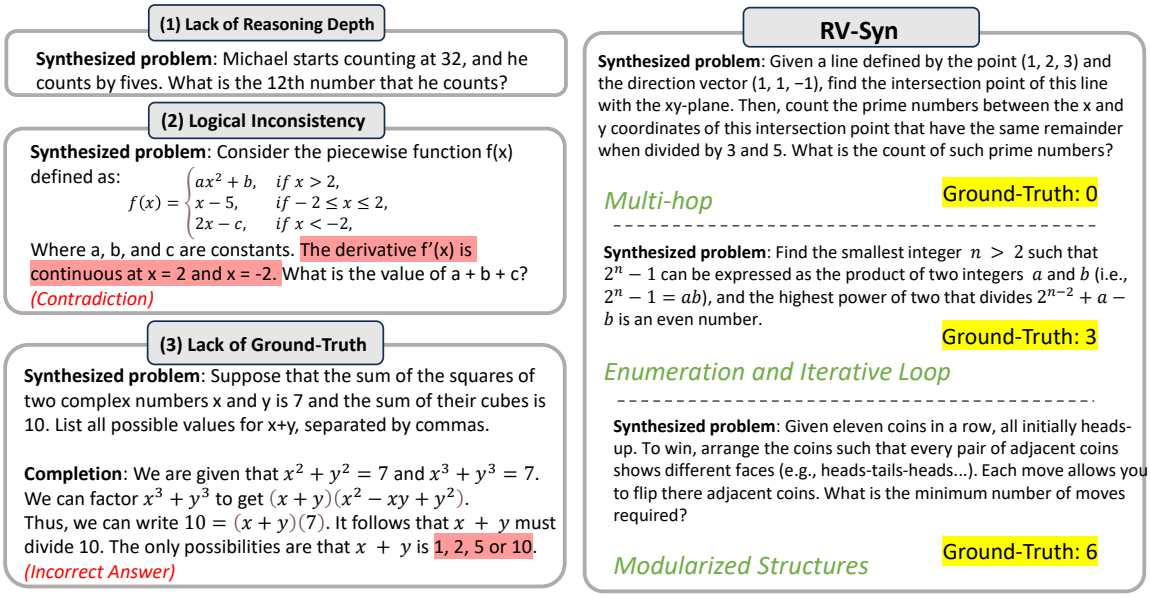

Figure 1: **(Left)** Illustration of the limitations of existing synthesis methods; **(Right)** Examples of data synthesized by RV-Syn, which naturally support sophisticated reasoning structures and provide ground-truth answers.

cal experience, such as "projecting a vector onto a plane", and "solving for the smallest prime number that satisfies specific conditions". Then, when designing a new question, they would combine these computation goals to obtain a new solution or introduce them into an existing solution. Finally, they derive the final problem based on this elaborated solution process. This strategy ensures the development of problems with coherent and logically consistent solution processes, thereby minimizing the internal inconsistencies of the final problems.

Inspired by this, we propose **RV-Syn**, a novel framework for mathematical problem synthesis that emphasizes enhanced rationale and verifiability. Instead of direct generation, RV-Syn first synthesizes a structured computational graph. This graph serves as a formal blueprint, explicitly defining the problem's reasoning structure and guaranteeing a valid, executable solution path. Subsequently, this logically consistent blueprint is back-translated into a natural language problem. By generating questions through solution guidance, RV-Syn effectively provides the rich semantics and control flows required for complex reasoning. As shown in Figure 1, this enables the synthesis of diverse and sophisticated reasoning structures—including multi-hop reasoning, iterative loops, and modular structures—while naturally providing verifiable ground-truth labels.

We conduct extensive experiments to evaluate the proposed RV-Syn method using various LLMs. The results demonstrate that our method achieves superior performance across five benchmarks compared to existing approaches, including those involving human-designed problems (*e.g.,* Numina-Math). Notably, RV-Syn outperforms the previous state-of-the-art method while utilizing only half the data, leading to a more efficient scaling curve.

2 Related Work

Mathematical Reasoning. Researchers have proposed various approaches to enhance the mathematical reasoning capabilities, including methods applied in training or inference stages. During the training stage, existing methods aim to enhance the LLMs from the aspects of pre-training and post-training. Specifically, some studies (Azerbayev et al., 2024; Shao et al., 2024; Chen et al., 2024a) involve collecting extensive math-related corpora and enhancing the foundational mathematical capabilities of LLMs through continual pre-training. In contrast, other studies (Chen et al., 2024c; Tong et al., 2024; Huang et al., 2024b; Toshniwal et al., 2024) focus on synthesizing a substantial amount of high-quality math-related instructional data, further refining the problem-solving abilities of LLMs through post-training. During the inference stage, there are two primary prompting approaches, including Chain-of-Thought (CoT) (Wei et al., 2022) and Program-of-Thought (PoT) (Chen et al., 2023). Specifically, some studies (Yao et al., 2023; Shinn et al., 2023; Zhou et al., 2023) utilize CoT to elicit

LLMs' inherent reasoning ability with a step-by-step reasoning process. Furthermore, other studies (Gou et al., 2024; Wang et al., 2024; Chen et al., 2024b; Wen et al., 2024) incorporate PoT to allow LLMs to utilize external computers during reasoning. In this work, we focus on the training stage and try to synthesize instruction data by leveraging the strengths of both the CoT and PoT methods.

Data Synthesis for Math Instruction. Existing research has proposed various synthesizing approaches: Firstly, existing studies focus on synthesizing new mathematical problems based on few-shot problems (Li et al., 2024a), math-related documents (Yue et al., 2024), key knowledge points (Huang et al., 2024a), or self-evolution (Luo et al., 2023). Furthermore, with the improvement of LLMs' capabilities, subsequent studies (Zhou et al., 2024; Ding et al., 2024) utilizingexplore the utilization of open-source LLMs (*i.e.,* DeepSeek-Math and Qwen-Math) to scale the number of synthesized problems. Despite their effectiveness, direct problem generation suffers from a lack of fine-grained control over the problem's internal logic, leading to lower-quality problems. Besides, these methods frequently lack mechanisms for validating the correctness of generated answers, which may introduce errors and affect the final performance. In contrast, our method starts from the internal logic of problem-solving, rather than directly synthesizing problems. This provides more refined logical consistency and improves problem quality, while also enabling effective correctness control.

3 Approach

In this section, we provide a comprehensive introduction to our proposed RV-Syn method. Its essence involves developing a comprehensive set of mathematical operations and skills. These skills are then systematically combined to construct the solution process, ultimately enabling the generation of complex problems through back-translation. The complete pipeline comprises three core stages: (1) decomposing existing seed problems to generate a set of computational graphs (Section 3.1), (2) extracting functions from these computational graphs and constructing a graph-format function library (Section 3.2), and (3) combining selected functions to reconstruct new computational graphs, which are subsequently back-translated into problems (Section 3.3). The overall data flow of our method is illustrated in Figure 2.

3.1 Problem Decomposition

We first represent the solutions of existing mathematical problems with sequences of functions into a standardized format shown in Appendix C leveraging LLMs. Each function represents a specific mathematical skill with defined parameters, forming an executable *computational graph* that captures the data flow of the problem-solving process (examples shown in Figure 2). To ensure the correctness of the extracted solutions and their associated functions, we filter out erroneous cases by executing the solution code using a Python interpreter and comparing its output with the ground-truth answer provided in the corresponding Chain-of-Thought annotation. This results in a large-scale, high-quality set of computational graphs derived from real-world problems.

3.2 Graph-based Function Library Construction

After decomposing the problem, we extract functions from the computational graphs. To further enhance the quality of the function library, we employ model-based labeling. We utilize LLMs to validate the correctness of the functions and identify their corresponding mathematical topics, eliminating any functions that contain errors. The prompts are shown in Appendix A. To describe the relationships between these functions, each function is represented as a node, and they are organized into a graph-based function library. Specifically, the construction process involves two key steps: structural and semantic-aware function merging, followed by co-occurrence and topic-based node connecting.

3.2.1 Structural and Semantic-aware Function Merging

As previously discussed, each extracted function primarily encapsulates a specific mathematical operation or skill. Consequently, different functions may exhibit similarities in semantics (*i.e.,* possessing similar skills) or structure (*i.e.,* sharing similar computational logic). To preserve the diversity of function expressions while minimizing redundancy within the final function library, we merge functions that are similar in either structure or semantic aspects into one node. Therefore, the final node is a set that contains one or more similar nodes. In the subsequent description, we will take this setting as the default.

Specifically, we examine the similarity between two functions by analyzing both structure and se-

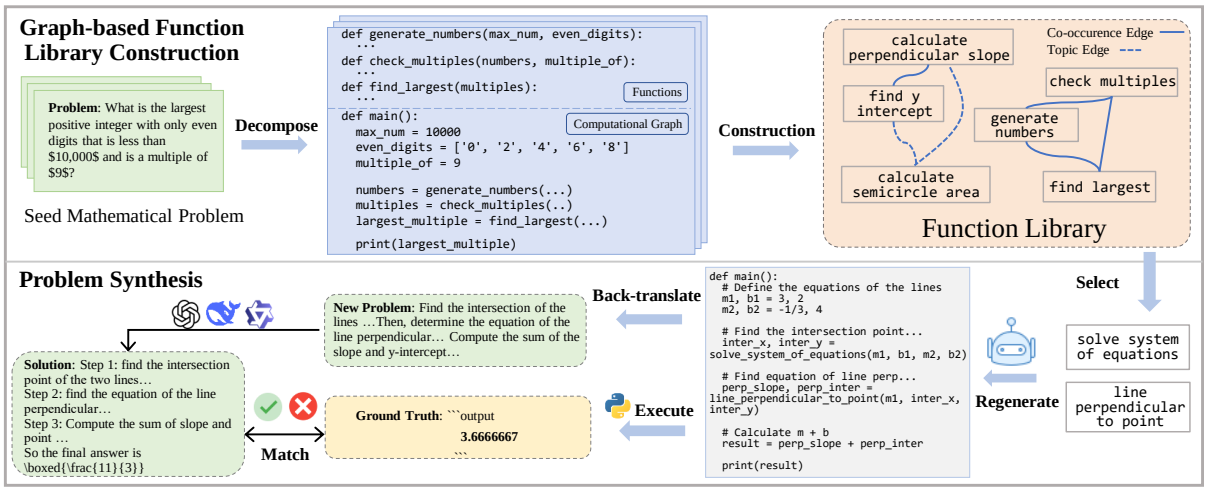

Figure 2: The pipeline of our proposed method. The upper part demonstrates the construction of the operation function library. The lower part illustrates the synthesis process.

mantic aspects. For the structure, we focus on the computational operations without considering specific variable names. For instance, expressions such as "$(a+b)/c$" and "$(m+n)/d$" are considered equivalent. To accomplish this, we extract the function body and parse it using Abstract Syntax Trees (ASTs). For the semantic, we compute hash values for the function's docstrings. If the ASTs or hash values of two functions are identical, we merge these functions into a single node. We show an example of two similar functions in Table 12 at Appendix H. This approach allows similar functions (e.g., matrix multiplication and matrix projection) to preserve their distinct mathematical meanings while minimizing redundancy in the final function library, offering advantages over direct deduplication. Our structure-based method using ASTs proves highly effective in practice, successfully identifying and merging approximately 28% of redundant functions from the initial library.

3.2.2 Co-occurrence and Topic-based Node Connecting

After obtaining the final set of nodes through function merging, we further establish connections between relevant nodes to construct the final graph. We define two types of edges: co-occurrence-based edges and topic-consistent edges.

Specifically, for any two given nodes (*i.e.,* two sets of functions), we first check whether there exists at least one pair of functions—one from each set—that co-occurred in the same computational graph. If so, we add a co-occurrence-based edge between these two nodes. If not, we then check whether there is at least one pair of functions sharing the same topic. If this condition is met, we add a topic-consistent edge. If neither condition is satisfied, no edge is added between the nodes.

In this way, we finally obtain an incompletely connected graph that contains multiple types of edges, which depicts the relationships between nodes. This serves as the basis for the subsequent recombination of functions.

3.3 Problem Synthesis

Based on the constructed graph-format function library, we are able to synthesize complex problems in a solution-guided approach. This process consists of two primary steps: first sampling various functions to regenerate the computational graph, then synthesizing the final complex problems.

3.3.1 Computational Graph Regeneration

To balance the reasonableness and novelty of the computational graphs, we design three sampling strategies: first, selecting nodes connected by co-occurrence-based edges; second, selecting nodes connected by topic-based edges; and third, selecting unconnected nodes. During the sampling process, nodes are randomly selected using these strategies. Since each node represents a set of functions, we further randomly select one function per node. Subsequently, we generate a computational graph based on these selected functions as detailed in Appendix A. This approach offers several advantages: Firstly, by combining functions, the generated computational graph can comprehensively cover mathematical skills while minimizing redun-

dancy. Secondly, the sampling strategy effectively balances high-frequency combinations of mathematical skills (i.e., connected by edges) with long-tail combinations (i.e., without connected edges). In our setup, we randomly select from the three strategies when synthesizing each data sample.

3.3.2 Problem Back-translation and Verification

The final stage of our pipeline generates a complete (problem, solution) pair and ensures its correctness, highlighting the core verifiability of our framework. First, we execute all newly generated computational graphs using an interpreter. Graphs that fail to execute due to errors are discarded. This step serves two key purposes: it validates the feasibility of the synthesized reasoning process and yields a reliable, ground-truth final answer from successful execution. With a verified computational graph and its ground-truth answer, we then back-translate the graph into a coherent natural-language math problem and generate its Chain-of-Thought (CoT) solution. Unlike direct generation methods, which struggle to verify the correctness of their outputs, our framework can compare the final answer derived from the LLM-generated CoT solution against the ground-truth answer obtained from executing the computational graph. Any (problem, solution) pair with mismatched answers is automatically discarded. This execution-based filtering acts as a precise and automatic quality control mechanism, effectively eliminating instances of flawed reasoning or incorrect calculations in the final LLM-generated solutions. We provide more statistics of the synthesis pipeline in Appendix E.

3.4 Comparison to Previous Work

We give a comparison in Table 1. The first line of research, including MetaMath (Yu et al., 2024), Orca-Math (Mitra et al., 2024), WizardMath (Luo et al., 2023), MuggleMath (Li et al., 2024b), MathGenie (Lu et al., 2024), primarily focuses on augmenting existing problems or solutions, which leads to new instructions that are too similar to the original ones, thus limiting diversity. Another line of research, represented by Mammoth2 (Yue et al., 2024), Jiuzhang3.0 (Zhou et al., 2024), KP-Math (Huang et al., 2024a), and ScaleQuest (Ding et al., 2024), synthesizes brand new mathematical problems based on math-related documents, knowledge points, or from scratch, offering greater diversity and scalability. However, many of these methods overlook the internal logical processes when directly synthesizing problems. Methods like ControlMath (Chen et al., 2024c) and PromptCoT (Zhao et al., 2025) attempt to enhance the rationality of synthesis by leveraging equation generators or injecting intermediate thought processes. Nevertheless, they still lack mechanisms to verify the correctness of the synthesized data. Our RV-Syn method approaches problem generation from the perspective of internal problem-solving logic rather than directly producing problem statements without methodical consideration, thereby enhancing problem quality. Additionally, through executable computation graphs, our method enables automatic verification of solution correctness.

Method	Category	Open-source Problem Crafter	Rational Problem Synthesize	Automatic Solution Verification
MetaMath	Aug	✗	✗	✗
Orca-Math	Aug	✗	✗	✗
WizardMath	Aug	✗	✗	✗
MathGenie	Aug	✓	✓	✗
Mammoth2	New	✓	✗	✗
Jiuzhang3.0	New	✓	✗	✗
ControlMath	New	✗	✓	✗
MathScale	New	✗	✗	✗
PromptCoT	New	✓	✓	✗
ScaleQuest	New	✓	✗	✗
RV-Syn	New	✓	✓	✓

Table 1: Comparison of Different Methods. *Category* specifies whether the method augments existing data ("Aug") or synthesizes new questions ("New"). *Open-source Problem Crafter* indicates whether the method utilizes open-source models (✓) or proprietary models like GPT-4 (✗) for problem generation. *Rational Problem Synthesize* denotes whether the synthesized problems incorporate internal problem-solving logic. *Automatic Solution Verification* denotes the method's ability to automatically verify the correctness of synthesized data.

4 Experiments

4.1 Experimental Setup

Compared Baselines. We compare our method with previous problem synthesis methods with publicly available datasets, including: (1) MetaMath (Yu et al., 2024) introduces several question bootstrapping techniques; (2) Orca-Math (Mitra et al., 2024) augments existing datasets using an Agent-Instruct method; (3) MathScale (Tang et al., 2024) uses topic and knowledge-point graphs to prompt new problem synthesis; (4) Mammoth2 (Yue et al., 2024) extracts QA pairs from webpages; (5) Jiuzhang3.0 (Zhou et al., 2024) uses math-related seed data to synthesize new QA pairs; (6) PromptCoT (Zhao et al., 2025) generate complex problems with intermediate thought; (7) Scale-

Method	Problem Crafter	MATH-500	GSM8K	GSM-Hard	College Math	Olympiad Bench	Avg (Rel. Imp.)
Models based on LLaMA-3-8B-Instruct							
Official Model	-	28.4	75.1	35.6	21.2	7.3	33.50
MetaMath	ChatGPT	42.4	83.1	36.2	22.3	10.1	38.82 (+15.9%)
Orca-Math	GPT-4	29.4	**86.6**	42.7	19.8	11.1	37.92 (+13.2%)
Mammoth2	72B Model	42.2	78.6	40.3	**33.0**	13.6	41.54 (+24.0%)
Jiuzhang3.0	7B Math Model	43.4	81.3	40.5	29.6	15.9	42.14 (+25.8%)
MathScale	ChatGPT	43.4	81.3	38.4	32.6	13.6	41.86 (+25.0%)
PromptCoT	72B Model	43.4	78.2	41.5	25.8	15.7	40.92 (+22.1%)
ScaleQuest	7B Math Model	45.8	83.6	41.9	29.4	13.3	42.80 (+27.8%)
RV-Syn	7B / 72B model	**50.4**	82.6	**44.5**	30.7	**16.4**	**44.92 (+34.1%)**
Models based on Qwen2.5-7B-Instruct							
Official Model	-	74.0	86.8	59.9	44.1	36.0	60.16
MetaMath	ChatGPT	74.0	90.9	67.6	45.5	35.0	62.60 (+4.1%)
Orca-Math	GPT-4	74.2	**92.0**	67.4	45.3	35.4	62.86 (+4.5%)
Mammoth2	72B Model	75.0	90.2	68.2	45.9	34.7	62.80 (+4.4%)
Jiuzhang3.0	7B Math Model	74.8	91.3	67.4	45.8	34.7	62.80 (+4.4%)
MathScale	ChatGPT	74.0	91.0	67.6	45.3	**36.4**	62.98 (+4.5%)
PromptCoT	72B Model	75.8	90.5	67.7	45.8	34.8	62.92 (+4.6%)
ScaleQuest	7B Math Model	75.0	91.4	67.6	45.9	34.4	62.86 (+4.5%)
RV-Syn	7B / 72B model	**76.8**	91.3	**69.6**	**46.1**	**36.4**	**64.04 (+6.4%)**
Models based on Phi-3-mini							
Official Model	-	43.0	73.3	54.1	35.7	15.7	44.40
MetaMath	ChatGPT	55.8	89.8	64.5	39.6	20.7	54.08 (+21.8%)
Orca-Math	GPT-4	55.2	**90.4**	**65.3**	40.0	19.6	54.10 (+21.9%)
Mammoth2	72B Model	59.0	87.7	62.9	41.6	21.3	54.50 (+22.7%)
Jiuzhang3.0	7B Math Model	58.2	88.9	63.8	41.5	19.6	54.40 (+21.8%)
MathScale	ChatGPT	58.2	89.5	63.5	40.3	20.3	54.36 (+22.5%)
PromptCoT	72B Model	58.2	87.4	63.5	39.6	21.3	54.00 (+21.6%)
ScaleQuest	7B Math Model	59.0	88.9	63.8	41.2	21.8	54.94 (+23.7%)
RV-Syn	7B / 72B model	**59.8**	88.8	65.0	**42.2**	**22.4**	**55.64 (+25.3%)**

Table 2: After training on 50k data, the evaluation results of our method compared with various synthesis approaches across five benchmarks. The best ones among LLMs with the same backbone model are marked in bold.

Quest (Ding et al., 2024) trains a problem generation model similar to Magpie (Xu et al., 2024b).

Tuning Data. Directly comparing the performance of models from previous works introduces potential unfair factors, as newer models may leverage more advanced models for backbone and answer generation (see Appendix F for details). To ensure a fair comparison, we use the same model, Qwen-2.5-Math-7B-Instruct (Yang et al., 2024) to generate answers for problems synthesized by each method and train the same backbone model. By controlling for other variables, we can focus on comparing the quality of the synthesized problems.

Evaluation and Metrics. We assess the models' performance on MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024), GSM8K (Cobbe et al., 2021), GSM-Hard (Gao et al., 2023), College Math (Tang et al., 2024) and OlympiadBench (He et al., 2024). The generated outputs are all in the form of natural language Chain-of-Thought (CoT) (Wei et al., 2022) through greedy decoding, and we report zero-shot pass@1 accuracy.

Implementation Details. We extract seed data from the high-quality Numina-Math dataset (Li et al., 2024d) to collect function nodes and fine-tuning data. The problem decomposition is performed using Qwen2.5-72B-Instruct. We then use the fine-tuned Qwen2.5-7B-Instruct to generate computational graphs with the strategy described in 3.3.1, followed by problem back-translation using Qwen2.5-72B-Instruct.

4.2 Main Results

As shown in Table 2, methods focusing on synthesizing diverse new problems (*e.g.,* Mammoth2, Jiuzhang3.0, ScaleQuest) outperform approaches that augment existing data (*e.g.,* MetaMath, Orca-Math). This performance gap may be attributed to augmentation-based methods leading to new instructions that are too similar to the original ones, emphasizing the crucial role of data diversity. Our approach outperforms traditional data synthesis methods without relying on proprietary models like

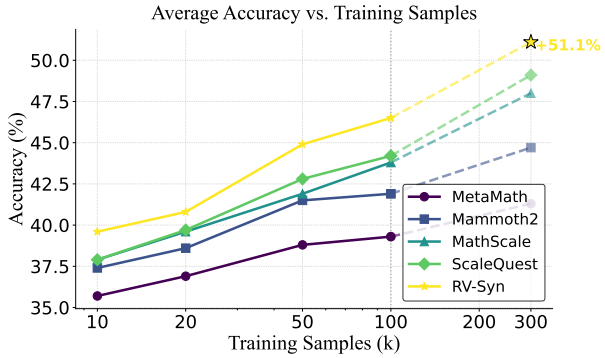

Figure 3: The performance of the model as the training data size increases. The "Average" refers to the mean performance across five datasets: MATH-500, GSM8K, GSM-Hard, College Math, and OlympiadBench.

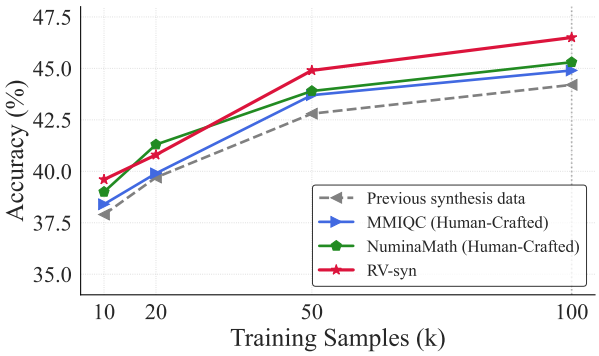

Figure 4: Comparison between RV-Syn and human-crafted data. Numina-Math serves as the seed data for our synthetic dataset. Previous state-of-the-art synthetic data methods are indicated by gray lines.

GPT-4. For example, on LLaMA-3-8B-Instruct, our method outperforms the previous state-of-the-art by 6.3%. Notably, this advantage is particularly evident on more challenging or computation-intensive datasets such as MATH-500, GSM-Hard, and OlympiadBench, further emphasizing the benefits of rational synthesis. We present experiments on distilling Long-CoT reasoning in Appendix 4.7.

4.3 Scalability Study

In this section, we study the scalability of our method. We compare our method with previous approaches by training on scaling datasets up to 300k using LLaMA-3-8B-Instruct (AI@Meta, 2024). The results in Figure 3 demonstrate a strong scaling curve, with performance consistently improving as the dataset size increases. Our method consistently outperforms other methods, and notably, achieves better performance with only half the dataset size (50k vs. 100k). This advantage is particularly pronounced on challenging or computation-intensive datasets, demonstrating the superiority of our method for enhancing advanced mathematical reasoning capabilities.

4.4 Comparison with Human-Crafted Data

To rigorously benchmark the quality of our synthesized data, we compare RV-Syn against datasets that contain a significant portion of high-quality, human-crafted problems. We include two such strong baselines in our analysis: MMIQC (Liu et al., 2024), which contains QA pairs from the Mathematics Stack Exchange, and Numina-Math (Li et al., 2024d), a large collection of both real-world human-crafted and synthesized math problems, which also serves as the source of our initial seed data. As illustrated in Figure 4, human-crafted datasets such as MMIQC and Numina-Math consistently outperform most synthetic methods, reflecting the superior quality of human-authored data over synthetic data, though they face significant scalability challenges. In contrast, RV-Syn demonstrates a clear competitive advantage: at nearly every data scale, it achieves the highest average performance, even surpassing Numina-Math, the very dataset used as its seed. This result validates that our synthesis process does not merely replicate existing examples, but instead generates novel, high-quality instructions that are more effective for model training than the original seed data itself. Consequently, RV-Syn successfully bridges the gap between scalability and quality, producing synthetic data that rivals human expertise without sacrificing performance. We provide further discussion on data diversity in Appendix D.

4.5 Analysis of Data Efficiency

Higher complexity through Rational Synthesis. First, we posit that the solution-first, rational synthesis approach leads to higher-quality and more complex problems. A higher-quality dataset can provide richer supervision signals per example, thus improving training efficiency. To quantify this, we analyze the difficulty and efficacy of different datasets using three key metrics, following PromptCoT (Zhao et al., 2025). First, accuracy serves as an indicator of problem difficulty. Lower performance signifies a higher level of difficulty. For datasets without ground-truth labels, we adopt the approach detailed in PromptCoT (Zhao et al., 2025) by leveraging a more powerful model, Qwen2.5-Math-72B-Instruct, to generate reference answers,

applying self-consistency with 8 rollouts to enhance reliability. Second, the average number of reasoning tokens provides insights into the complexity of the reasoning process. A larger average number of tokens suggests that the model necessitates more extensive reasoning steps. Finally, the average accuracy on the benchmark after tuning reflects the dataset's contribution to improving model performance. Larger improvements in benchmark accuracy observed after fine-tuning suggest that the respective dataset is more effective in enhancing the model's capabilities. Experimental results in Table 3 show a strong consistency among these three metrics, and our method achieves the best results across all three difficulty evaluation metrics. Specifically, problems synthesized by RV-Syn are the most difficult (lowest accuracy at 55.6%), require the longest reasoning chains (highest token count), and yield the largest performance improvement after fine-tuning (+34.1%). This demonstrates that our solution-first, rational synthesis approach successfully generates more challenging and effective training instances compared to other methods.

Dataset	Accuracy (↓)	Avg. Reasoning Tokens (↑)	Avg. Benchmark Accuracy (↑)
MetaMath	97.2	1,339	38.82 (+15.9%)
Orca-Math	86.3	919	37.92 (+13.2%)
MathScale	82.0	2,565	41.86 (+25.0%)
Mammoth2	70.6	3,272	41.54 (+24.0%)
ScaleQuest	75.8	2,591	42.80 (+27.8%)
Numina-Math	72.9	3,858	43.90 (+31.0%)
RV-Syn	**55.6**	**4,431**	**44.92 (+34.1%)**

Table 3: Difficulty and efficacy evaluation for different datasets. **Accuracy:** Performance of Qwen2.5-Math-7B-Instruct on the problems in different datasets. **Avg. Reasoning Tokens:** Average number of tokens in reasoning processes generated by DeepSeek-R1-Distill-Qwen-7B when processing the problems. **Avg. Benchmark Accuracy:** Performance of LLaMA-3-8B-Instruct after fine-tuning on different datasets.

Higher Quality through Verifiability. In this part, we investigate the verifiable characteristics of our method to ensure data correctness. We sample 1,000 data points from the training data of each method, evaluate them and analyze the error rates for both problems and solutions (see Appendix B for details). As shown in Table 4, firstly, the problem error rate reflects the quality of the synthesized problems. It can be observed that problems generated by humans exhibit a lower error rate, while directly synthesized problems contain more errors. Our method approaches the level of control that humans exert in problem design. Secondly, the solution error rate reflects the extent to which each method controls the correctness of training data. Since other methods cannot naturally obtain the ground truth, their solution error rates are relatively high. Further quality control requires heavy reliance on powerful models for post-filtering, annotation, voting, or scoring, which incurs significant additional costs. In contrast, our method inherently provides the ground truth and enables rule-based matching, ensuring an extremely low error rate. We then manually inspect our data labeled as erroneous and find that most solutions have correct final answers, with only minor errors in intermediate steps.

Method	Problem Error Rate (%) ↓	Solution Error Rate (%) ↓
Mammoth2	0.9	10.6
Jiuzhang3.0	2.4	6.5
MathScale	1.6	4.9
ScaleQuest	5.6	8.0
Numina-Math	**0.6**	5.5
RV-Syn	0.9	**1.4**

Table 4: Error rate analysis across different methods. The problem error refers to issues such as inconsistency or conflict in problem conditions and statements; the solution error indicates computational mistakes or misinterpretation of the problem in the solution.

4.6 Ablation Study

To provide a comprehensive understanding of RV-Syn, we conduct ablation studies to analyze the impact of our graph sampling strategies and the controllability of problem difficulty.

Impact of sampling strategies. We introduce three strategies to sample function nodes for computational graph regeneration: co-occurrence-based, topic-based, and edgeless. To evaluate their individual contributions, we synthesize 20k training samples using each strategy exclusively and train LLaMA-3-8B-Instruct. As shown in Table 5, each strategy excels in different areas. For example, the co-occurrence strategy is strongest on GSM8K. The edgeless strategy performs best on MATH-500. The topic-based strategy shows particular strength on the more complex GSM-Hard and OlympiadBench datasets. This validates our decision to use a mix of all three, creating a balanced and robust dataset that leverages their complementary advantages.

Strategy	GSM8K	MATH	Hard	College	Olym.
Co-occur.	**77.9**	35.2	35.3	26.5	11.0
Topic	77.2	35.2	**39.2**	26.7	**12.9**
Edgeless	77.3	**39.6**	37.8	**27.1**	12.6

Table 5: Ablation study on different sampling strategies.

Controllability of problem difficulty. A key advantage of RV-Syn is the fine-grained control over synthesized problem complexity. We analyze our ability to control problem complexity by varying the number of function nodes sampled during graph regeneration. We train models on 50k datasets synthesized with a fixed number of nodes (1, 2, and 3 nodes) and evaluate them on MATH-500 and GSM-Hard. As shown in Table 6, models trained on single-node problems achieve the highest performance on MATH-500, a dataset emphasizing direct application of middle-school-level concepts. In contrast, performance on the more calculation-intensive GSM-Hard improves with increasing node count, indicating that multi-node graphs effectively model complex, multi-step reasoning. This demonstrates RV-Syn's ability to deliberately calibrate difficulty for targeted training needs.

Sampled Nodes	MATH-500	GSM-Hard
1 Node	**49.0**	42.6
2 Nodes	48.6	43.8
3 Nodes	47.2	**44.4**

Table 6: Ablation on problem difficulty control by varying sampled node counts.

4.7 Analysis of Long-CoT Distillation

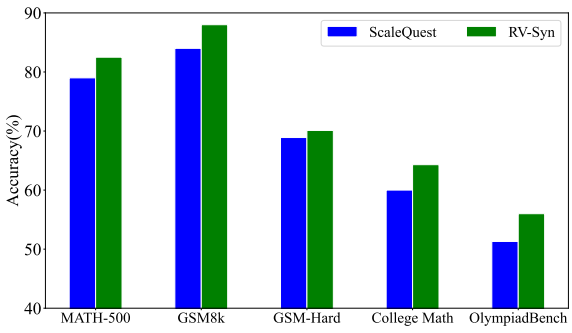

Figure 5: Performance comparison of distilling Long-CoT on 20k data.

In addition to short-CoT, we also explore the ability of our data to distill Long-CoT abilities. We replace the answer generation model with DeepSeek-R1-Distill-Qwen-7B (Guo et al., 2025) and train Qwen2.5-7B-Instruct on a 20k dataset, comparing our method with ScaleQuest. During the evaluation, we set the maximum sequence length to 16k and exclude samples where both models exceed the maximal length. We show the results in Figure 5. We can see that our synthesis method not only performs well on short-CoT but also excels in distilling Long-CoT compared to the baseline.

5 Conclusion

In this paper, we introduced RV-Syn, a novel approach to mathematical problem synthesis that enhances both rationale and verifiability. It first constructed a graph-format function library, which was composed of a large number of mathematical functions and was scalable to insert new ones. Then, it generated the new solutions by combining functions into computation graphs in a controllable manner, which can be then executed to support solution verification. Finally, it back-translated these computation graphs to problems. Our experiments demonstrate that RV-Syn outperforms existing methods, achieving higher performance with less data. This efficiency highlights its potential for improving the reasoning capabilities of LLMs while minimizing resource consumption. RV-Syn represents a significant advancement in reasoning data generation, offering a scalable and reliable framework. Future work can explore extending it to other reasoning domains and further refining control mechanisms for broader applicability.

Limitations

Despite the promising results demonstrated by the RV-Syn method, several limitations remain. First, the approach relies heavily on the quality and diversity of the initial function library. While we have curated a comprehensive set of functions from a large scale of existing synthetic datasets, there may still be gaps in representing certain mathematical concepts, potentially limiting the scope of problems that can be synthesized. Second, although our experiments show improved performance across several benchmarks, the generalization of RV-Syn to other domains beyond mathematics remains unexplored, such as the code domain. Future work should investigate the adaptability of this approach to different types of reasoning tasks and datasets. We aim to address these limitations in future re-

search by expanding the function library, optimizing computational efficiency, and exploring cross-domain applications.

Acknowledgments

This paper was partially supported by the National Natural Science Foundation of China No. 92470205 and Beijing Major Science and Technology Project under Contract no. Z251100008425002. Xin Zhao is the corresponding author.

References

AI@Meta. 2024. Llama 3 model card.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2024. Llemma: An open language model for mathematics. In *ICLR*. OpenReview.net.

Jie Chen, Zhipeng Chen, Jiapeng Wang, Kun Zhou, Yutao Zhu, Jinhao Jiang, Yingqian Min, Wayne Xin Zhao, Zhicheng Dou, Jiaxin Mao, Yankai Lin, Ruihua Song, Jun Xu, Xu Chen, Rui Yan, Zhewei Wei, Di Hu, Wenbing Huang, and Ji-Rong Wen. 2024a. Towards effective and efficient continual pre-training of large language models. *CoRR*, abs/2407.18743.

Nuo Chen, Hongguang Li, Baoyuan Wang, and Jia Li. 2024b. From good to great: Improving math reasoning with tool-augmented interleaf prompting. *CoRR*, abs/2401.05384.

Nuo Chen, Ning Wu, Jianhui Chang, Linjun Shou, and Jia Li. 2024c. Controlmath: Controllable data generation promotes math generalist models. In *EMNLP*, pages 12201–12217. Association for Computational Linguistics.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Yuyang Ding, Xinyu Shi, Xiaobo Liang, Juntao Li, Qiaoming Zhu, and Min Zhang. 2024. Unleashing reasoning capability of llms via scalable question synthesis from scratch. *CoRR*, abs/2410.18693.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: program-aided language models. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *ICLR*. OpenReview.net.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. Olympiadbench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *ACL (1)*, pages 3828–3850. Association for Computational Linguistics.

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. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Yiming Huang, Xiao Liu, Yeyun Gong, Zhibin Gou, Yelong Shen, Nan Duan, and Weizhu Chen. 2024a. Key-point-driven data synthesis with its enhancement on mathematical reasoning. *CoRR*, abs/2403.02333.

Yinya Huang, Xiaohan Lin, Zhengying Liu, Qingxing Cao, Huajian Xin, Haiming Wang, Zhenguo Li, Linqi Song, and Xiaodan Liang. 2024b. MUSTARD: mastering uniform synthesis of theorem and proof data. In *ICLR*. OpenReview.net.

Chen Li, Weiqi Wang, Jingcheng Hu, Yixuan Wei, Nanning Zheng, Han Hu, Zheng Zhang, and Houwen Peng. 2024a. Common 7b language models already possess strong math capabilities. *CoRR*, abs/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. *CoRR*, abs/2310.05506.

Chengpeng Li, Zheng Yuan, Hongyi Yuan, Guanting Dong, Keming Lu, Jiancan Wu, Chuanqi Tan, Xiang Wang, and Chang Zhou. 2024b. Mugglemath: Assessing the impact of query and response augmentation on math reasoning. In *ACL (1)*, pages 10230–10258. Association for Computational Linguistics.

Haoran Li, Qingxiu Dong, Zhengyang Tang, Chaojun Wang, Xingxing Zhang, Haoyang Huang, Shaohan Huang, Xiaolong Huang, Zeqiang Huang, Dongdong

Zhang, Yuxian Gu, Xin Cheng, Xun Wang, Si-Qing Chen, Li Dong, Wei Lu, Zhifang Sui, Benyou Wang, Wai Lam, and Furu Wei. 2024c. Synthetic data (almost) from scratch: Generalized instruction tuning for language models. *CoRR*, abs/2402.13064.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. 2024d. Numinamath. [https://huggingface.co/AI-MO/NuminaMath-CoT](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf).

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Haoxiong Liu, Yifan Zhang, Yifan Luo, and Andrew Chi-Chih Yao. 2024. Augmenting math word problems via iterative question composing. *CoRR*, abs/2401.09003.

Zimu Lu, Aojun Zhou, Houxing Ren, Ke Wang, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. 2024. Mathgenie: Generating synthetic data with question back-translation for enhancing mathematical reasoning of llms. In *ACL (1)*, pages 2732–2747. Association for Computational Linguistics.

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. *CoRR*, abs/2308.09583.

Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. 2024. Orca-math: Unlocking the potential of slms in grade school math. *CoRR*, abs/2402.14830.

OpenAI. 2024. Learning to reason with llms.

Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Nan Rosemary Ke, Michael Mozer, Yoshua Bengio, Sanjeev Arora, and Anirudh Goyal. 2024. Ai-assisted generation of difficult math questions. *CoRR*, abs/2407.21009.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*.

Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. 2024. Mathscale: Scaling instruction tuning for mathematical reasoning. In *ICML*. OpenReview.net.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. 2024. Dart-math: Difficulty-aware rejection tuning for mathematical problem-solving. In *NeurIPS*.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. 2024. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. In *NeurIPS*.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2024. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning. In *ICLR*. OpenReview.net.

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

Jiaxin Wen, Jian Guan, Hongning Wang, Wei Wu, and Minlie Huang. 2024. Codeplan: Unlocking reasoning potential in large langauge models by scaling code-form planning. *CoRR*, abs/2409.12452.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024a. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *ICLR*. OpenReview.net.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. 2024b. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *CoRR*, abs/2406.08464.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. 2024. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *CoRR*, abs/2409.12122.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *ICLR*. OpenReview.net.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Metamath: Bootstrap your own mathematical questions for large language models. In *ICLR*. OpenReview.net.

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

Xiang Yue, Tuney Zheng, Ge Zhang, and Wenhu Chen. 2024. Mammoth2: Scaling instructions from the web. *CoRR*, abs/2405.03548.

Weihao Zeng, Can Xu, Yingxiu Zhao, Jian-Guang Lou, and Weizhu Chen. 2024. Automatic instruction evolving for large language models. In *EMNLP*, pages 6998–7018. Association for Computational Linguistics.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A survey of large language models. *CoRR*, abs/2303.18223.

Xueliang Zhao, Wei Wu, Jian Guan, and Lingpeng Kong. 2025. Promptcot: Synthesizing olympiad-level problems for mathematical reasoning in large language models. *CoRR*, abs/2503.02324.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *ICLR*. OpenReview.net.

Kun Zhou, Beichen Zhang, Jiapeng Wang, Zhipeng Chen, Wayne Xin Zhao, Jing Sha, Zhichao Sheng, Shijin Wang, and Ji-Rong Wen. 2024. Jiuzhang3.0: Efficiently improving mathematical reasoning by training small data synthesis models. *CoRR*, abs/2405.14365.

A Prompts for Data Generation Pipeline

In this part, we present the prompts used in the three main components of our method: Problem Decomposition, Computational Graph Regeneration, and Problem Back-translation.

Prompt for Problem Decomposition

Your task is to solve the following math problem using Python code, employing a ‘main‘ function along with several custom operation functions.

Problem: {PROBLEM}

Answer the question step by step first, understand the logic of the problem. Then:

1. Represent calculation operations as functions, where each function specifies its functionality, input parameter types, and output return types.
2. Each operation function should represent a complete, meaningful computational skill or operation, and should be reusable across different problems.
3. Simple operations such as addition, subtraction, multiplication and division do not need to be expressed as functions.
4. Each function should have a specific, independent purpose for performing a particular calculation. You may use functions from standard or imported libraries (like SymPy, math, etc.; Import within the function itself), but **cannot** use other custom operation functions.
5. The operation functions should be flexible, not hardcoded, and able to handle various scenarios by accepting different parameters.
6. Solve the problem by calling the operation functions within the main function.
7. Only print the answer without any text.

Prompt for Function Labelling

Please analyze the following mathematical function operation, judge its functional correctness and assign it a subject category.

{function}

- First analyze the function and evaluate its correctness. Output Yes or No for correctness judgement.
- There are 7 possible subject categories to choose from: Prealgebra, Intermediate Algebra, Algebra, Number Theory, Precalculus, Geometry, Counting Probability, and Miscellaneous.

Output example:
<analyse>analyze here</analyse>
<correctness>Yes</correctness>
<subject>Number Theory</subject>

Prompt for Computational Graph Regeneration Fine-tuning

Given a set of mathematical operation functions:
```
{FUNCTIONS}
```
Write a ‘main‘ function that intelligently combines these operations to solve a meaningful and non-trivial mathematical problem.

Prompt for Problem Back-translation

Given a piece of code that solves a mathematical problem:
```
{CODE}
```
First, understand what the code snippet is doing within the ‘<analysis></analysis>‘ tags. Then, identify the original problem the functions are solving and output it within the ‘<problem></problem>‘ tags. Finally, rewrite the problem concisely, in a manner similar to questions found in textbooks or test papers, and output it within the ‘<final problem></final problem>‘ tags. The question should have only one answer.

Prompt for Correctness Verification

Your goal is to determine if the given problem and its solution contain mistakes.

If both the problem and the solution are correct, output 'Correct'.
If the problem contains error such as inconsistency, conflict or no solution, output 'Problem Error'.
If the problem is correct but the solution contains error, output 'Solution Error'.

Problem: {problem}

Solution: {solution}

B Details for Correctness Evaluation

Considering the substantial cost associated with human evaluations, we use one of the most powerful models currently available, GPT-4o, to conduct correctness evaluation. To ensure its validity, we conducted an experiment using GPT-4o to evaluate the accuracy of Qwen2.5-Math-7B-Instruct on the benchmark test set and compared the results with ground-truth evaluations. As shown in Table 7, the evaluation results—74.25% (GPT-4o) vs. 76.61% (Ground truth)—demonstrate a close alignment. False positive samples account for only 1.89% of all cases, while the false negative rate is even higher, indicating that GPT-4o tends to be more stringent in its evaluation. This is likely because it not only verifies the final answer but also checks intermediate steps. Considering that even ground-truth-based evaluations may contain some degree of error, we believe this discrepancy falls within an acceptable range. The prompt we use is shown in Appendix A.

		Ground Truth		
		Correct (+)	**Incorrect (–)**	**Total**
GPT-4o	**Correct (+)**	0.7236	0.0189	0.7425
	Incorrect (–)	0.0425	0.2150	0.2575
Total		0.7661	0.2339	1.0000

Table 7: Confusion matrix for GPT-4o evaluation.

C Example of Problem Decomposition

Problem: What is the largest positive integer with only even digits that is less than $10,000$ and is a multiple of 9?

Generated_text:

{CoT reasoning step by step here}

```python
def generate_numbers(max_num, even_digits):
    """
    Generate all possible numbers with even digits up to max_num.
    Parameters:
        max_num (int): The maximum number up to which we generate numbers.
        even_digits (list): A list of even digits.
    Returns:
        list: A list of generated numbers.
    """
    numbers = []
    for i in range(max_num):
        num_str = str(i)
        if all(digit in even_digits for digit in num_str):
            numbers.append(i)
    return numbers

def check_multiples(numbers, multiple_of):
    ...

def find_largest(multiples):
    ...

def main():
    max_num = 10000
    even_digits = ['0', '2', '4', '6', '8']
    multiple_of = 9

    numbers = generate_numbers(max_num, even_digits)
    multiples = check_multiples(numbers, multiple_of)
    largest_multiple = find_largest(multiples)

    print(largest_multiple)
main()
```

D Analysis of Data Diversity

A key limitation of data augmentation methods like MetaMath (Yu et al., 2024) is their tendency to produce problems that are structurally similar to the original seeds. This often results from the inherent paraphrasing nature of in-context learning, leading to limited diversity. For example, a seed problem asking to "choose 3 captains from a team of 11 people" might be augmented to "choose 2 captains from a team of 9 people", a trivial variation. RV-Syn also utilizes seed data, but in a fundamentally different manner. Instead of rewriting whole problems, RV-Syn decomposes them to extract a library of atomic mathematical operations (functions). Our synthesis process then generates new problems not by modifying an existing problem, but by creating entirely new computational graphs by combining these atomic functions. This combinatorial approach allows for the creation of novel reasoning paths and complex problem structures that can be fundamentally different from any single seed problem, thereby fostering genuine diversity.

To empirically validate this claim, we quantitatively compare the diversity of data generated by RV-Syn against MetaMath, relative to the seed data. We employed three metrics: Δ Mean of Embedding Variance: The change in the average variance of problem embeddings. A positive value indicates increased diversity. Centroid Cosine Distance and Maximum Mean Discrepancy (MMD^2): Both measure the dissimilarity between the generated data distribution and the original seed distribution. Larger values signify greater divergence.

The results, presented in Table 8, strongly support our hypothesis. MetaMath shows a negative change in embedding variance, suggesting it slightly reduces diversity. In sharp contrast, RV-

Method	Δ Mean of Emb. Variance	Centroid Cosine Dist.	MMD^2
MetaMath	-0.0017	0.0170	0.000825
RV-Syn	**+0.0019**	**0.0398**	**0.002493**

Table 8: Diversity and similarity analysis. RV-Syn increases embedding variance and generates data that is more distinct from the seed set compared to MetaMath.

Syn increases diversity. Furthermore, the significantly larger Centroid Cosine Distance and MMD^2 values for RV-Syn confirm that its output diverges more substantially from the seed distribution. This demonstrates that our function-centric, combinatorial synthesis is more effective at producing novel and diverse problems than simple seed-based augmentation.

E Statistics of the Synthesis Pipeline

Here, we present statistics for the function library and the final dataset. Figure 6 (a) shows the topic distribution in our function library, where each function is labeled according to the categories defined in the MATH dataset. Figure 6 (b) displays the difficulty distribution of the synthesized dataset, with the majority of samples concentrated in the higher-difficulty range. This distribution helps explain why RV-Syn achieves more substantial performance gains on more challenging benchmarks. Figure 6 (c) details the filtering proportions during the synthesis pipeline. We observed that 16% of the generated computational graphs failed to execute and were discarded. Among the remaining valid graphs, an additional 37% of the generated problem–solution pairs were filtered out because no Chain-of-Thought solution matched the ground-truth answer produced by graph execution. This automated, graph-based quality control mechanism is crucial for ensuring the high correctness of our final dataset. Table 9 provides additional statistics for the function library.

Metric	Value
CB Connected Components Number	23419
CB Largest Connected Component Size	10847
CB Average Degree per Node	1.225
TB Connected Components Number	8
TB Largest Connected Component Size	13638
TB Average Degree per Node	8607

Table 9: Graph-based function library statistics. CB denotes Co-occurrence Based, TB denotes Topic Based.

F Impact of the Answer Annotation Model

Previous works often directly fine-tune models on publicly available datasets released by baseline methods and compare their performance. However, we find that the choice of the answer annotation model is one of the most critical factors influencing downstream performance, which introduce substantial unfairness into such comparisons. To illustrate this, we re-annotate existing datasets using a unified model, Qwen2.5-Math-7B-Instruct, and compare the performance of LLaMA-3-8B-Instruct fine-tuned on 50k these newly annotated datasets versus the original versions. The results in Table 10 show that many methods achieve significant performance improvements, and the performance gap between them narrows. In all our experiments, we use the same annotation model to ensure a fair comparison and to isolate the impact of problem quality. We observe consistent performance improvements from our method. Although some of the improvements may appear modest under the uniform annotation setup, they in fact represent substantial enhancements in data quality, which are partially obscured by the controlled evaluation conditions.

Dataset	Original	Re-annotation	Performance Change
Jiuzhang3.0	30.5	42.1	+38.0%
MathScale	26.5	41.9	+58.1%
MMIQC	34.9	43.7	+25.2%
Numina-Math	38.7	43.9	+13.4%

Table 10: Performance of LLaMA-3-8B-Instruct Fine-tuned on Re-annotated Datasets using Qwen2.5-Math-7B-Instruct.

G Influence of Seed Dataset

Since RV-Syn utilizes the high-quality Numina-Math dataset as seed data, we determine whether the performance gains stem from our synthesis framework or merely the superior seed data. We conduct a controlled experiment using the Jiuzhang3.0 baseline. We replace its original web-corpus seed data with Numina-Math and train it on 50k synthesized samples. The results in Table 11 show that simply using better seed data with a direct synthesis method (Jiuzhang3.0) does not improve performance; in fact, the score slightly drops compared to its original setting (39.5 vs. 42.1). We attribute this to the fact that direct synthesis

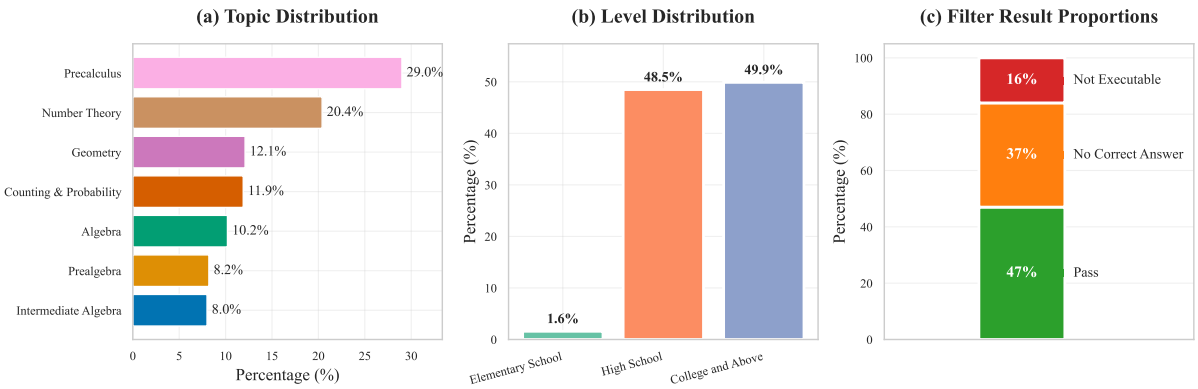

Figure 6: Statistical analysis of the RV-Syn pipeline. (a) Distribution of mathematical topics within the function library. (b) Difficulty distribution of the final synthesized problems. (c) Proportions of data filtered out at different stages of the synthesis process.

Method	Seed Data	Avg. Score
Jiuzhang3.0	Original (Web)	42.1
Jiuzhang3.0	Numina-Math	39.5
RV-Syn (Ours)	Numina-Math	**44.9**

Table 11: Impact of seed dataset on performance (50k data, LLaMA-3-8B). Comparing our method against Jiuzhang3.0 seeded with Numina-Math.

methods rely heavily on the diversity of broad pre-training corpora, and constraining them to a structured QA dataset may limit their generative potential. In contrast, RV-Syn achieves a significantly higher average score (44.92), confirming that our improvements are driven by the rational synthesis framework rather than the seed data alone.

H Examples of Function Similarity

We provide examples of semantic and structural similarity in Table 12. Specifically, functions that share these similarities or have the same docstring are grouped into the same function set.

Semantic Similarity

```python
def find_gcd(a: int, b: int) -> int:
    """
    Find the greatest common divisor of two numbers.
    Args:
    a (int): The first number.
    b (int): The second number.
    Returns:
    int: The GCD of a and b.
    """
    return math.gcd(a, b)
```

```python
def gcd(a, b):
    """
    Find the greatest common divisor of two numbers.
    Args:
    a (int): The first number.
    b (int): The second number.
    Returns:
    int: The GCD of a and b.
    """
    while b:
        a, b = b, a % b
    return a
```

Structure Similarity

```python
def calculate_matrix_product(matrix1, matrix2):
    """
    Calculate the product of two matrices.
    Parameters:
        matrix1 (numpy.array): The first matrix.
        matrix2 (numpy.array): The second matrix.
    Returns:
        numpy.array: The product of matrix1 and matrix2.
    """
    return np.dot(matrix1, matrix2)
```

```python
def matrix_transformation(projection_matrix1, projection_matrix2):
    """
    Calculate transformation matrix by multiplying
    projection matrices.
    Args:
    - projection_matrix1: The first projection matrix.
    - projection_matrix2: The second projection matrix.
    Returns:
        numpy array: The transformation matrix.
    """
    return np.dot(projection_matrix2, projection_matrix1)
```

Table 12: Examples of Semantic and Structure Similarity.