AdaR: A Framework for Equipping LLMs with Adaptive Reasoning

Anonymous ACL submission

Abstract

Mathematical reasoning is a primary indicator of large language models (LLMs) intelligence. However, existing LLMs exhibit failures of robustness and generalization. This paper attributes these deficiencies to spurious reasoning, wherein generated reasoning traces bear negligible causal connection to answers. To address this challenge, we propose the AdaR framework to equip LLMs with adaptive reasoning, wherein models rely on problem-solving logic to produce answers. AdaR automatically synthesizes logically equivalent queries by varying variable values, and trains models with Reinforcement Learning with Verifiable Rewards (RLVR) on these data to penalize spurious logic while encouraging adaptive logic. To ensure data quality, we extract the problem-solving logic from the original query and generate the corresponding answer by code execution and then apply sanity check. Experimental results demonstrate that AdaR achieves substantial improvement in mathematical reasoning while maintaining high data efficiency. Furthermore, even for advanced LLMs, there still exists robustness and generalization deficiencies, which our work effectively mitigates.

1 Introduction

Large Language Models (LLMs) have demonstrated strong performance across various reasoning tasks (Wei et al., 2022a). Among these, mathematical reasoning serves as a crucial cognitive skill that supports problem-solving across tasks (Huang and Chang, 2023). Beyond early direct inference attempts (Liu et al., 2023; Brown et al., 2020), Chain-of-Thought (CoT) reasoning has been recognized as an effective approach to enhance mathematical reasoning (Wei et al., 2022b), as it breaks down complex problems into manageable steps (Chu et al., 2024).

However, existing mathematical LLMs still exhibit failures at two levels: (i) robustness on in-domain tasks (Mirzadeh et al., 2024); and (ii) generalization on out-of-domain tasks (Jahin et al., 2025). We attribute these deficiencies to **spurious reasoning**, where LLMs derive the answer y from superficial patterns rather than the underlying problem-solving logic L, yielding CoTs that are semantically similar but logically inequivalent to correct ones. As shown in Figure 1, the model solves query in training set correctly, but after only numerical values are changed, it produces a similar CoT that omits a key reasoning token, e.g., "cube", thereby altering the problem-solving logic and producing an incorrect answer. Thus, even with an unchanged query template, models relying on spurious reasoning fail to adapt to changes in the variable set x. Meanwhile, spurious reasoning cannot be constructed from atom of thoughts (Teng et al., 2025). Therefore, it's non-compositional along causal relations which in turn causes models to generalize ineffectively.

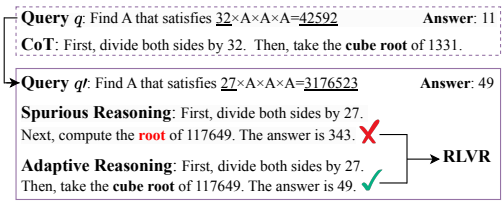


Figure 1: The examples of spurious and adaptive reasoning.

To overcome the mentioned weaknesses, the model is expected to understand the logic L as a function $L(.)$, by correctly understanding the query q and disentangling x from q, which is an ability that reflects algebraic reasoning (Kieran, 2004). Applying the disentangled x to $L(.)$ may reveal the correct answer y. We introduce this process as **adaptive reasoning**. The query template is denoted as T, with x removed from q.

Learning adaptive reasoning from limited data is difficult. Thus previous work employ data syn-

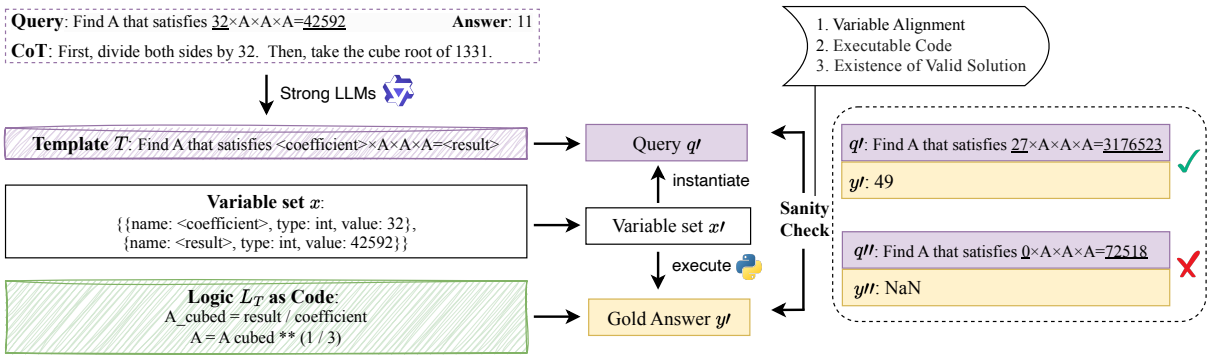


Figure 2: The pipeline of how we get the query-answer pairs by controllably perturbing variable values while preserving problem-solving logic and sanity.

thesis (Yu et al., 2023; Lu et al., 2024). According to the above analysis, synthesizing mathematical data given an L have two essential dimensions: the query template T, and the variable set x. Yu et al. (2023) predominantly concentrated on perturbing T (e.g., paraphrasing) while keeping x fixed. However, training with varing T with fixed x may increase the risk of spurious reasoning overfitting to x. In contrast, Lu et al. (2024) attempted to prompt the LLM to modify x; however, the employed strategy leaves x uncontrolled, and may also inadvertently alter L_T caused by hallucination (Huang et al., 2025), making it difficult to ensure the correctness of generated gold answer y.

In order to equip LLMs with **Ada**ptive **R**easoning, we propose the framework **AdaR**, which synthesizes data along both T and x dimensions and performs reinforcement learning with them. Crucially, AdaR introduces an automatic process for high-precision, controllable numerical perturbations, while ensuring verified y. To achieve this end, we decompose the complex perturbation task into the following manageable, verifiable subtasks. As shown in Figure 2, we first prompt an strong LLM to generate T, L_T (as code, e.g. a Python program), and x. Subsequently, we *controllably perturb* values in x to predefined magnitudes and types. The perturbed variable set x' are then used to instantiate T to generate queries q'; while be provided as input to L_T, which is executed to produce y'. Furthermore, we introduce *sanity check* to filter invalid instances.

AdaR then trains the model by Reinforcement Learning with Verifiable Rewards (RLVR) (Shao et al., 2024) to equip LLMs with adaptive reasoning. In the single-data setting, outcome rewards are unreliable, since spurious reasoning may still lead to correct answers through superfical patterns. However, when inferring on perturbed queries from AdaR, spurious reasoning is unlikely to remain valid across these queries, outcome correctness provides a more reliable reward, penalizing spurious reasoning and pushing the model to explore the adaptive problem-solving logic.

Extensive experiments across in-domain robust tasks and out-of-domain tasks demonstrate that AdaR achieves great gains (+8.50 points on average), with only 9K synthetic data on both math-specialized and general base models, thereby demonstrating that our approach enhances the model's robustness and generalization. Further analysis indicates that:

- Advanced LLMs still exhibit pronounced robustness and generalization deficiencies, which our work effectively mitigates.
- Perturbing the variable values is more effective than perturbing the query template for improving model's performance.
- Evidence of enhanced sensitivity to logical order and improved algebraic thinking demonstrates that AdaR equips LLMs with adaptive reasoning

2 Method

In this section, we first present methods for synthesizing data that ensure controllability and sanity throughout the generation process. Following this, we introduce our training strategy, which is employed to more effectively integrate with synthetic data, preventing the model from learning spurious reasoning, thereby facilitating adaptive reasoning.

2.1 Data Synthesis with Executable Code and Verifiable Answers

A straightforward way to perturb values in queries and obtain correct answers without human annotation is to directly prompt a LLM (Wang et al.,

2025). However, our preliminary experiments show that such approach provides limited control over neither the magnitudes nor types of perturbations, and cannot guarantee answer correctness. To address these limitations, AdaR decomposes data synthesis into a sequence of manageable, verifiable sub-tasks. Specifically, the LLM is only tasked with extracting necessary metadata within the data, including query template and variable set, and translating the given CoT into code; controllable perturbations are then applied externally, and sanity check are performed to ensure the correctness of synthesized query-answer pairs. We detail each sub-task below.

Convert Logic in Text to Logic in Code. Executable problem-solving code can serve as a problem-solving logic provided its correctness is guaranteed. Crucially, such code generalizes effectively: by substituting the input variable set, it can solve any perturbed query of the original query, for its output can serve as reliable gold answer. When CoT and gold answer are provided, problem-solving code generation becomes a straightforward process of translating the textual logic (i.e. CoT) into executable code, thereby reducing the model's reasoning burden and enabling direct verification of correctness against the gold answer. Building on these insights, we provide a query q, the corresponding CoT z and gold answer y to an open-source LLM to synthesize the logic L_T as code for producing gold answer y' of subsequent perturbed query q'. Because code generation requires abstracting the concrete numerical values in q into a mapping to input variables, we further instruct the model to produce query template T as a byproduct by applying this mapping into the original query q. Appendix A.3 details the employed prompt .

Controllable Perturbation. By comparing q with T, we construct a variable set x that records each variable's name, numeric type and its value as it appears in the original query. Since the values are explicitly extracted, perturbations can be applied directly at a predefined magnitude while ensuring numerical validity, thereby avoiding the unpredictability of perturbations performed through an LLM. More specifically, we then apply independent perturbations by sampling each variable's value within a range of $\pm\alpha\%$ of its original value. The parameter α allows the control of the perturbation magnitude. To ensure numerical validity, both the numeric type and the sign of every variable must remain unchanged before and after perturbation. Formally, for the i-th variable with original value x_0^i, we draw:

$$x^i = x_0^i \times (1 + \Delta_i), \Delta_i \sim \text{Uniform}(-\alpha\%, \alpha\%)$$
$$\text{s.t. type}(x^i) = \text{type}(x_0^i),\ \text{sign}(x^i) = \text{sign}(x_0^i) \tag{1}$$

Sanity Check. After applying perturbations, we instantiate T with x' to obtain q', and execute L_T as code with the same x' as input to generate y'. Although providing z and y has reduced task difficulty, T and L_T as code generated by LLMs remain uncertain and may contain errors. Furthermore, a meaningful query imposes inter-variable constraints on its variables; independent perturbations can violate these constraints and thereby introduce errors. To ensure that the perturbed data remain well-posed, we conduct a *sanity check* along the following aspects:

- **Variable Alignment (VA).** We compare variables referenced in T with those used in each L_T as code. Any mismatch indicates potential errors (e.g. hallucination).
- **Executable Code (EC).** Since L_T as code is used to derive the y, executability is a critical requirement: (i) the code runs without runtime errors; and (ii) providing the original variable set x_0 as input reproduces the gold answer y_0.
- **Existence of Valid Query (EVQ).** As independent perturbations may violate implicit inter-variable constraints, they can produce invalid queries (e.g., selecting 20 items from a set of 10). To provide evidence that a valid query exists, we cross-validate the code-executed answer with the answer produced by a mathematical LLM. This validation is based on the following rationale. LLMs are prone to hallucinate under out-of-distribution inputs (i.e. invalid queries) (Huang et al., 2025). Given the large solution space of mathematical queries, exact agreement between the two answers is unlikely unless the query is valid and both code and the LLM follow the correct logic. The latter condition is implemented because EC ensures that L_T as code is correctly translated from provided CoT, meanwhile we provide L_T as contextual guidance to help LLM align with correct logic when inferring. We therefore use answer agreement as a evidence of the existence of a valid query.

If a perturbed instance fail to pass the *sanity check*, we reattempt *controllable perturbation*. If the number of attempts exceeds τ times, we conclude this instance likely involves complex inter-variable constraints and then discard it from synthesis.

Queries Paraphrasing. We adopt paraphrasing (Yu et al., 2023) as a data augmentation strategy to increase the diversity of T. This approach complements the *controllable perturbation* introduced earlier to increase the diversity of x. The impact of these scalable dimensions on model performance is further explored in Section 3.3 and Section 4.2.

Remark. *Our synthetic data contains no CoT, it consists solely of the query and the corresponding gold answer. As will be described in the next subsection, CoTs generated through either spurious reasoning or adaptive reasoning can be sampled from the target LLM.*

2.2 Training Strategy

The training objective of Supervised Fine-Tuning (SFT) and its variant Rejection sampling Fine-Tuning (RFT) (Yuan et al., 2023) (detailed in Appendix A.2) makes models prone to memorizing provided CoTs rather than developing adaptive reasoning (Chu et al., 2025), thereby causing them to converge to local optimum (Kang et al., 2025). RLVR has recently been widely adopted to enhance generalization (Guo et al., 2025). However, its outcome reward generated by spurious reasoning is indistinguishable from that generated by adaptive reasoning, which may inadvertently reinforce spurious reasoning.

To address this, we combine RLVR with our synthetic data. RLVR's reward-driven enables the model to learn from comparison among rewards obtained by solving perturbed queries using different reasoning process. More specifically, when given the original query q, it's hard to determine where the gold answer y is derived. However, when the model is evaluated on perturbed query q', reliance on CoT from spurious reasoning is more likely to yield incorrect outcome, whereas reliance on CoT from adaptive reasoning $L_T(.)$ is more likely to yield correct outcome. All perturbed queries are placed into the same batch. Then, each reasoning process is more likely to receive appropriate feedback, thereby promoting adaptive reasoning.

3 Experiment

3.1 Experimental Setup

Data synthesis. We use the Qwen2.5-72B-Instruct (Yang et al., 2024) as the open-source LLM to generate the query templates and the problem-solving codes. We select $9K$ instances from ORCA-MATH (Mitra et al., 2024) as seed data for data synthesis. Using these seed data, AdaR synthesizes instances with a predefined magnitude $\alpha = 500$ and a maximum of $\tau = 50$ attempts. For each data in seed data, we select one corresponding synthetic data to construct the "ORCA-AdaR-train", which contains $9K$ instances. We select another $2.5K$ instances to form "ORCA-AdaR-test", ensuring no overlap with "ORCA-AdaR-train". The details are shown in Appendix A.4.1.

Models. To prove the adaptability of our framework, we conduct experiments on two categories of base models: math specialized base LLM, specifically Qwen2.5-Math-7B (Yang et al., 2024) and DeepSeekMath-7B (Shao et al., 2024), and 8B general base LLM, specifically LLaMA3-8B (Grattafiori et al., 2024). In Section 4.5, we further explore the performance of AdaR on Instruct Model.

Evaluation. For a comprehensive evaluation of mathematical reasoning, we adopt 7 benchmarks covering both in-domain and out-of-domain evaluation. Specifically, we evaluate in-domain mathematical competence using GSM8K (Cobbe et al., 2021) and evaluate in-domain robustness using ORCA-AdaR-test (ORCA-AdaR) and GSM-SYM (main/p1/p2) (Mirzadeh et al., 2024). Within GSM-SYM, the main, p1, and p2 subsets exhibit increasing difficulty. For out-of-domain evaluation, we use MATH (Hendrycks et al., 2021), CollegeMath (Tang et al., 2024), TheoremQA (Chen et al., 2023), and American Invitational Mathematics Examination (AIME) problems for 2025 to evaluate generalization. Results are reported as pass@1 for all datasets except AIME 2025, for which, in accordance with (Yu et al., 2025), we report avg@32. Further details about the evaluation setup and benchmarks are provided in the Appendix A.4.3.

Baseline. We primarily compare our framework, AdaR, against the following baselines: (1) The base models undergo SFT using 9K seed data—this same data is also leveraged for data synthesis.

Method	In-Domain					Out-of-Domain				AVG
	GSM8K	ORCA-AdaR	main	p1	p2	MATH	College	Theorem	AIME	
Qwen2.5-MATH (7B MATH-Specialized Base Model)										
Initial SFT	81.43	77.12	74.26	64.48	52.44	42.84	28.55	14.88	4.27	48.92
Standard-RLVR	86.96	81.80	81.34	74.32	64.68	61.74	29.48	20.38	10.94	56.85
MetaMATH	84.61	82.44	79.78	71.24	63.28	66.92	39.25	26.13	9.38	58.11
MathGenie	87.64	84.96	80.40	72.60	63.56	55.94	21.11	20.05	10.31	55.17
AdaR	**91.81**	**86.08**	**89.72**	**82.74**	**73.24**	**75.90**	**48.62**	**37.13**	**14.27**	**66.61**
DeepSeekMath (7B MATH-Specialized Base Model)										
Initial SFT	70.28	65.48	61.70	50.26	29.36	28.44	22.32	11.50	0.21	37.73
Standard-RLVR	81.43	72.56	74.76	64.34	40.00	39.26	21.61	20.38	0.73	46.12
MetaMATH	80.89	70.28	75.14	64.54	39.56	38.02	19.13	22.63	0.21	45.60
MathGenie	80.13	73.44	74.18	64.02	40.04	39.16	22.39	23.75	**3.33**	46.72
AdaR	**81.43**	**75.44**	**76.56**	**64.94**	**42.00**	**41.18**	**24.23**	**25.25**	**3.33**	**48.26**
Llama3 (8B General Base Model)										
Initial SFT	67.17	57.84	59.98	47.30	24.92	18.20	9.33	8.13	0.00	32.54
Standard-RLVR	73.09	55.68	67.06	51.58	25.16	19.02	9.01	9.00	0.00	34.40
MetaMATH	74.52	59.96	70.74	56.58	26.72	19.68	9.04	9.38	0.00	36.29
MathGenie	72.47	58.90	67.26	53.72	25.38	18.30	8.92	9.00	0.00	34.88
AdaR	**77.77**	**61.52**	**73.96**	**56.90**	**30.28**	**20.48**	**9.62**	**9.63**	0.00	**37.80**

Table 1: Performance comparison across In-Domain and Out-of-Domain mathematical benchmarks. The datasets "main", "p1", and "p2" are from GSM-SYM. Best results are highlighted in bold.

These models are referred to as the "Initial SFT" models. (2) In "Standard-RLVR" setting, the "Initial SFT" is subsequently trained with RLVR on another 9K subset of ORCA-MATH, with its templates being disjoint from the training set used for "Initial SFT". (3) We also include other synthesis methods, such as MetaMATH (Yu et al., 2023), which improves query diversity through paraphrasing and self-verification; MathGenie (Lu et al., 2024), which generate questions by applying back-translation to paraphrased responses. For all RLVR training strategy, we adopt DAPO (Yu et al., 2025). Additional details of the training setup are provided in the Appendix A.4.2.

3.2 Main Results

The main results are summarized in Table 1. We highlight the following three observations:

Observation 1: Our method with a small amount of synthetic data yields substantial performance gains. Using only 9K synthetic data, our method surpasses other methods across all base models. Compared to other synthetic data methods, we improve over MetaMATH by 8.50 points and over MathGenie by 11.44 points on average.

Observation 2: Our method significantly enhances model robustness and generalization. The enhancement manifests across three levels: (i) perturbed variable values in seen queries during training (ORCA-AdaR-test); (ii) perturbed variable values in unseen queries during training (GSM-SYM main/p1/p2, where main, p1, and p2 represent increasing levels of difficulty); (iii) out-of-domain data. Across all base models, we observe gains on the first two levels compared to the sub-optimal MetaMATH ($+4.66$ / $+4.75$ points). At a finer level of granularity, we observe that "Initial SFT" rarely produces correct answers across all perturbations of variable values within a specific query template. In contrast, AdaR significantly improves this capability. Besides, on the third level we obtain larger improvements: when using Qwen2.5-MATH-7B as the base model, the average gain reaches +8.56 points. These results indicate that the model generates CoTs from adaptive reasoning rather than spurious reasoning, which better supports robustness and generalization.

Observation 3: The effectiveness of our method correlates with the mathematical reasoning ability of the base model. The number of additional rollouts required to obtain both correct and incorrect answers follows the ordering Qwen2.5-MATH(458.00) < DeepSeek-Math(563.20) < LLama3(942.83), which is inversely correlated with their respective performance gains (+17.69, +10.53, and +5.26 points). It implies possessing sufficient mathematics-related knowledge is a necessary condition for sampling both positive and negative responses, thereby facilitating the learning of adaptive reasoning. The result also suggests that AdaR is complementary to

Method	Strategy	Sanity Check			Paraphrase	In-Domain				Out-of-Domain
		VA	EC	EVQ		ORCA-AdaR	main	p1	p2	
Initial SFT	-	-	-	-	-	77.12	74.26	64.48	52.44	22.63
AdaR	RFT	✓	✓	✓	✓	82.32	76.54	67.66	55.48	23.82
	RLVR	✗	✓	✓	✓	85.36	87.98	80.34	69.12	39.04
	RLVR	✓	✗	✓	✓	-	-	-	-	-
	RLVR	✓	✓	✗	✓	85.48	89.06	81.90	71.88	40.17
	RLVR	✓	✓	✓	✗	**87.56**	89.63	80.96	71.88	41.31
	RLVR	✓	✓	✓	✓	86.08	**89.72**	**82.74**	**73.24**	**43.98**

Table 2: Ablation Study. Best results are highlighted in bold.

the mathematical pre-training with a large amount of real or synthetic mathematical data.

3.3 Ablation study

The AdaR framework employs techniques to ensure the model can master adaptive reasoning from controllable and diverse synthetic data: (i) deploying the RLVR training strategy; (ii) introducing the *sanity check*, comprising Variable alignment (VA), Executable Code (EC), and Existence of Valid Query (EVQ); (iii) incorporating the paraphrase to further enhance the diversity of query templates.

As shown in Table 2, each technique within the AdaR framework is essential to the final performance. Notably, without EC, directly applying subsequent EVQ retains only 0.2% data, which is insufficient to support training; consequently, no result is reported for this configuration. Although this indicates that EVQ functionally contains EC, the rule-based EC is more computationally efficient than the model-based EVQ, by $218\times$ (as shown in Appendix A.8), and thus remains essential. Besides, the result proves that paraphrase is complementary to perturbation of variable values, since it enhances the ability to understand different templates of specific problem-solving logic.

To further analysis the benefits of RLVR, we disentangle reasoning ability from computational ability. Reasoning ability is evaluated by the accuracy of code generation. Computational ability is evaluated by providing the code and instructing the model to execute it (the full prompt is provided in Appendix A.3). As summarized in the Table 3, relative to "Initial-SFT", "AdaR-RFT" improves computation while degrading reasoning, whereas "AdaR" with RLVR yields substantial improvements in both reasoning and computation ability. It is consistent with Chu et al. (2025): RFT's objective tends to memorize superficial features, thereby exacerbating overfitting to computation patterns rather than the corresponding problem-solving logic, while RLVR improves both abilities by encouraging the maximization of the exploration reward.

Method	ORCA-AdaR-test	Reasoning	Computation
Initial-SFT	77.12	75.24	54.92
Standard-RLVR	81.80	69.52	68.52
AdaR-RFT	82.32	72.36	75.36
AdaR	**86.08**	**80.96**	**90.76**

Table 3: Evaluation of reasoning and computation ability on ORCA-AdaR.

4 Analysis

In this subsection, we address the following Research Questions (RQs) regarding AdaR: (1) Does training with AdaR enable the model to master adaptive reasoning? (2) To what extent does the scaling along different dimensions (query template T, variable set x, and problem-solving logic L) influence the model's performance respectively? (3) During the RLVR stage, when employing synthetic data whose templates are unseen by the target LLM, does AdaR continue to exhibit strong performance? (4) To what extent does diversity in the computational difficulty levels of synthetic queries affect the model's performance? (5) Is AdaR applicable to Instruct models? To answer these RQs (excluding RQ5), we conduct the following analysis using Qwen2.5-MATH-7B as the base model. Because the AIME evaluation metric differs from those of the other out-of-domain benchmarks, we exclude AIME from the analysis for better reporting.

4.1 AdaR Enables Adaptive Reasoning

Spurious reasoning causes generalization deficiencies. Spurious reasoning that relies on superficial features can derive answers without adhering to a correct logical order. By contrast, adaptive reasoning requires strict adherence to the correct logical order to derive answers step by step. To quantify this effect, we introduce the metric, Influence to Logical Order (ILO), which measures relative change rate in perplexity (PPL) of correct answers when the sentence order of the correspond-

ing CoT is shuffled.

$$\text{ILO} = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|\text{PPL}(y \mid q, z) - \text{PPL}(y \mid q, R^i(z))\right|}{\text{PPL}(y \mid q, z)}, \tag{2}$$

where q denotes the query, z denotes the CoT, y denotes the answer, R denotes a random permutation function operating at the sentence level, and n denotes the number of random permutations (set to 5 to keep variance small). We find that the ILO of CoTs that are unable to generate correct answers across all perturbations of variable values within a specific query template is significantly lower than that of CoTs capable of producing correct answers across all such perturbations (114.24% vs. 221.87%). This result indicates that spurious reasoning (characterized by a lower ILO metric) is more likely to induce generalization deficiencies than adaptive reasoning (characterized by a higher ILO metric). Based on ILO, we show that AdaR significantly improves the adaptive reasoning capability over "Initial SFT" (150.49% vs. 119.22%).

Enhancing algebraic thinking. We further examine model outputs and observe that, despite no code data being provided during training, the proportion of CoTs containing structural code snippets (e.g. "... `B + H = 157 - (23 + 41). Substituting the value of H into the equation, we get` ...") increases from 55% to 90% after training with AdaR (please refer to Appendix A.7 for more details), indicating the emergence of algebraic thinking, treating unknown and known variables on equal footing and solving queries via variable calculation. Furthermore, we provide a attention heatmap in Appendix A.5 and a theoretical analysis in Appendix A.6 to justify this phenomenon. We then likewise evaluate reasoning ability by measuring the accuracy of problem-solving code generation which engages algebraic thinking. As shown in Table 3, although RLVR is employed in both, training on standard data (Standard-RLVR) decreases accuracy, whereas training on our synthetic data with queries which are semantically similar but instantiate different variables improves accuracy.

4.2 Effect of Scaling Different Dimensions

In AdaR, we modulate sampling frequency to scale data size along query template T and values in variable set x. In addition, we scale data size along problem-solving logic L by subsampling, which

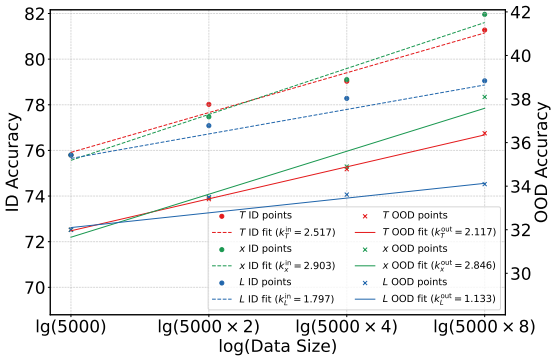


Figure 3: Performance of scaling.

regulates diversity in common scenarios Accordingly, we evaluate performance trends when scaling x, T or L and show the result in Figure 3. We observe steady improvements in model performance with increasing data size in both in-domain (ID) and out-of-domain (OOD) settings. To characterize marginal returns with scale, we fit the scaling curves with a log-linear approximation. The fitted coefficients satisfy $k_L < k_T < k_x$ throughout, indicating that scaling along the dimension x yields the greatest marginal returns. This is mainly because perturbing T facilitates query comprehension, whereas perturbing x is the essential driver of adaptive reasoning. We attribute the limited effectiveness of scaling along L to the simple subsampling, which causes data from same distribution constrains the control of diversity. We leave more comprehensive exploration of L to future work.

4.3 Effect of Seen Perturbed Queries

AdaR learns adaptive reasoning by comparing feedback on responses to perturbed queries that preserve the same problem-solving logic; hence, such queries should be included during RLVR. We identify two sources of comparison.

First, SFT may induce spurious reasoning by memorizing superficial features. When faced with perturbations of queries which are seen during SFT, the model produces unstable rollouts, yielding contrasting feedback. This is supported by Table 4, where training AdaR on perturbed data conditioned on SFT seed data outperforms "Standard-RLVR" trained on normal data without contrasts.

Second, contrast can also arise directly during RLVR: perturbed queries with shared logic but different values expose spurious reasoning through inconsistent feedback. To test this, we sample a non-overlapping 2.25K subset from ORCA-MATH and apply four rounds of *controllable perturbation*, denoted "AdaR-Lite×4". As shown in Table 4, this

Method	# Samples	In-Domain	Out-of-Domain
Standard-RLVR	9K	77.82	37.20
AdaR-Lite×4	2.25K × 4	83.42	49.90
AdaR	9K	84.72	53.88

Table 4: Effect of seen perturbed queries.

setting also outperforms "Standard-RLVR", further supporting our hypothesis.

4.4 Effect of Perturbation Magnitude

We investigate the influence of perturbation magnitude and present the results in Figure 4. We observe that the performance increases with larger perturbation in variable values up to a point and then exhibits a slight decline. Larger perturbation magnitude enables the model to learn previously unseen numerical calculations, and thereby improves performance. However, an overly large α increases the likelihood that perturbed variable sets violate implicit inter-variable constraints, producing more invalid candidates before the *sanity check*. If we suppose the success rate of sanity check is limited, these invalid candidates introduce additional noise, ultimately degrading model performance.

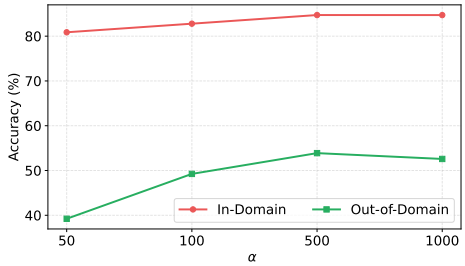


Figure 4: Influence of perturbation magnitude.

4.5 AdaR is Applicable to Instruct Model

We further investigate using Qwen3-4B-Instruct-2507 (Yang et al., 2025) as the base model. AdaR still yields substantial performance gains (+5.86) and effectively mitigates robustness failures that persist under test-time scaling (+1.51). We additionally observe that the model's rollout time on seed dataset is shorter than that of AdaR ($\approx 0.36\times$), suggesting that generating correct answers becomes more difficult when the model is exposed to mild numerical perturbations, revealing pronounced brittleness. Taken together, these findings highlight a pronounced limitation in advanced LLMs: strong performance on conventional test sets often obscures fundamental deficiencies in generalization and robustness. In contrast, AdaR effectively exposes and mitigates such latent deficiencies, demonstrating its broad applicability. Additional details are provided in Appendix A.9.

5 Related Work

Analysis of Robustness and Generalization. Wang et al. (2023) discovered that even invalid CoT demonstrations achieve comparable performance to valid ones. Similarly, we demonstrate that CoT arising from spurious reasoning is less sensitive to random variations in logical order. LLMs exhibit brittleness when facing problem variations: the GSM-SYM benchmark (Mirzadeh et al., 2024) demonstrates poor robustness on in-domain tasks with altered numerical values, while Jahin et al. (2025) shows limited generalization to out-of-domain problems, indicating models may memorize patterns rather than genuinely understand reasoning. The generation of GSM-SYM relies on human effort. In contrast, AdaR is fully automatic.

Approaches to Improving Reasoning Ability. Data synthesis has emerged as a promising direction (Wang et al., 2025). Prior work diversifies question templates via paraphrasing (Li et al., 2023) or LLM-based generation and verification (Lu et al., 2024), but often preserves the underlying mathematical structures or relies on error-prone verification, limiting gains in adaptive reasoning. Our proposed notion of adaptive reasoning helps explain these limitations. Concurrently, Liang et al. (2025) synthesizes queries online from model responses and back-propagates training loss to guide data generation. Our method is complementary: incorporating the model under training into AdaR for template and code generation, combined with such online training, may further improve performance.

6 Conclusion

Robustness and generalization remain central challenges for LLMs when solving mathematical problems. We attribute these failures to spurious reasoning that relies on superficial features and encourage adaptive reasoning that can adapt to varying variable values. Therefore, we propose AdaR, a framework that enables adaptive reasoning and comprises a data synthesis component and a model training component. Experimental results demonstrate substantial performance improvements with a small amount of data on both in-domain and out-of-domain tasks. Further analyses indicate that AdaR indeed facilitates adaptive reasoning and is a scalable and broadly applicable framework.

Limitations

Our framework is subject to several limitations. Firstly, although Section 4.2 underscores the importance of scaling L, AdaR currently enhances only T and x. We therefore identify scaling the diversity of L as a promising direction for future work. Secondly, AdaR requires the entire problem-solving process of each seed queries to be expressible as a fully executable program, and it assumes the presence of easily perturbed input variables. These requirements implies applying AdaR in more general settings (e.g., theorem proving) would require more sophisticated designs. Thirdly, due to computational constraints, we do not investigate large reasoning models or models with larger parameter counts as base.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023. Theoremqa: A theorem-driven question answering dataset. *arXiv preprint arXiv:2305.12524*.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. 2025. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. 2024. Navigate through enigmatic labyrinth a survey of chain of thought reasoning: Advances, frontiers and future. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1173–1203.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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*.

Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards reasoning in large language models: A survey. In *61st Annual Meeting of the Association for Computational Linguistics, ACL 2023*, pages 1049–1065. Association for Computational Linguistics (ACL).

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and 1 others. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55.

Afrar Jahin, Arif Hassan Zidan, Wei Zhang, Yu Bao, and Tianming Liu. 2025. Evaluating mathematical reasoning across large language models: A fine-grained approach. *arXiv preprint arXiv:2503.10573*.

Feiyang Kang, Michael Kuchnik, Karthik Padthe, Marin Vlastelica, Ruoxi Jia, Carole-Jean Wu, and Newsha Ardalani. 2025. Quagmires in sft-rl post-training: When high sft scores mislead and what to use instead. *arXiv preprint arXiv:2510.01624*.

Carolyn Kieran. 2004. Algebraic thinking in the early grades: What is it. *The mathematics educator*, 8(1):139–151.

Chengpeng Li, Zheng Yuan, Hongyi Yuan, Guanting Dong, Keming Lu, Jiancan Wu, Chuanqi Tan, Xiang Wang, and Chang Zhou. 2023. Mugglemath: Assessing the impact of query and response augmentation on math reasoning. *arXiv preprint arXiv:2310.05506*.

Xiao Liang, Zhongzhi Li, Yeyun Gong, Yelong Shen, Ying Nian Wu, Zhijiang Guo, and Weizhu Chen. 2025. Beyond pass@ 1: Self-play with variational problem synthesis sustains rlvr. *arXiv preprint arXiv:2508.14029*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys*, 55(9):1–35.

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

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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2732–2747.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. 2024. Mathscale: Scaling instruction tuning for mathematical reasoning. *arXiv preprint arXiv:2403.02884*.

Fengwei Teng, Quan Shi, Zhaoyang Yu, Jiayi Zhang, Yuyu Luo, Chenglin Wu, and Zhijiang Guo. 2025. Atom of thoughts for markov llm test-time scaling. *arXiv preprint arXiv:2502.12018*.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. 2024. Dart-math: Difficulty-aware rejection tuning for mathematical problem-solving. *Preprint*, arXiv:2407.13690.

Vladimir Vapnik. 2013. *The nature of statistical learning theory*. Springer science & business media.

Boshi Wang, Sewon Min, Xiang Deng, Jiaming Shen, You Wu, Luke Zettlemoyer, and Huan Sun. 2023. Towards understanding chain-of-thought prompting: An empirical study of what matters. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2717–2739, Toronto, Canada. Association for Computational Linguistics.

Zaitian Wang, Pengfei Wang, Kunpeng Liu, Pengyang Wang, Yanjie Fu, Chang-Tien Lu, Charu C. Aggarwal, Jian Pei, and Yuanchun Zhou. 2025. A comprehensive survey on data augmentation. *Preprint*, arXiv:2405.09591.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022a. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022b. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Liang Wen, Yunke Cai, Fenrui Xiao, Xin He, Qi An, Zhenyu Duan, Yimin Du, Junchen Liu, Tanglifu Tanglifu, Xiaowei Lv, and 1 others. 2025. Light-r1: Curriculum sft, dpo and rl for long cot from scratch and beyond. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 318–327.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, and 1 others. 2024. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

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*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

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

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

A Appendix

A.1 The use of LLMs

We use LLMs (e.g., GPT-5) only to polish writing. More specifically, their application focuses on two key areas: correcting grammatical errors and suggesting more appropriate word choices to enhance expression. Additionally, we conduct a thorough double check of all content refined by LLMs. This verification process is critical to preventing the inclusion of harmful information and ensuring the overall accuracy, reliability, and appropriateness of the final content.

A.2 Training Objective

A.2.1 SFT & RFT

Supervised fine-tuning (SFT) is a mainstream post-training strategy for eliciting step-by-step reasoning. Let π_θ denote a policy over responses given queries, parameterized by θ. Given a dataset $\mathcal{D}$ of query–response pairs (q, r), SFT minimizes the negative log-likelihood objective

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(q,r)\sim\mathcal{D}}\big[\log \pi_\theta(r \mid q)\big], \quad (3)$$

Reject sampling Fine-Tuning (RFT) constructs an SFT dataset by sampling from the model to be trained and retaining high-scoring outputs.

A.2.2 RLVR

Reinforcement Learning with Verifiable Reward (RLVR) is a reinforcement learning method that saves video memory usage. Given a query q, the model samples $r \sim \pi_\theta(\cdot \mid q)$ and a verifier $v(q, r) \in [0, 1]$ evaluates it. RLVR maximizes

$$J(\theta) = \mathbb{E}_{q\sim\mathcal{D}, r\sim\pi(\cdot|q)}[v(q, r)] \quad (4)$$

In mathematical reasoning, v typically checks whether the predicted answer exactly matches the gold answer.

A.3 Prompts

We show the prompts used for generating the query template and problem-solving code in Prompt 1, Solution Existence in Prompt 2, code generation which evaluates reasoning ability in Prompt 3, code execution which evaluates computational ability in Prompt 4.

A.4 General Settings

A.4.1 Data Synthesis

Given that most publicly available math word problem datasets are constructed based on GSM8K (Cobbe et al., 2021), and considering that existing LLMs may have been trained on substantial variations of this dataset, which alleviates to some extent the model's shortcut learning issues, we have opted for a newer and larger dataset ORCA-MATH (Mitra et al., 2024). We sample the data for synthesis by uniformly selecting using 42 as the random seed.

We use Qwen2.5-72B for data synthesis, including problem-solving codes, query templates, the existence of valid queries, and implementations of other baselines. The temperature is set to 0.7, the top p is set to 0.95, and the maximum generation length is 4096 tokens.

In addition, we experiment with using Qwen2.5-7B for data synthesis under the same parameter settings, with the results reported in the Table 7. Although employing a weaker model for data synthesis leads to a reduction in overall model performance, we observe that by scaling data size along the variable value x, the performance gap can be reduced from 8.14 to 3.09.

A.4.2 Training

All baselines are first trained with the same SFT procedure, after which RLVR training is performed using their respective data. For SFT, we use LLaMA-Factory (Zheng et al., 2024). All models are fine-tuned for 3 epochs with a batch size of 128 on 4 NVIDIA A100 GPUs. The peak learning rate is 1e-5 with a linear warm-up over the first 3% of training steps, followed by cosine decay to a minimum of 1e-7. The maximum generation length is set to 4096 tokens. For DAPO, we use veRL (Sheng et al., 2024), adopting the training setup of Yu et al. (2025). We utilize the AdamW optimizer (Loshchilov and Hutter, 2017) with a constant learning rate of 1e-6 and a linear warm-up over 20 rollout steps. For rollout, the prompt batch size is 256 and we sample 16 responses per prompt. For training, the mini-batch size is 32, corresponding to 128 gradient updates per rollout step. Overall, SFT training requires approximately 12 hours and RL training requires approximately 2 days.

A.4.3 Evaluation

We compare AdaR with baselines on the following 7 benchmarks:

- **GSM8K** [MIT] (Cobbe et al., 2021): The test set comprises 1,319 high-quality grade-school

mathematics word problems, each requiring between 2 and 8 reasoning steps.

- **ORCA-AdaR-test**: The test set consists of 2,500 high-quality grade-school mathematics word problems synthesized by AdaR. Seed problems are selected from ORCA [MIT License] (Mitra et al., 2024) and subjected to 1-4 numerical perturbations to assess models' robustness in mathematical reasoning.

- **GSM-SYM** [CC-BY-4.0] (Mirzadeh et al., 2024): The test set includes three subsets—*main*, *p1*, and *p2*—with 5,000, 5,000, and 2,500 instances, respectively. Starting from the GSM8K test problems, a symbolic template is constructed for each problem; each template yields 50 instances. The subsets augment the original logical structure by adding 0 (*main*), 1 (*p1*), or 2 (*p2*) additional reasoning clauses, thereby enabling a more rigorous evaluation of mathematical reasoning robustness.

- **MATH** [MIT] (Hendrycks et al., 2021): The test set comprises 5,000 problems drawn from high-school mathematics competitions. Problems are categorized into seven types (Prealgebra, Intermediate Algebra, Algebra, Precalculus, Geometry, Counting & Probability, and Number Theory) and five difficulty levels.

- **CollegeMath** (Tang et al., 2024): The test set contains 2,818 college-level problems curated from nine college-level mathematics textbooks, covering seven key disciplines: Algebra, Precalculus, Calculus, VectorCalculus, Probability, LinearAlgebra, and Differential Equations.

- **TheoremQA** (Chen et al., 2023): A theorem-driven question-answering benchmark containing 800 problems grounded in 350 theorems, designed to evaluate LLMs' ability to apply domain-specific theorems across Mathematics, Physics, Electrical Engineering, Computer Science, and Finance.

- **AIME 2025**: A test set containing 30 problems from the 2025 American Invitational Mathematics Examination (AIME), curated to evaluate LLMs on challenging, Olympiad-level high-school mathematics across Algebra, Geometry, Number Theory, and Combinatorics.

We adopt the evaluation pipeline of Tong et al. (2024) with some modifications. Unless otherwise noted, we set the sampling temperature to 0.7 and the nucleus parameter top_p to 0.9.

Regarding evaluation metrics, pass@1 is defined as the accuracy of the first sampled output, whereas avg@32 is the mean accuracy computed over 32 sampled outputs. As shown in Table 5, when the test set is sufficiently large (approximately 1,000 samples), the performance differences between pass@1 and avg@32 remain within 1 percentage point and the avg@32 results fully consistent with the conclusions drawn from the avg@1 results.

Method	GSM8K	GSM-SYM-main	MATH
Initial SFT	81.43 / 80.42	74.26 / 73.89	42.84 / 42.57
Standard-RLVR	86.96 / 86.88	81.34 / 81.51	61.71 / 61.29
MetaMATH	84.61 / 83.80	79.78 / 78.88	66.92 / 66.29
AdaR	91.81 / 92.32	89.72 / 89.39	75.90 / 76.00

Table 5: pass@1 / avg@32 results across different benchmarks.

A.5 Attention Analysis

We further conduct an attention-based case study to examine how SFT may induce spurious reasoning, and how AdaR promotes adaptive reasoning. We use the example in Figure 1 for illustration. When generating the token "cube", the model is expected to attend to the query-template component "$\times A \times A \times A =$", which encodes the underlying problem-solving logic, rather than to the instantiated numerical values, such as 32 and 42592.

Figure 5 presents the attention heatmaps of different models. Before SFT, the base model mainly attends to the local expression "$2 \times A \times A \times A =$" when reasoning about the token "cube". After SFT, however, the regions with increased attention become irregular and less selective: they cover not only the relevant template information "$\times A \times A \times A =$", but also irrelevant numerical tokens such as 32. This suggests that SFT may encourage the model to associate reasoning steps with superficial numerical features, thereby introducing spurious reasoning.

In contrast, after training with AdaR, the model exhibits a more logic-aware attention pattern. Specifically, attention to the template component "$\times A \times A \times A =$" is substantially strengthened, while attention to variable-value tokens, such as 32 and 42592, is reduced. This indicates that AdaR

helps the model better disentangle the problem-solving logic from instantiated variable values.

Overall, this case provides intuitive evidence that AdaR promotes adaptive reasoning by enhancing the model's understanding of query template information that is causally related to the reasoning process.

A.6 Theoretical Analysis

We formalize a math problem as a triplet (T, x, L), where T is a problem template, x denotes instance-specific variables, and L is the problem-solving logic. A problem is generated as

$$q = T(x),$$

and the correct reasoning chain is $L(x)$. The model π is trained to predict $L(x)$ from q.

In standard training (Standard-RLVR), superficial features $s(x)$ may correlate with $L(x)$ under the data distribution $\mathcal{D}$, i.e.,

$$I_{\mathcal{D}}(s(x); L(x) \mid q) > 0,$$

allowing the model to increase $P(L \mid q)$ by exploiting superficial features rather than learning the underlying logic.

We construct a variable-perturbed distribution $\mathcal{D}_T$ by generating diverse x for each template T. Logically, the specific numerical values themselves should be independent of problem-solving logic; otherwise, the model may rely on numerical heuristics rather than reasoning. Therefore, we assume the conditional mutual information satisfies:

$$I_{\mathcal{D}_T}(s(x); L(x) \mid q) = 0.$$

Thus, superficial features are not predictive of the problem-solving logic within each template. Under statistical learning theory (Vapnik, 2013), the Bayes-optimal predictor is

$$\pi^*(q) = \mathbb{E}[L(x) \mid q].$$

Since $q = T(x)$, and $s(x)$ provides no additional information about $L(x)$ beyond T, any risk-minimizing predictor cannot systematically rely on superficial features. Instead, it must recover the instance-specific variables x from q and approximate the invariant mapping $L(x)$.

A.7 Structural Code Snippet

To calculate the frequency of structural code snippet in the responses, we inspect 20 responses to queries from GSM-SYM. We present cases of structural code snippets from "Initial-SFT" and AdaR in Figure 9.

A.8 Generation Cost

The generation costs for all aspects included in the *sanity check* are reported in Table 6. This experiment is conducted on a single compute node.

A.9 Performance on Instruct Models

Dataset. To evaluate whether AdaR remains effective on more challenging datasets, beyond the grade-school mathematics problems used in earlier experiments (i.e., ORCA dataset), we adopt the Light-R1 dataset (Wen et al., 2025). From this dataset, we sample 7.5K seed data and apply numerical perturbations 1–4 times, resulting in a 17K-instance training set, denoted as "R1-AdaR-train". To perform a fine-grained analysis of whether AdaR improves model robustness and to avoid potential of data leakage, we construct two test sets, each containing 1K instances: "seen", whose query templates appear in "R1-AdaR-train", and "unseen", whose templates do not appear in the "R1-AdaR-train".

Baselines. In this experiment, we introduce two strong baselines: "Qwen3-4B-Instruct-2507" and the test-time-scaling-enabled "Qwen3-4B-Thinking-2507". Because the Instruct model already possesses strong mathematical reasoning capabilities, we skip the SFT stage and directly perform RLVR training. The model trained using RLVR on the 7.5K seed data is referred to as "Standard-RLVR-Instruct", while the model trained with AdaR synthetic data is denoted "AdaR-Instruct".

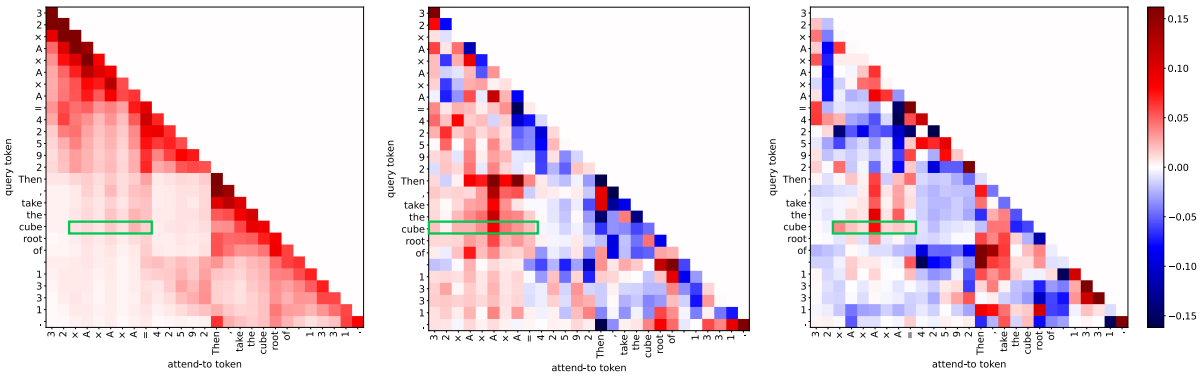
Table 8 shows the full results.

A.10 Training Dynamic Analysis

As shown in Figure 6 and Figure 7, during RLVR training, as the training steps increase, the model's generated CoT sequences grow progressively longer. In the early stage, although the sequence length steadily increases, the model rapidly acquires stable short reasoning patterns, often accompanied by certain forms of spurious reasoning, resulting in a decrease in entropy. As training continues and CoT sequences further lengthen, the model begins to develop more adaptive reasoning due to exposure to perturbed data. This transition increases the model's predictive uncertainty, leading to the subsequent rise in entropy.

In parallel, Figure 8 shows that the AIME 2024 scores exhibits a rapid improvement during the initial training stage and then plateaus with minor

fluctuations in later steps. This trajectory aligns with entropy dynamics: the model first stabilizes around short reasoning patterns that yield quick performance gains, and later explores a broader reasoning space as CoT sequences grow, leading to small oscillations around a higher performance level.



Base Changes after SFT Changes after AdaR

Figure 5: Attention heatmaps across different training stages. Green boxes highlight the attention weights associated with generating the token "cube".

Sanity Check	Generation Times (s per 1K samples)	Retention rate (%)
Variable Alignment	2.90	67.70
Executable Code	2.80	70.70
Existence of Valid Query	612	95.17

Table 6: Generation cost of component in *sanity check*.

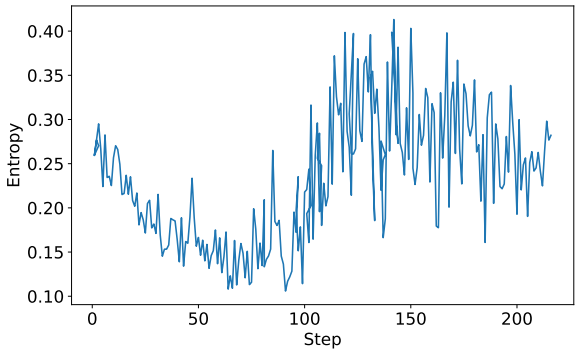


Figure 6: Entropy of actor model.

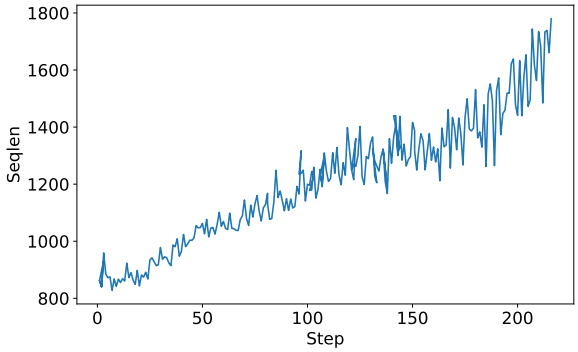


Figure 7: Generated sequence length of actor model.

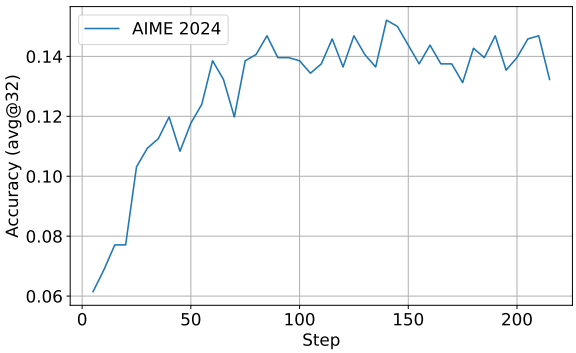


Figure 8: AIME 2024 scores on AdaR during training.

q_i: A group of 157 students has various hobbies. 23 like to play video games, 41 like to play volleyball, and the rest like to either hike or paint. How many like to hike if the number that like to paint is twice the number that prefer playing volleyball?
y_i: 11

z_i from "Inital-SFT":
... we multiply the number of volleyball players by 2:

Number of volleyball players = 41
Number of students who like to paint = 2 * 41 = 82
...
We subtract the sum of these from the total number of students to find out how many like to hike
...
Number of students who like to hike = 157 - (23 + 41 + 82)
...
Number of students who like to hike = 11...

z_i from AdaR:
Let's denote the number of students who like to paint as \(P \) and the number of students who like to hike as \(H\)
...
The number of students who like to hike is twice the number that prefer playing volleyball, so \(P = 2 \times 41 = 82 \)
...
Therefore, the number of students who like to hike and paint is the total number of students minus the number of students who like to play video games and play volleyball.

So, we can write the equation:
\[H + P = 157 - (23 + 41) \]

Substituting the value of \(P \) into the equation, we get:\n\[H + 82 = 157 - (23 + 41) \]
\[H + 82 = 157 - 64 \]
\[H + 82 = 93 \]
Now, we solve for \(H \):
\[H = 83 - 82 \]
\[H = 11 \]...

q_j: A group of 183 students has various hobbies. 42 like to play video games, 25 like to play volleyball, and the rest like to either hike or paint. How many like to hike if the number that like to paint is twice the number that prefer playing volleyball?
y_j: 66

z_j from "Inital-SFT":
... Since the number of students who like to paint is twice the number that prefer playing volleyball, we can calculate
...
Number of students who like to paint = 2 * 25 = 50
...
We can subtract these from the total to find out how many like to hike
...
Number of students who like to hike = 183 - (42 + 50)
...
Number of students who like to hike = 91...

z_j from AdaR:
Let's denote the number of students who like to paint as \(P \) and the number of students who like to hike as \(H \)
...
The number of students who like to paint is twice the number that prefer playing volleyball, so \(H = 2 \times 25 = 50 \)
...
Therefore, the number of students who like to hike or play volleyball is the total number of students minus the number of students who like to play video games and the number of students who like to play volleyball. This can be expressed as:
\[H + P = 183 - 42 - 25 \]

Substituting the value of \(P \) into the equation, we get:
\[H + 50 = 183 - 42 - 25 \]

Now, let's calculate the right-hand side of the equation:
\[183 - 42 = 141 \]
\[141 - 25 = 116 \]

So, we have:
\[H + 50 = 116 \]

To find \(H \), we subtract 50 from both sides of the equation:
\[H = 116 - 50 \]
\[H = 66 \]...

Figure 9: The case study of structural text in outputs. The green background indicates the correct reasoning step. The red background indicates the wrong reasoning step.

Method	# Samples	In-Domain					Out-of-Domain				AVG
		GSM8K	ORCA-AdaR	main	p1	p2	MATH	College	Theorem	AIME	
Initial SFT	$0K$	81.43	77.12	74.26	64.48	52.44	42.84	28.55	14.88	4.27	48.92
Qwen2.5-7B	$9K$	87.34	83.72	81.02	73.84	62.08	64.36	35.42	25.13	13.33	58.47
	$9K \times 4$	89.31	85.36	88.50	80.72	70.76	70.70	41.34	30.13	14.90	63.52
Qwen2.5-72B	$9K$	91.81	86.08	89.72	82.74	73.24	75.90	48.62	37.13	14.27	66.61
	$9K \times 4$	92.72	87.16	90.32	83.50	73.80	76.20	49.96	37.50	16.46	67.51

Table 7: Performance comparison when employing different models for data synthesis.

Method	Robustness			Generalization					AVG	CoT Length
	seen	unseen	GSM-SYM	GSM8K	MATH	College	Theorem	AIME		
Qwen3-4B-Instruct-2507	60.99	56.62	85.50	92.27	91.12	50.85	47.50	46.46	66.41	1672
Qwen3-4B-Thinking-2507	66.32	63.42	91.19	93.33	**95.38**	**53.97**	**58.13**	**75.83**	**74.70**	4784
Standard-RLVR-Instruct	62.86	57.72	90.47	93.78	91.52	52.09	48.25	52.50	68.65	2355
AdaR-Instruct	**68.42**	**65.17**	**91.88**	**94.69**	90.83	53.30	55.00	58.83	72.27	3435

Table 8: Comparison with the Instruct model and Thinking model.

Prompt 1: Generate Template and Code

Task Description:
You are given a natural language query and its chain-of-thought response.
Your task is to: Generate a Query Template by abstracting specific values into variables.
Generate Python Code that executes the logic described in the COT response using the abstracted variables.

Input Format:
Query: Original query with specific values
Response: Chain-of-thought reasoning that leads to the answer

Output Requirements:
Query Template:
Replace only concrete values in the query with angle-bracketed placeholders like <variable_name>. Do not replace names or general nouns (e.g., do not change "Jungkook" to <person_name>). Preserve the original wording and structure of the query as much as possible.
Python Code:
Begin by defining variables that correspond to the placeholders in the template. Translate the logic in the response into executable Python code. The code should end with a print() statement that prints only the final result. Do not include comments with explanations or reasoning. Use the same variable names as in the template for consistency.

=== START EXAMPLE ===

{example}

=== END EXAMPLE ===

Query:
{query}

Response:
{response}

Prompt 2: Existence of Valid Query

Task Description:
Your task is to generate a Chain-of-Thought (CoT) explanation that answers the user's question by reasoning through the logic implied in a provided Python script. Use the script to inform your explanation, but do not output or reproduce any code.

Input Format:
Query: A question involving specific values or conditions.
Python Code: A script that solves the query or provides a key computational procedure.

Output Requirements:
Start by interpreting the question clearly. Reason through the problem step by step, using the Python code as a guide to inform your logic. Refer to relevant steps in the code as part of your reasoning. Do not output or reference the code in any form. Explicitly state the final answer after the final step within \boxed{}.

Query:
{query}

Python Code:
{code}

Prompt 3: Code Generation

Please write a Python code to solve the following problem. Just give me the code, no explanation, no comments, no input statements. The code should be runnable and print the answer in the end.

Query:
{query}

Python Code:

Prompt 4: Code Execution

Please help me run the following Python code and return its output result instead of the code itself:
{code}