

Disagreement-Aware Repeated Sampling and Selective Rewriting for Complex Mathematical Reasoning

Anonymous ACL submission

Abstract

Large Reasoning Models (LRMs) have achieved remarkable performance in mathematical reasoning tasks, but they still struggle with hard samples. Existing test-time scaling methods, such as repeated sampling, self-correction, and tree search, can improve performance but are computationally inefficient and quickly plateau on challenging problems. Rewriting techniques, which reformulate problem expressions to enhance model understanding, have demonstrated effectiveness on challenging samples. However, they are often unnecessary or even harmful for easier samples. In this work, we propose a training-free framework that adaptively applies majority voting and rewriting based on model disagreement. The experiments conducted on seven mathematical benchmark datasets and three models show that our method improves accuracy on mathematical reasoning tasks by 3%-7%, while requiring fewer samples than existing methods.

1 Introduction

Large Reasoning Models (LRMs) have achieved remarkable results in mathematical reasoning tasks such as Olympiad (Yang et al., 2024b; Shao et al., 2024; DeepSeek-AI et al., 2024), but they still have shortcomings with truly difficult problems. A promising direction to close this gap is test-time scaling, an approach that trades more computation at inference for better answers. Key examples of this approach include repeated sampling (Wang et al., 2022; Li et al., 2024a; Lin et al., 2024; Gui et al., 2024), self-correction (Shinn et al., 2023; Chen et al., 2023; Li et al., 2024b; Kumar et al., 2024), and tree search (Yao et al., 2023; Zhang et al., 2024a; Bi et al., 2024; Gao et al., 2024). These techniques are certainly effective, but they hit a hard limit. As Xue et al. (2023) discovers, simply scaling up sampling or search is computationally inefficient, especially in difficult tasks

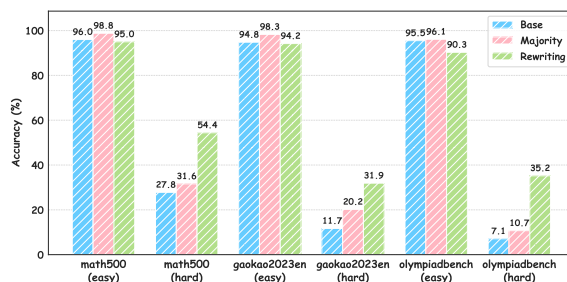


Figure 1: The effect of rewriting and majority voting (6 samplings) for Qwen3-8B on samples of different difficulty levels, where easy samples are those the model answers correctly on the first attempt, and hard samples are those it answers incorrectly on the first attempt.

where performance improvements quickly plateau. This reveals a fundamental inefficiency in current scaling strategies.

Therefore, we need a method that can improve the accuracy of the model in these difficult samples. Rewriting is an effective method of improving model performance without retraining, based on the principle that the model is highly sensitive to the way the problem is expressed. Research has proven its effectiveness on difficult samples. Zhou et al. (2024) uses a voting mechanism for rewriting to improve the robustness of the model. Zhang et al. (2025) rewrites the problem before generating code. Lai et al. (2025) goes further, forcing consistency in prompt, expression, and language.

However, rewriting inevitably introduces drawbacks. For example, changing the meaning of the original sample or giving incorrect conditions. As shown in the Figure 1, for easy samples, rewriting offers little help and may even be counterproductive, but it significantly improves the performance of hard samples. This reveals an interesting trade-off: majority voting works well for easy problems, but rewriting is key to solving difficult problems. In addition, we also find that in majority voting, samples that exhibit severe disagreement and produce

multiple different answers with a flat answer distribution are almost always answered incorrectly.

Based on this finding, we design a framework that requires no training, fine-tuning or external model assistance. This framework consists of three stages. The first stage (**Disagreement Filter**) filters out no disagreement samples through a self-consistency comparison. These samples do not require multiple rounds of sampling voting or rewriting to determine the answer, effectively reducing the total number of samples. The second stage (**Vote Resolve**) filters out minor disagreement samples through another round of self-consistency comparison and determines the final answer through multiple rounds of sampling and voting. The final stage (**Rewrite & Rethink**) rewrites and reconsiders the remaining severe disagreement samples to determine the final answer. This framework reduces resource consumption by filtering out samples that do not require additional sampling in the first stage, while the latter two stages focus on specific samples and apply appropriate methods to them, ensuring a better solution to the problem.

We conduct experiments on seven mathematical benchmarks and three large reasoning models. The experimental results show that our method improves the model’s accuracy in mathematical reasoning by about 3% to 7%, achieving significant progress, especially on benchmark tests where the original model performed poorly. Furthermore, our method uses only about 50% of the total sample size compared to other methods. We also conduct experiments to demonstrate that our method can generalize to code generation tasks. Finally, we analyze the rationale and advantages of each module design, as well as the applicability of the rewrite.

2 Related Work

2.1 Test-Time Scaling For Math Reasoning

Test-Time Scaling is a method that involves performing more computation during the inference phase to obtain better answers, and many studies have shown that it is effective for mathematical reasoning tasks. For example, by repeated sampling (Wang et al., 2022; Li et al., 2024a; Lin et al., 2024; Gui et al., 2024; Zhang et al., 2024b), which typically generates multiple reasoning paths and selects the final answer by majority voting across the sampled outputs, across multiple results to select one as the final answer. And utilize the model’s self-correction (Gao et al., 2025; Shinn et al., 2023;

Chen et al., 2023; Li et al., 2024b; Kumar et al., 2024), commonly implemented by prompting the model to reflect on and iteratively revise its initial reasoning, to modify those potentially incorrect answers accordingly. Tree search methods (Yao et al., 2023; Zhang et al., 2024a; Bi et al., 2024; Gao et al., 2024) explicitly explore a structured space of reasoning paths through branching and expansion, organizing intermediate reasoning steps into a tree structure. By systematically expanding, evaluating, and pruning candidate reasoning branches to identify high-quality solution paths. Test-time scaling has also been extended to incorporate reinforcement learning (Setlur et al., 2025; Zuo et al., 2025). These strategies aim to increase answer reliability in practice by leveraging multiple model outputs or iterative refinement mechanisms.

2.2 Rewriting

Rewriting is used in database systems to optimize SQL queries and improve retrieval efficiency (Liu and Mozafari, 2024). Shu et al. (2024) proposes RewriteLM, which uses instruction tuning and reinforcement learning to train LLMs to rewrite, combined with chain-of-thought prompting and reward models for quality evaluation. Ma et al. (2023) introduces a rewrite-retrieve-read framework, adding a query rewriting step before retrieval to bridge the semantic gap in retrieval-augmented LLMs. Li et al. (2024c) develops LLM-R2, training a contrastive model to distinguish semantically equivalent but more effective query rewrites. Kong et al. (2024) explores PRewrite, using reinforcement learning to automatically optimize prompts. Chen et al. (2025) proposes a question rewriter using Direct Preference Optimization to generate model-friendly question versions. There has also been research on rewriting in mathematical reasoning tasks. Zhou et al. (2024) has shown that the surface form of math problems significantly affects model performance, and it introduces SCoP, which rephrases problem statements and applies self-consistency for mathematical reasoning. Zhang et al. (2025) adopts a rewrite-then-execute strategy to improve the robustness of mathematical reasoning models. Lai et al. (2025) emphasizes multi-dimensional consistency, enforcing alignment across prompts and linguistic variants. These studies demonstrate diverse directions of rewrite-based reasoning enhancement methods, providing new ideas for improving the consistency and generalization ability of LLMs in mathematical reasoning.

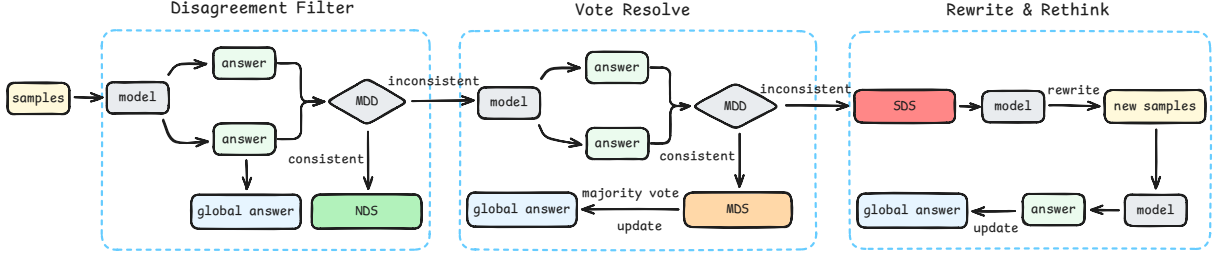


Figure 2: This figure illustrates the overall workflow of our method, where MDD, NDS, MDS, and SDS stand for Minimal Disagreement Detector, No Disagreement Samples, Minor Disagreement Samples, and Severe Disagreement Samples, respectively. The four model blocks in the figure all refer to the same model.

Algorithm 1 Minimal Disagreement Detector and Rewriting

Input: Sample S , model M , reasoning prompt P_{reason} , rewriting prompt P_{rewrite} , majority voting Maj

Output: global answer a^*

- 1: $a_1, a_2 \leftarrow M(S, P_{\text{reason}})$
 - 2: **if** $a_1 = a_2$ **then**
 - 3: **return** $a^* \leftarrow a_1$
 - 4: **else if** $a_1 \neq a_2$ **then**
 - 5: $a_3, a_4 \leftarrow M(S, P_{\text{reason}})$
 - 6: **if** $a_3 = a_4$ **then**
 - 7: **return** $a^* \leftarrow Maj(a_1, a_2, a_3, a_4)$
 - 8: **else if** $a_3 \neq a_4$ **then**
 - 9: $S_{\text{rewrite}} \leftarrow M(S, P_{\text{rewrite}})$
 - 10: $a_{\text{rewrite}} \leftarrow M(S_{\text{rewrite}}, P_{\text{reason}})$
 - 11: **return** $a^* \leftarrow a_{\text{rewrite}}$
 - 12: **end if**
 - 13: **end if**
-

3 Methodology

Our method framework is shown in Figure 2, which includes three stages (**Disagreement Filter**, **Vote Resolve**, **Rewrite & Rethink**). We use two minimal disagreement detectors to divide the samples into three categories based on the degree of divergence (No Disagreement Samples, Minor Disagreement Samples, and Severe Disagreement Samples), and use majority voting and rewriting strategies to determine the answers for Minor Disagreement Samples and Severe Disagreement Samples, respectively, and update them in the global answer. In the pseudocode of Algorithm 1, it demonstrates how a sample can determine the answer through our method. In Section 3.1, we describe the definition of the minimum divergence detector and the classification of samples. In Section 3.2, we discuss how the first two stages utilize the minimum divergence

detector to filter and partition samples, as well as the selection of iteration times. In Section 3.3, it is described how to determine the answer to Severe Disagreement Samples through rewriting.

3.1 Sample Categorization via the Minimal Disagreement Detector

We sample all samples n times and obtain a set of answer sets containing m answers for each sample. When the distribution of these m answers is sharp, with answer A being the most frequent, the majority voting can easily select A as the final answer, as multiple samplings indicate a stronger consensus of the model on answer A . However, if the distribution of m answers is flat, with no single dominant answer, at least one incorrect answer exists, and the reliability of the majority voting decreases, as the model exhibits significant disagreement at this point. Leveraging this disagreement, we design a **minimal disagreement detector (MDD)** that generates results a_1 and a_2 from a sample S using the minimal number of multiple samplings (two samplings). If $a_1 \neq a_2$, the sample S is identified as exhibiting minimal disagreement. As shown in Figure 2, samples that reach agreement through the MDD are not rewritten, and experiments in Figure 1 show that rewriting is less ineffective on correct samples. To minimize unnecessary rewriting of correct samples, samples that reach agreement through the MDD should have a high recall of correct samples. Table 6 shows that when more samples are used to determine consistency, a larger portion of actually correct samples is ignored, leading to a lower recall rate of correct samples. Therefore, it suffices to determine disagreement using only the results from two samplings.

We conduct multiple detections on sample S using the MDD. Based on the number of minimal

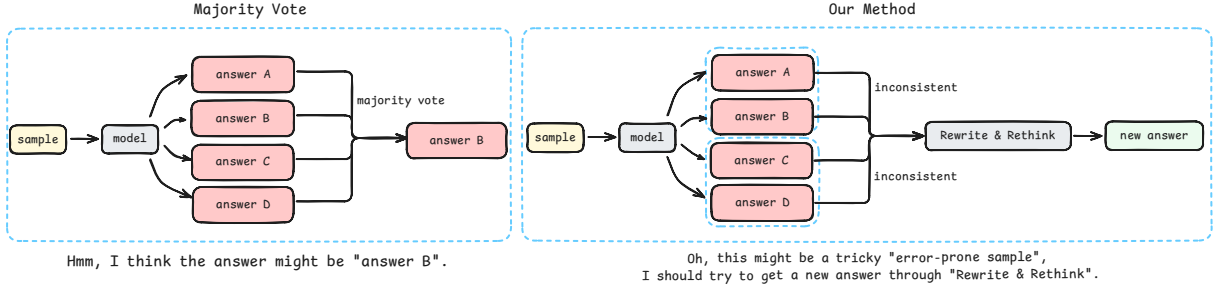


Figure 3: When the samples are too difficult to yield consistent answers. Our method utilizes the minimal disagreement detector to provide these samples with a second chance to think through a simple, self-contained checking process that requires no additional complex computation, rather than selecting one incorrect answer as the final output like majority voting does.

disagreements d_i observed for each sample S , we categorize the samples into three types:

No Disagreement Samples (NDS): $d_i = 0$. No disagreement occurs. The current answer exhibits high reliability, and no additional samplings are required to confirm it.

Minor Disagreement Samples (MDS): $d_i = 1$. Exactly one instance of disagreement occurs. The current answer is considered unreliable, and additional samplings are needed to reduce noise and obtain a more robust result.

Severe Disagreement Samples (SDS): $d_i \geq 2$. Two or more disagreements occur. The answer distribution is flat, and it is difficult to determine a reliable answer by simply increasing the number of samplings. In such cases, rewriting is required to determine the answer.

If the answers generated from two samplings are inconsistent, we consider that the model shows disagreement even with minimal sampling, making it unreliable to directly determine the answer. At this point, the sample is classified as either an MDS, which requires additional sampling for resolution, an SDS, for which increasing the number of samplings is insufficient to resolve the disagreement.

3.2 Iterative Filtering Strategy

To efficiently allocate computational resources and focus on the most uncertain samples, we adopt an iterative filtering strategy. This strategy categorizes samples into NDS, MDS, and SDS, and applies rewriting only to SDS, for which majority voting is insufficient to reliably determine the correct answer. As shown in Figure 1, rewriting significantly improves performance on incorrectly answered hard samples. Since rewriting is used to determine the final answers of SDS after iterative filtering, SDS

is required to retain a high recall of incorrect samples to ensure the effectiveness of rewriting. As shown in Figure 4, as the number of iterations increases, the recall of error samples decreases while the required sampling count rises. Therefore, we select two iterations for data filtering to balance effectiveness and efficiency.

As shown in Figure 2, during the first iteration (Disagreement Filter), the model performs two samplings for all samples and selects one of the sampling results to record in the global answer, which stores the determined answer for each sample. Simultaneously, the two sampling results are checked by the MDD. Samples with consistent answers are categorized as NDS, and their answers in the global answer remain unchanged. The remaining samples proceed to the second iteration.

During the second iteration (Vote Resolve), the model performs two samplings for each remaining sample and continues to use the MDD to check consistency. Samples showing agreement in this round are categorized as MDS. For these MDS, majority voting is applied to all available sampling results (i.e., the two from the first iteration plus the two from the second, making four in total) to determine the final answer, which is then updated in the global answer. The remaining inconsistent samples are categorized as SDS. These samples proceed to the next stage (Rewrite & Rethink), where the final answer is determined through rewriting. As shown in Figure 3, when samples exhibit persistent disagreement (i.e., multiple samplings fail to produce a consistent answer, indicating a flat rather than sharp answer distribution), our method uses MDD to identify samples that cannot be resolved by majority voting, and determines their final answers through rewriting and rethinking.

Model / Method	Math500	Gaokao En	Olympiad	GSM8K	AMC23	AIME25	AIME24	Avg
Qwen3-8B	84.4	77.7	46.4	95.7	65.0	23.3	36.7	61.3
Qwen3-8B (majority)	87.6	79.7	50.7	95.8	85.0	33.3	50.0	68.9
Qwen3-8B (SCoP)	88.2	79.5	52.6	95.4	82.5	33.3	40.0	67.4
Qwen3-8B (BoN)	85.8	75.6	49.0	95.1	65.0	33.3	40.0	63.4
Qwen3-8B (Ours)	92.8	82.9	61.9	95.8	87.5	43.3	66.7	75.8
Qwen3-4B	85.4	75.6	49.0	94.2	65.0	23.3	30.0	60.4
Qwen3-4B (majority)	88.6	79.5	53.2	95.5	82.5	26.7	50.0	68.0
Qwen3-4B (SCoP)	87.8	80.3	55.6	93.5	82.5	26.7	40.0	66.6
Qwen3-4B (BoN)	87.4	76.4	50.1	94.1	72.5	26.7	50.0	65.3
Qwen3-4B (Ours)	90.6	81.8	57.9	94.5	87.5	43.3	56.7	73.2
DS-Llama-8B	80.2	74.0	46.4	85.0	67.5	30.0	40.0	60.4
DS-Llama-8B (majority)	84.6	79.8	51.6	87.5	80.0	33.3	46.7	66.2
DS-Llama-8B (SCoP)	88.2	78.2	51.9	89.0	85.0	33.3	40.0	66.5
DS-Llama-8B (BoN)	79.0	68.8	45.3	78.8	75.0	30.0	43.3	60.1
DS-Llama-8B (Ours)	88.4	81.3	54.7	91.2	87.5	33.0	50.0	69.4

Table 1: This table presents the evaluation results of three models on seven mathematical benchmarks in the 8K-context setting. The models include Qwen3-8B, Qwen3-4B, and DS-Llama-8B (abbreviated from DeepSeek-R1-Distill-Llama-8B). Each model is evaluated under different reasoning enhancement strategies: baseline, majority voting, SCoP, BoN, and our proposed method.

3.3 Rewriting And Rethinking Mechanism

After the iterative filtering stage, the remaining samples are marked as SDS, indicating that the generated answers exhibit a high degree of uncertainty. For such samples, it is difficult to derive a reliable answer through majority voting. We therefore employ rewriting to determine the final answer. This method requires neither additional training nor the introduction of external models. Instead of generating more reasoning paths, the model is prompted to reformulate or reinterpret the surface form of the problem and then reattempt to solve it.

We apply the rewriting prompt template shown in Appendix C to each SDS sample for a single rewrite, thereby generating a new set of samples $S_{rewrite}$. For each sample in $S_{rewrite}$, we perform a single sampling to produce an answer $a_{rewrite}$, which is recorded in the global answer. This process, by reformulating the surface form of the question, allows the model to rethink the same problem from different expression, thereby attempting to break away from previously erroneous reasoning paths. With only minimal additional computation, it can significantly improve the accuracy.

4 Experiments

4.1 Experimental Settings

We evaluate our approach on seven widely used mathematical reasoning benchmarks, including GSM8K (Cobbe et al., 2021), Math500 (Hendrycks et al., 2021), Gaokao2023en (Zhang et al., 2023), Olympiadbench (He et al., 2024),

AMC23, AIME24, and AIME25, using three base models (Qwen3-4B (Team, 2025), Qwen3-8B, and DeepSeek-R1-Distill-Llama-8B (DeepSeek-AI, 2025)). And we use the accuracy of the answers as the evaluation metric.

To better understand the performance of our method, we select three representative methods for comparison: majority voting, Best of N with outcome reward model (BoN), and SCoP (Zhou et al., 2024). The first two methods are abbreviated as Majority and BoN in the following sections for convenience. All three methods perform six rounds of sampling in total: The Majority method samples six times per instance and selects the final answer through simple voting. The BoN method also samples six times per instance, then uses the Skywork-Reward-V2-Llama-3.1-8B-40M (Liu et al., 2025) model as an outcome reward model to score each answer and choose the one with the highest reward as the final answer; SCoP, on the other hand, performs one round of problem rewriting followed by five rounds of sampling, and finally selects the final answer through simple voting. Our experimental settings use a temperature of 0.6, a top- p value of 0.95, and a top- k value of 20. The model uses different prompt templates for reasoning and rewriting, with all other hyperparameters kept the same. We conduct experiments under two context lengths, 8K and 16K, and the results are presented in Table 1 and Table 4. Our experiments are conducted on the Qwen2.5-Math¹ evaluation framework.

¹<https://github.com/QwenLM/Qwen2.5-Math>

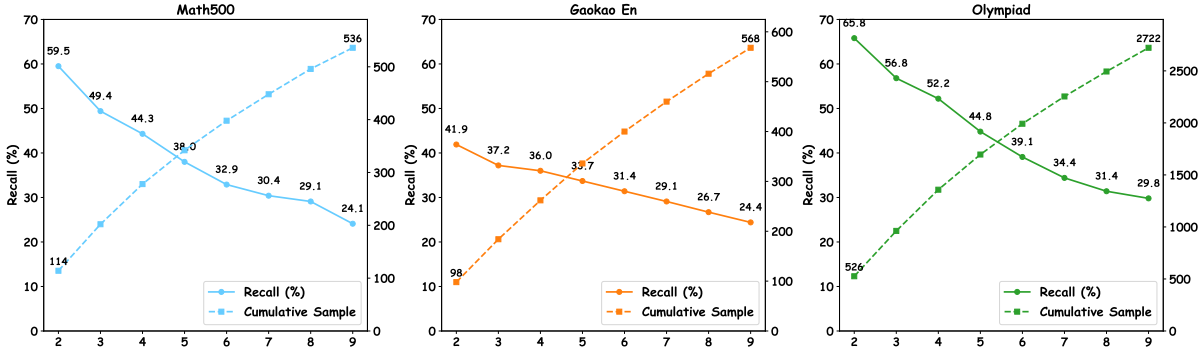


Figure 4: The figure shows the recall (%) of incorrect samples as a function of the number of iterations N on three datasets: Math500, Gaokao En, and Olympiad. Solid lines represent the recall achieved by our iterative filtering method, while dashed lines indicate the cumulative number of samples used at each iteration. The figure shows that as the number of iterations increases, the recall decreases while the cumulative number of samples increases.

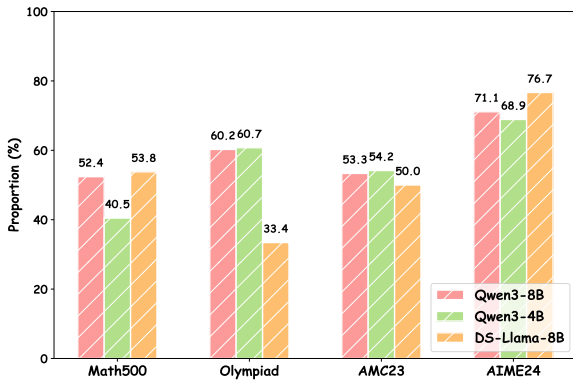


Figure 5: This figure illustrates the proportion of total samples required by our method compared to other methods in four mathematical benchmark tests. Each set of bars represents a dataset, with each bar indicating the relative proportion of total samples when our method is applied to Qwen3-8B, Qwen3-4B, and DeepSeek-R1-Distill-Llama-8B, respectively.

4.2 Main Results

The main experimental results are shown in Table 1. Our method outperforms baselines on almost all benchmarks across three models, achieving the highest average accuracy. Through experiments, we find that significant improvements are achieved on all benchmarks except GSM8K, while on GSM8K, the performance changes of Qwen3-8B and Qwen3-4B are not as pronounced as those of DeepSeek-R1-Distill-Llama-8B. This is because Qwen3-4B and Qwen3-8B already achieve high scores on GSM8K, with only about 5% of samples answered incorrectly, indicating that GSM8K contains only a small fraction of samples that cause the model to disagree. As shown in Figure 5, for Qwen3-4B and Qwen3-8B on the GSM8K bench-

mark, the SDS after two rounds of iterative filtering account for only 1.5% to 2% of the total samples, making rewriting yield almost no improvement; in contrast, for DeepSeek-R1-Distill-Llama-8B, SDS constitute 16% of the total samples, suggesting that the quantity of SDS influences the effectiveness of rewriting to some extent.

In addition, we record the number of samplings used in the experiments. We observe that, across three models, our method requires significantly fewer samplings than other methods while achieving comparable or even higher accuracy. We select four benchmarks (Math500, Olympiad, AMC23, and AIME24) for illustration and visualize the results in Figure 5. On Math500, Olympiad, and AMC23, our method uses approximately 30% to 60% of the samplings required by other methods, and even on the more challenging AIME24 benchmark, it uses only about 70% of the samplings. We also show in Figure 7 the average output tokens and wall-clock time for each sample.

4.3 Method Generalization Evaluation

To examine whether our method is limited to mathematical reasoning tasks, we further evaluate it on code generation tasks using the same hyperparameters as in the mathematical reasoning experiments. Specifically, we keep all sampling strategies and context lengths identical to those used in the math experiments to ensure a fair and consistent comparison. We adopt EvalPlus² as the evaluation framework and assess five models (Qwen3-8B, Qwen3-4B, DeepSeek-R1-Distill-Llama-8B, Qwen2.5-Coder-7B-Instruct (Yang et al., 2024a)

²<https://github.com/evalplus/evalplus>

Model / Method	HumanEval	HumanEval+	MBPP	MBPP+
Qwen3-8B	84.8	80.5	70.5	60.8
Qwen3-8B(Ours)	87.8	81.1	71.7	61.4
Qwen3-4B	80.5	75.0	69.0	60.9
Qwen3-4B(Ours)	81.7	75.0	69.8	61.1
DS-Llama-8B	44.5	37.8	31.7	26.7
DS-Llama-8B(Ours)	54.3	47.0	38.4	31.2
Qwen2.5-Coder-7B-Instruct	86.6	82.3	82.5	70.1
Qwen2.5-Coder-7B-Instruct(Ours)	89.0	83.5	87.8	73.5
Llama-3.1-8B-Instruct	65.9	57.3	68.5	56.6
Llama-3.1-8B-Instruct(Ours)	67.2	58.5	69.0	57.7

Table 2: This table presents the evaluation results of our method on code generation tasks. We evaluate five models (Qwen3-8B, Qwen3-4B, DeepSeek-R1-Distill-Llama-8B, Qwen2.5-Coder-7B-Instruct, and Llama-3.1-8B-Instruct) on two datasets (HumanEval and MBPP) as well as their enhanced variants (HumanEval+ and MBPP+).

Method	Math500	Olympiad	AMC23	AIME24
Baseline	84.4	46.4	65.0	36.7
OnlyRewrite	90.4	59.0	85.0	56.7
OnlyMajority	87.6	48.6	70.0	40.0
Full Method	92.8	61.9	87.5	66.7

Table 3: This table presents an ablation study of our method on the Qwen3-8B model across multiple mathematical benchmarks. The baseline row shows the performance of the original model without any enhancement. The OnlyRewrite variant applies only the selective rewriting mechanism, while the OnlyMajority variant applies only majority voting.

and Llama-3.1-8B-Instruct), on two code generation benchmarks (HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021)), as well as their enhanced variants (HumanEval+ and MBPP+).

In contrast to mathematical reasoning tasks, for code generation tasks we do not directly use answer equality as the criterion for disagreement; instead, we assess disagreement based on functional and logical equivalence of the generated programs. This approach accounts for the fact that two programs can produce the same correct results even if their implementations differ syntactically. Specifically, we evaluate consistency by executing both programs on the same set of test cases. If the outputs of the two programs match exactly across all test cases, we consider them functionally and logically equivalent. The experimental results are presented in Table 2.

5 Analysis and Discussion

5.1 Ablation Study

To verify the contribution of each component in our framework, we conducted an ablation study on the

Qwen3-8B. We sequentially present the experimental results of four settings: the model without any component (Baseline), the version with only the rewriting component (OnlyRewrite), the version with only the majority voting component (OnlyMajority), and the complete framework with all components integrated (Full Method).

As shown in Table 3, when applying only the rewriting mechanism OnlyRewrite, the model shows a significant improvement over the baseline, especially on more challenging benchmarks such as Olympiad and AIME24. This indicates that rewriting is more effective on difficult problems that tend to cause disagreement. The OnlyMajority also improves over the baseline, but the improvement is less pronounced than that of OnlyRewrite, likely because majority voting struggles to effectively resolve highly ambiguous or difficult samples with severe disagreement, whereas rewriting demonstrates stronger efficacy on such samples. In contrast, for easier problems, majority voting can often achieve high accuracy without rewriting, highlighting the task-dependent nature of each component. When combining both the rewriting and majority voting mechanisms, the method achieves the best performance, highlighting the synergy between the two components: the rewriting module improves accuracy on challenging tasks, while the majority voting mechanism filters out noise generated from ambiguous samples. Together, they enable the framework to achieve comprehensive improvements in both accuracy and reasoning stability.

5.2 Analysis of Rewriting Applicability

We categorize the change in correctness before and after rewriting into three cases: Effective Rewriting

(the rewrite turns an incorrect answer into a correct one), Ineffective Rewriting (the rewrite leaves an incorrect answer unchanged), and Harmful Rewriting (the rewrite turns an originally correct answer into an incorrect one). In addition, we provide representative cases in Appendix D.

Effective Rewriting We observe that most samples that can be successfully corrected through rewriting are not originally wrong due to insufficient computational ability, but rather because the problems contain implicit conditions or structural information (**hidden info error**). These hidden elements often cause the model to oscillate during the reasoning process: it struggles to quickly and accurately identify the underlying mathematical relationships and fails to stabilize on a correct solution path. In some cases, the model even constructs an incorrect equation early in the reasoning process due to overlooking implicit constraints.

The rewriting mechanism makes such hidden information explicit, restructures the problem statement, or directly provides a solution strategy. This substantially reduces the analytical burden required for reasoning, allowing the model to enter the computation phase without lengthy deliberation. The improvement is particularly significant on OlympiadBench, where many problems require long token sequences for structural parsing. Rewriting effectively compresses this reasoning chain, leading to notable gains in accuracy.

Ineffective Rewriting In another class of cases, rewriting offers no improvement. The errors in these situations primarily stem from: answer formats that do not conform to the required specifications (**answer format error**); the loss of context or intermediate variables in long reasoning chains (**context loss error**); and arithmetic errors in the calculation steps, such as incorrect orders of magnitude in astrophysics or microscopic atomic calculations (**calculation error**). These errors do not originate from implicit information in the problem statement. Therefore, even if rewriting provides a clearer version of the problem, the model is still likely to repeat the same formatting or arithmetic errors during its entire reasoning process.

Harmful Rewriting In our experiments, we find that rewriting is not always effective and can even cause originally correct answers to become incorrect. We identify an important reason for this: the original question is already simple, but the rewrit-

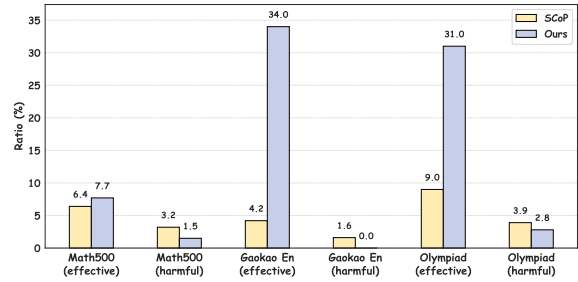


Figure 6: This figure shows, using Qwen3-8B on Math500, Gaokao En and Olympiad, a comparison of the proportions of effective and harmful rewrites between SCoP and our method.

ten version compresses the text excessively, removing or obscuring key conditions, which leads the model to reason incorrectly (**over-compression error**). Therefore, avoiding rewriting short and clear questions reduces the negative impact of rewriting.

We also compute, for both SCoP and our method, the proportions of rewritten samples for which rewriting is effective or harmful, as shown in Figure 6. Compared to SCoP, our method achieves a higher proportion of effective rewrites while reducing the proportion of harmful ones.

6 Conclusion

This work proposes a training-free framework that leverages the degree of disagreement to classify samples into three categories. It then uses majority voting and rewriting in a targeted manner to determine the answers for different types of samples, aiming to improve the mathematical reasoning performance of LRMs. The framework enables computational resources to focus more on unstable samples while avoiding, to some extent, the negative effects of rewriting. We conduct experiments using three models on seven mathematical reasoning benchmarks. The results show that our method significantly improves the accuracy of mathematical reasoning tasks while using fewer samplings than other methods. Additionally, the method can be extended to code generation tasks beyond mathematical reasoning. We also further analyze the cases where rewriting is effective, ineffective, and harmful. In future work, we plan to enhance the rewriting mechanism and explore a more fine-grained boundary between rewriting and majority voting, with the goal of further improving overall performance. Overall, this research provides a simple and effective direction for improving the performance of LRMs at inference time.

552 Limitations

553 This work leaves room for further exploration in
554 certain aspects.

- 555 • Rewriting is effective on incorrect samples,
556 but some problems remain unresolved, and we
557 do not explore whether improving the strategy
558 could further address them.
- 559 • Due to computational constraints, we do not
560 evaluate larger-scale models, and experiments
561 are limited to mathematical reasoning and
562 code generation tasks.
- 563 • We use the same model for both rewriting
564 and reasoning, but have not yet investigated
565 whether using different models collaboratively
566 could yield better results.

567 References

568 Jacob Austin, Augustus Odena, Maxwell Nye, Maarten
569 Bosma, Henryk Michalewski, David Dohan, Ellen
570 Jiang, Carrie Cai, Michael Terry, Quoc Le, and
571 Charles Sutton. 2021. [Program synthesis with large
572 language models](#). *Preprint*, arXiv:2108.07732.

573 Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and
574 Yunhe Wang. 2024. Forest-of-thought: Scaling test-
575 time compute for enhancing llm reasoning. *arXiv
576 preprint arXiv:2412.09078*.

577 Junhao Chen, Bowen Wang, Zhouqiang Jiang, and Yuta
578 Nakashima. 2025. Putting people in llms’ shoes:
579 Generating better answers via question rewriter. In
580 *Proceedings of the AAAI Conference on Artificial
581 Intelligence*, volume 39, pages 23577–23585.

582 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan,
583 Henrique Ponde de Oliveira Pinto, Jared Kaplan,
584 Harri Edwards, Yuri Burda, Nicholas Joseph, Greg
585 Brockman, Alex Ray, Raul Puri, Gretchen Krueger,
586 Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela
587 Mishkin, Brooke Chan, Scott Gray, and 39 others.
588 2021. [Evaluating large language models trained on
589 code](#). *Preprint*, arXiv:2107.03374.

590 Xinyun Chen, Maxwell Lin, Nathanael Schärli, and
591 Denny Zhou. 2023. [Teaching large language models
592 to self-debug](#). *Preprint*, arXiv:2304.05128.

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

599 DeepSeek-AI. 2025. [Deepseek-r1: Incentivizing rea-
600 soning capability in llms via reinforcement learning](#).
601 *Preprint*, arXiv:2501.12948.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingx-
uan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng,
Chong Ruan, Damai Dai, Daya Guo, Dejian Yang,
Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli
Luo, Guangbo Hao, Guanting Chen, and 138 others.
2024. [Deepseek-v2: A strong, economical, and effi-
cient mixture-of-experts language model](#). *Preprint*,
arXiv:2405.04434.

Kuofeng Gao, Huanqia Cai, Qingyao Shuai, Di-
hong Gong, and Zhifeng Li. 2025. [Embedding
self-correction as an inherent ability in large lan-
guage models for enhanced mathematical reasoning](#).
Preprint, arXiv:2410.10735.

Zitian Gao, Boye Niu, Xuzheng He, Haotian Xu,
Hongzhang Liu, Aiwei Liu, Xuming Hu, and Lijie
Wen. 2024. Interpretable contrastive monte carlo tree
search reasoning. *arXiv preprint arXiv:2410.01707*.

Lin Gui, Cristina Gârbaacea, and Victor Veitch. 2024.
Bonbon alignment for large language models and the
sweetness of best-of-n sampling. *Advances in Neural
Information Processing Systems*, 37:2851–2885.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding
Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han,
Yujie Huang, Yuxiang Zhang, and 1 others. 2024.
Olympiadbench: A challenging benchmark for pro-
moting agi with olympiad-level bilingual multimodal
scientific problems. In *Proceedings of the 62nd An-
nual Meeting of the Association for Computational
Linguistics (Volume 1: Long Papers)*, pages 3828–
3850.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul
Arora, Steven Basart, Eric Tang, Dawn Song, and Ja-
cob Steinhardt. 2021. Measuring mathematical prob-
lem solving with the math dataset. *arXiv preprint
arXiv:2103.03874*.

Weize Kong, Spurthi Hombiah, Mingyang Zhang,
Qiaozhu Mei, and Michael Bendersky. 2024.
Prewrite: Prompt rewriting with reinforcement learn-
ing. In *Proceedings of the 62nd Annual Meeting of
the Association for Computational Linguistics (Vol-
ume 2: Short Papers)*, pages 594–601.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal,
Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli,
Shariq Iqbal, Colton Bishop, Rebecca Roelofs, and
1 others. 2024. Training language models to self-
correct via reinforcement learning. *arXiv preprint
arXiv:2409.12917*.

Huiyuan Lai, Xiao Zhang, and Malvina Nissim. 2025.
Multidimensional consistency improves reasoning in
language models. *arXiv preprint arXiv:2503.02670*.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and
Deheng Ye. 2024a. [More agents is all you need](#).
Preprint, arXiv:2402.05120.

Loka Li, Zhenhao Chen, Guangyi Chen, Yixuan Zhang,
Yusheng Su, Eric Xing, and Kun Zhang. 2024b. [Con-
fidence matters: Revisiting intrinsic self-correction](#)

658	capabilities of large language models. <i>Preprint</i> , arXiv:2402.12563.	Mingfeng Xue, Dayiheng Liu, Wenqiang Lei, Xingzhang Ren, Baosong Yang, Jun Xie, Yidan Zhang, Dezhong Peng, and Jiancheng Lv. 2023. Dy- namic voting for efficient reasoning in large language models. In <i>Findings of the Association for Computa- tional Linguistics: EMNLP 2023</i> , pages 3085–3104.	712 713 714 715 716 717
660	Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024c. Llm-r2: A large language model enhanced rule-based rewrite sys- tem for boosting query efficiency. <i>arXiv preprint</i> arXiv:2404.12872.	An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Hao- ran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024a. Qwen2 technical report. <i>arXiv</i> <i>preprint arXiv:2407.10671</i> .	718 719 720 721 722 723 724
665	Lei Lin, Jiayi Fu, Pengli Liu, Qingyang Li, Yan Gong, Junchen Wan, Fuzheng Zhang, Zhongyuan Wang, Di Zhang, and Kun Gai. 2024. Just ask one more time! self-agreement improves reasoning of lan- guage models in (almost) all scenarios. <i>Preprint</i> , arXiv:2311.08154.	An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jian- hong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. 2024b. Qwen2.5-math tech- nical report: Toward mathematical expert model via self-improvement. <i>Preprint</i> , arXiv:2409.12122.	725 726 727 728 729 730 731
671	Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiakai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, Yang Liu, and Yahui Zhou. 2025. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. <i>arXiv preprint</i> arXiv:2507.01352.	Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. <i>Advances in neural information processing systems</i> , 36:11809–11822.	732 733 734 735 736
677	Jie Liu and Barzan Mozafari. 2024. Query rewrit- ing via large language models. <i>arXiv preprint</i> arXiv:2403.09060.	Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. 2024a. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. <i>arXiv preprint</i> arXiv:2406.07394.	737 738 739 740 741
680	Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval- augmented large language models. In <i>Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing</i> , pages 5303–5315.	Ruiqi Zhang, Momin Haider, Ming Yin, Jiahao Qiu, Mengdi Wang, Peter Bartlett, and Andrea Zanette. 2024b. Accelerating best-of-n via speculative rejec- tion. In <i>ICML 2024 Workshop on Structured Probabilistic Inference</i> {&} <i>Generative Modeling</i> .	742 743 744 745 746
685	Amrith Setlur, Nived Rajaraman, Sergey Levine, and Aviral Kumar. 2025. Scaling test-time compute with- out verification or rl is suboptimal. <i>arXiv preprint</i> arXiv:2502.12118.	Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. 2023. Evaluating the performance of large language models on gaokao benchmark. <i>arXiv preprint arXiv:2305.12474</i> .	747 748 749 750
689	Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathemati- cal reasoning in open language models. <i>Preprint</i> , arXiv:2402.03300.	Yu Zhang, Shujun Peng, Nengwu Wu, Xinhan Lin, Yang Hu, and Jie Tang. 2025. Rm-pot: Reformulating mathematical problems and solving via program of thoughts. <i>arXiv preprint arXiv:2502.12589</i> .	751 752 753 754
695	Noah Shinn, Federico Cassano, Edward Berman, Ash- win Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal rein- forcement learning. <i>Preprint</i> , arXiv:2303.11366.	Yue Zhou, Yada Zhu, Diego Antognini, Yoon Kim, and Yang Zhang. 2024. Paraphrase and solve: Exploring and exploiting the impact of surface form on math- ematical reasoning in large language models. <i>arXiv</i> <i>preprint arXiv:2404.11500</i> .	755 756 757 758 759
699	Lei Shu, Liangchen Luo, Jayakumar Hoskere, Yun Zhu, Yinxiao Liu, Simon Tong, Jindong Chen, and Lei Meng. 2024. RewritelM: An instruction-tuned large language model for text rewriting. In <i>Proceedings of the AAAI Conference on Artificial Intelligence</i> , volume 38, pages 18970–18980.	Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xin- wei Long, Ermo Hua, and 1 others. 2025. Ttrl: Test-time reinforcement learning. <i>arXiv preprint</i> arXiv:2504.16084.	760 761 762 763 764
705	Qwen Team. 2025. Qwen3 technical report. <i>Preprint</i> , arXiv:2505.09388.		
707	Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. <i>arXiv</i> <i>preprint arXiv:2203.11171</i> .		

A Experimental Results

765

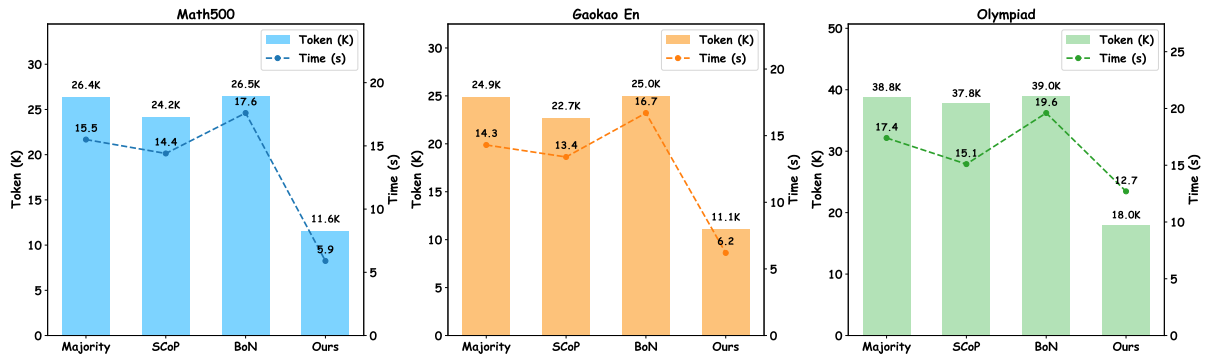


Figure 7: Average tokens and wall-clock time per sample for Qwen3-8B using four methods on three benchmarks.

Model / Method	Math500	Gaokao En	Olympiad	GSM8K	AMC23	AIME25	AIME24	Avg
Qwen3-8B	91.6	82.9	60.7	94.9	90.0	46.7	66.7	76.2
Qwen3-8B (majority)	93.9	84.2	64.7	96.4	95.0	53.3	70.0	79.6
Qwen3-8B (SCoP)	92.4	84.5	65.6	95.2	95.0	53.3	63.3	78.5
Qwen3-8B (BoN)	93.4	83.6	64.6	95.7	90.0	53.3	63.3	77.7
Qwen3-8B (Ours)	95.2	86.0	67.3	96.1	95.0	66.7	80.0	83.8
Qwen3-4B	91.8	83.6	63.4	95.0	92.5	46.7	66.7	77.7
Qwen3-4B (majority)	95.2	84.9	66.5	95.0	95.0	60.0	70.0	81.0
Qwen3-4B (SCoP)	92.0	84.7	64.6	93.6	92.5	46.7	60.0	76.3
Qwen3-4B (BoN)	92.8	82.1	64.1	94.8	90.0	60.0	66.7	78.4
Qwen3-4B (Ours)	95.6	85.7	67.1	95.1	97.5	70.0	70.0	83.0
DS-Llama-8B	82.4	73.8	54.1	85.0	87.5	33.3	40.0	65.2
DS-Llama-8B (majority)	88.8	80.8	58.1	89.0	87.5	40.0	56.7	71.6
DS-Llama-8B (SCoP)	88.4	78.2	55.3	89.3	87.5	33.3	43.3	67.9
DS-Llama-8B (BoN)	77.2	69.4	47.1	80.5	75.0	36.7	50.0	62.3
DS-Llama-8B (Ours)	91.0	81.2	58.5	90.8	92.5	43.3	60.0	73.9

Table 4: This table presents the evaluation of three models on seven mathematical benchmarks with a 16K-context.

Model / Samples	Math500	Gaokao En	Olympiad	GSM8K	AMC23	AIME25	AIME24
Qwen3-8B (NDS)	332	242	373	1287	26	8	12
Qwen3-8B (MDS)	50	39	59	12	4	4	2
Qwen3-8B (SDS)	118	104	243	20	10	18	16
Qwen3-4B (NDS)	437	313	373	1267	27	8	13
Qwen3-4B (MDS)	19	27	49	28	1	2	2
Qwen3-4B (SDS)	44	45	253	24	12	20	15
DS-Llama-8B (NDS)	316	209	291	918	28	6	8
DS-Llama-8B (MDS)	61	66	91	188	4	0	5
DS-Llama-8B (SDS)	123	110	293	213	8	24	17

Table 5: Sample sizes of NDS, MDS, and SDS for three models on seven benchmarks with an 8K-context.

Consistency Threshold (n)	Math500	Gaokao En	Olympiad	GSM8K
n=2	94.5	96.2	87.5	99.0
n=3	91.4	94.9	83.2	97.2
n=4	89.1	89.6	77.9	97.1
n=5	84.5	87.2	72.4	96.9
n=6	82.7	85.6	69.5	96.1

Table 6: Recall of correct samples under different consistency levels (**n**) across multiple generations. Higher **n** means stricter consistency, requiring at least **n** generated answers to agree. As **n** increases, recall decreases, indicating stricter constraints filter out some correct but non-deterministic cases.

B Theoretical Analysis

To better understand how consistency judgments reflect the sharpness or flatness of the answer distribution, we conduct a theoretical analysis.

We denote the probability of the most frequent answer in multiple model samplings as P , and the total probability of all other non-most-frequent answers as $1 - P$. In two samplings:

- The probability of obtaining the same most frequent answer twice is P^2 .
- The probability of obtaining the same non-most-frequent answer twice is $\leq \sum_i q_i^2 \leq (1 - P)^2$, where q_i is the probability of each non-most-frequent answer.
- The probability of obtaining inconsistent answers is

$$1 - (P^2 + \sum_i q_i^2) \geq 2P(1 - P)$$

When the answer distribution is sharp, the probability P of the most frequent answer approaches 1, and $1 - P$ approaches 0. In this case, P^2 is large and $(1 - P)^2$ is small, so obtaining a consistent answer is more likely to correspond to the most frequent answer. Meanwhile, the probability of inconsistent answers $2P(1 - P)$ also decreases, since $1 - P$ is small and the chance of observing non-most-frequent answers is low.

When the answer distribution is flat, P is close to the average probability $\frac{1}{n}$ (with $n \geq 2$ candidate answers). Then the probability of consistency is approximately $\sum_i p_i^2 \approx n \cdot (\frac{1}{n})^2 = \frac{1}{n}$, while the probability of inconsistency is $1 - \frac{1}{n} = \frac{(n-1)}{n}$, which is greater than the consistency probability. Therefore, in a flat distribution, the probability of obtaining inconsistent answers in two samplings is higher than that of obtaining consistent answers. In summary, whether two samplings yield consistent answers reflects the sharpness of the answer distribution: consistent answers are more likely to be the most frequent answer, whereas inconsistent answers are more likely to correspond to flat distributions.

C Prompt Templates

Prompt Templates

Mathematical Reasoning Prompt:

Please reason step by step, and put your final answer within boxed{ }.

Rewriting Prompt for Mathematical Problems:

Please remove unnecessary descriptions from the following question, simplify its length while keeping the original meaning unchanged, and retain important numbers and symbols. Only provide the revised question without answers or calculations.

Code Generation Prompt:

Please provide a self-contained Python script that solves the following problem in a markdown code block:

{input}

Below is a Python script with a self-contained function that solves the problem and passes corresponding tests:

Rewriting Prompt for Coding Problems:

Rewrite the following programming problem in your own words. Keep the meaning, constraints, and examples exactly the same, but rephrase the description naturally. Do NOT provide any solution or any code besides the rewritten problem text.

{input}

Here is the rewritten problem:

D Case Study

Case	Original	Rewritten
hidden info error	<p>Question: Determine the smallest positive real number k with the following property. Let $ABCD$ be a convex quadrilateral, and let points A_1, B_1, C_1 and D_1 lie on sides AB, BC, CD and DA, respectively. Consider the areas of triangles $AA_1D_1, BB_1A_1, CC_1B_1$, and DD_1C_1; let S be the sum of the two smallest ones, and let S_1 be the area of quadrilateral $A_1B_1C_1D_1$. Then we always have $kS_1 \geq S$.</p> <p>Answer: 1</p> <p>Output: ... By choosing points A_1, B_1, C_1 and D_1 arbitrarily close to the vertices of the quadrilateral, the area S_1 of $A_1B_1C_1D_1$ can be made arbitrarily small ... Final Answer $\boxed{\frac{1}{2}}$</p>	<p>Question: Let $ABCD$ be a convex quadrilateral. Choose points $A_1 \in AB, B_1 \in BC, C_1 \in CD, D_1 \in DA$ strictly inside each side so that $A_1B_1C_1D_1$ is convex and lies entirely inside $ABCD$. Define $T_1 = [AA_1D_1], T_2 = [BB_1A_1], T_3 = [CC_1B_1], T_4 = [DD_1C_1]$ and let S be the sum of the two smallest of these areas. Let $S_1 = [A_1B_1C_1D_1]$. Find the smallest $k > 0$ such that $kS_1 \geq S$ holds for all configurations.</p> <p>Answer: 1</p> <p>Output: Let $ABCD$ be convex, and points A_1, B_1, C_1, D_1 chosen cyclically. Then $A_1B_1C_1D_1$ is always convex inside $ABCD$... Final Answer $\boxed{1}$</p>
answer format error	<p>Question: Given sets $M = \{x x+2 \geq 0\}, N = \{x x-1 < 0\}$, find $M \cap N$.</p> <p>Answer: $\{x -2 \leq x < 1\}$</p> <p>Output: To find the ... this is: $[-2, 1)$ Final Answer $\boxed{[-2, 1)}$</p>	<p>Question: $M = \{x x+2 \geq 0\}, N = \{x x-1 < 0\}$, find $M \cap N$.</p> <p>Answer: $\{x -2 \leq x < 1\}$</p> <p>Output: To find the ... but not including, Final Answer $\boxed{[-2, 1)}$</p>
context loss error	<p>Question: Let A be the sum of the digits of the number you will receive from position 7, and let B be the sum of the digits of the number you will receive from position 9. Let (x, y) be a point randomly selected from the interior of the triangle whose consecutive vertices are $(1, 1), (B, 7)$ and $(17, 1)$. Compute the probability that $x > A - 1$.</p> <p>Answer: $\frac{79}{128}$</p> <p>Output: Let the triangle ... For a fixed y, let $x_{min} = 1$... Since x ranges from 1 to 17, we compute $P(x > A - 1) = \frac{17-(A-1)}{16}$...</p>	<p>Question: Compute the probability that $x > A - 1$ where A is the sum of the digits of the number from position 7 and B is the sum of the digits of the number from position 9, with (x, y) selected randomly from the interior of the triangle with vertices $(1, 1), (B, 7)$ and $(17, 1)$.</p> <p>Answer: $\frac{79}{128}$</p> <p>Output: Let (x, y) be uniformly ... The area of the triangle is $S = \frac{1}{2}(17-1)(7-1) = 48$... we compute $P(x > A - 1) = \frac{17-(A-1)}{16}$...</p>
calculation error	<p>Question: Determine the minimum potential in V (to 2 decimal places) that must be applied to an α-particle so that on interaction with a hydrogen atom, a ground state electron will be excited to $n = 6$.</p> <p>Answer: 6.62</p> <p>Output: ... Sixth energy level ($n = 6$): $E_6 = -\frac{13.6}{36} \approx -0.3778\text{eV}$. Energy difference required to excite the electron $\Delta E = E_6 - E_1 = -0.3778 - (-13.6) = 13.2222\text{eV}$... Solving for V: $V = 13.22/2 = 6.61\text{V}$. Final Answer $\boxed{6.61}$</p>	<p>Question: Determine the minimum electric potential (rounded to 2 decimal places) that must be applied to an α-particle (charge $+2e$) so that, when it interacts with a hydrogen atom, it transfers enough energy to excite the hydrogen atom's electron from the ground state ($n = 1$) to the sixth energy level ($n = 6$).</p> <p>Answer: 6.62</p> <p>Output: ... Determine the Energy Required for Excitation. The energy levels of a hydrogen atom are given by the formula: $E_n = -\frac{13.6}{n^2}$ eV For the ground state $n = 1$: $E_1 = -13.6$ eV ... Solving for V: $V = \frac{13.2222}{2} = 6.6111\text{V}$. ... Final Answer $\boxed{6.61}$</p>
over-compression error	<p>Question: 50 people sign up for a football club, 60 people sign up for a table tennis club, and 70 people sign up for either the football or the table tennis club. There are no other clubs and everyone signs up for at least one of the clubs. If someone signs up for the football club, what is the probability that the same person signs up for the table tennis club?</p> <p>Answer: 0.8</p> <p>Output: To solve this problem, we are given the following information: 50 people signed up for the football club. 60 people signed up for the table tennis club. 70 people signed up for either the football or the table tennis club. ... Final Answer $\boxed{0.8}$</p>	<p>Question: If someone signs up for the football club, what is the probability they also signed up for the table tennis club?</p> <p>Answer: 0.8</p> <p>Output: To determine the probability that someone who signed up for the football club also signed up for the table tennis club, we are dealing with a conditional probability problem. ... However, no specific data or probabilities are provided in the question. Without knowing ... Hypothetical Scenario If we assume a common textbook-type scenario, for example: Total of 100 people, 60 signed up for football, 20 signed up for both football and table tennis ... the probability would be: $\boxed{\frac{1}{3}}$</p>

Case 1 (Original)

Original Question:

Suppose the polynomial

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

has integer coefficients and distinct integer roots. Given $a_n = 2$ and $a_0 = 66$, find the least possible value of $|a_{n-1}|$.

Model Output:

To solve the problem, we are given a polynomial of the form:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$$

with integer coefficients, distinct integer roots, and the following constraints: $a_n = 2$ $a_0 = 66$ We are to find the least possible value of $|a_{n-1}|$ Step 1: Use Vieta's Formulas. The polynomial can be written in factored form as:

$$f(x) = 2(x - r_1)(x - r_2) \cdots (x - r_n)$$

where r_1, r_2, \dots, r_n are distinct integers (the roots), and $a_n = 2$. The constant term a_0 is given by:

$$a_0 = 2 \cdot (-1)^n \cdot r_1 r_2 \cdots r_n$$

We are told $a_0 = 66$, so: $2 \cdot (-1)^n \cdot (r_1 \cdots r_n) = 66 \Rightarrow (-1)^n \cdot (r_1 \cdots r_n) = 33$ Thus, the product of the roots (with sign) is:

$$r_1 r_2 \cdots r_n = \pm 33$$

Step 2: Express a_{n-1} in Terms of Roots. From Vieta's formulas, the coefficient a_{n-1} is given by:

$$a_{n-1} = -2 \cdot (\text{sum of roots})$$

So:

$$|a_{n-1}| = 2 \cdot |\text{sum of roots}|$$

Our goal is to minimize this value, which reduces to finding a set of distinct integers whose product is ± 33 , and whose sum has the smallest absolute value. Step 3: Try Different Degrees n . Let's consider different possible values of n , and for each, find the smallest possible absolute value of the sum of distinct integer roots with product ± 33 . Case $n = 4$: We want four distinct integers whose product is 33 (since $n = 4$ is even, the product is positive). Let's try: $r_1 = 1, r_2 = -1, r_3 = 3, r_4 = -11$ Check product:

$$1 \cdot (-1) \cdot 3 \cdot (-11) = 33$$

Sum:

$$1 + (-1) + 3 + (-11) = -8 \Rightarrow |a_{n-1}| = 2 \cdot 8 = 16$$

This is a candidate for the minimum. Try other combinations: $r_1 = -1, r_2 = 1, r_3 = -3, r_4 = 11$; product = 33, sum = $-1 + 1 - 3 + 11 = 8$, same result. Thus, minimum $|a_{n-1}| = 16$ for $n = 4$. Check for smaller $|a_{n-1}|$: Try other values of n :

- $n = 2$: roots (3, 11), sum = 14 $\Rightarrow |a_{n-1}| = 28$
- $n = 3$: roots (-3, 1, 11), sum = 9 $\Rightarrow |a_{n-1}| = 18$
- $n = 1$: root -33 $\Rightarrow |a_{n-1}| = 66$
- $n = 5$: no valid set of 5 distinct integers with product -33
- $n = 6$: no valid set of 6 distinct integers with product 33

So, the minimal value occurs at $n = 4$ with $|a_{n-1}| = 16$. Final Answer 16

Case 1 (Rewritten)

Rewritten Question:

Suppose the polynomial

$$f(x) = 2x^n + a_{n-1}x^{n-1} + \cdots + a_0$$

has distinct integer roots and $a_0 = 66$. What is the least possible value of $|a_{n-1}|$?

Model Output:

To determine the least possible value of $|a_{n-1}|$ for a polynomial with integer coefficients, distinct integer roots, leading coefficient $a_n = 2$, and constant term $a_0 = 66$, we proceed as follows: Step 1: Polynomial Structure. The polynomial is of the form:

$$f(x) = 2(x - r_1)(x - r_2) \cdots (x - r_n)$$

where r_1, r_2, \dots, r_n are distinct integers. The leading coefficient is 2, and the constant term a_0 is given as 66. Step 2: Constant Term and Product of Roots. Using the expansion of the polynomial, the constant term is:

$$a_0 = 2 \cdot (-1)^n \cdot r_1 r_2 \cdots r_n$$

Given $a_0 = 66$, we solve:

$$2 \cdot (-1)^n \cdot (r_1 r_2 \cdots r_n) = 66 \Rightarrow r_1 r_2 \cdots r_n = \frac{66}{2 \cdot (-1)^n} = \frac{33}{(-1)^n}$$

Thus, the product of the roots is ± 33 , and the roots must be distinct integers. Step 3: Coefficient a_{n-1} and Its Absolute Value. Using Vieta's formula, the coefficient a_{n-1} is related to the sum of the roots:

$$a_{n-1} = -2 \cdot (r_1 + r_2 + \cdots + r_n)$$

So, the absolute value is:

$$|a_{n-1}| = 2 \cdot |r_1 + r_2 + \cdots + r_n|$$

To minimize $|a_{n-1}|$, we must minimize the absolute value of the sum of the roots, subject to the constraint that the product of the roots is ± 33 , and the roots are distinct integers. Step 4: Searching for Optimal Root Combinations. We need to find distinct integers whose product is ± 33 and whose sum has the smallest absolute value. The integer divisors of 33 are:

$$\pm 1, \pm 3, \pm 11, \pm 33$$

We test combinations of three distinct integers (since more than three would require duplicate factors or make the sum too large) that multiply to ± 33 . Among the valid combinations, the one that minimizes $|r_1 + r_2 + r_3|$ is: $r_1 = 1, r_2 = 3, r_3 = -11$ Product: $1 \cdot 3 \cdot (-11) = -33$ Sum: $1 + 3 - 11 = -7$ Absolute value of sum: 7 This yields:

$$|a_{n-1}| = 2 \cdot |-7| = 14$$

Other combinations (e.g., $-1, -3, 11$) also yield the same minimal sum of 7 in absolute value. Step 5: Verifying Optimality. Trying all other combinations of 2 or 4 distinct integers with product ± 33 results in larger absolute sums. For example:

$$(3, -11) : \quad \text{sum} = -8, \quad |a_{n-1}| = 16,$$

$$(1, -1, 3, -11) : \quad \text{sum} = -8, \quad |a_{n-1}| = 16,$$

$$(1, -1, -3, 11) : \quad \text{sum} = 8, \quad |a_{n-1}| = 16.$$

All of these are greater than 14. Hence, the minimum possible value of $|a_{n-1}|$ is achieved with the root set $(1, 3, -11)$ or $(-1, -3, 11)$, both giving a minimal sum of 7 in absolute value. Final Answer $\boxed{14}$

Case 2 (Original)

Original Question:

An integer $n \geq 3$ is given. We call an n -tuple of real numbers (x_1, x_2, \dots, x_n) *Shiny* if for each permutation y_1, y_2, \dots, y_n of these numbers we have $\sum_{i=1}^{n-1} y_i y_{i+1} = y_1 y_2 + y_2 y_3 + \dots + y_{n-1} y_n \geq -1$. Find the largest constant $K = K(n)$ such that $\sum_{1 \leq i < j \leq n} x_i x_j \geq K$ holds for every Shiny n -tuple (x_1, x_2, \dots, x_n) .

Model Output:

To solve the problem, we are given an integer $n \geq 3$, and an n -tuple (x_1, x_2, \dots, x_n) is called Shiny if for every permutation (y_1, y_2, \dots, y_n) of the tuple, the sum

$$\sum_{i=1}^{n-1} y_i y_{i+1} \geq -1$$

holds. Our goal is to find the largest constant $K(n)$ such that for every Shiny n -tuple, the sum of all pairwise products

$$\sum_{1 \leq i < j \leq n} x_i x_j$$

is at least $K(n)$. Step 1: Understanding the Objective. We are asked to find the minimum value of the expression:

$$\sum_{1 \leq i < j \leq n} x_i x_j$$

over all Shiny n -tuples. Then, the largest constant $K(n)$ such that this sum is always at least $K(n)$ is precisely that minimum value. Step 2: Key Observations. We can express the sum of all pairwise products as:

$$\sum_{1 \leq i < j \leq n} x_i x_j = \frac{1}{2} \left(\left(\sum_{i=1}^n x_i \right)^2 - \sum_{i=1}^n x_i^2 \right)$$

However, this expression is not immediately helpful for our purpose. Instead, we focus on constructing examples of Shiny n -tuples and computing the value of the pairwise sum. Step 3: Constructing a Shiny n -tuple with Minimal Pairwise Sum. Let us consider the n -tuple:

$$(x_1, x_2, \dots, x_n) = \left(a, -\frac{1}{a}, 0, 0, \dots, 0 \right)$$

Then, the pairwise sum becomes:

$$\sum_{1 \leq i < j \leq n} x_i x_j = a \cdot \left(-\frac{1}{a} \right) = -1$$

Now, we must verify that this tuple is Shiny, i.e., that for every permutation of the tuple, the sum of adjacent products is at least -1 . In any permutation, the only non-zero product will be between a and $-\frac{1}{a}$, which is -1 . All other products will be between a and 0 , or between $-\frac{1}{a}$ and 0 , or between 0 and 0 , all of which are 0 . Therefore, the sum of adjacent products in any permutation is either -1 (when the two non-zero elements are adjacent) or 0 (otherwise). Thus, the minimum sum over all permutations is -1 , which satisfies the Shiny condition. Step 4: Can We Do Better?

Suppose there exists a Shiny n -tuple with pairwise sum less than -1 . Let's assume that such a tuple exists. Then, the minimum sum over all permutations of adjacent products would be less than -1 , which would violate the Shiny condition. Therefore, such a tuple cannot exist. This implies that the minimal value of the pairwise sum over all Shiny n -tuples is exactly -1 . Step 5: Conclusion. We have constructed a Shiny n -tuple with pairwise sum -1 , and we have shown that it is not possible to construct a Shiny n -tuple with a pairwise sum less than -1 . Therefore, the largest constant $K(n)$ such that the pairwise sum is always at least $K(n)$ for all Shiny n -tuples is:

$$\boxed{-1}$$

Case 2 (Rewritten)

Rewritten Question:

An integer $n \geq 3$ is given. We call an n -tuple of real numbers (x_1, x_2, \dots, x_n) *Shiny* if for every permutation (y_1, y_2, \dots, y_n) of these numbers, the following inequality holds: $y_1y_2 + y_2y_3 + \dots + y_{n-1}y_n \geq -1$. For every Shiny n -tuple, consider the sum of all pairwise products $S = \sum_{1 \leq i < j \leq n} x_i x_j$. Find the largest constant $K(n)$ such that $S \geq K(n)$ holds for all Shiny n -tuples.

Model Output:

To determine the largest constant $K(n)$ such that for every Shiny n -tuple, the sum of all products of pairs $x_i x_j$ for $i < j$ is at least $K(n)$, we begin by understanding the key constraints imposed by the Shiny condition. Understanding the Shiny Condition. A tuple (x_1, x_2, \dots, x_n) is Shiny if, for any permutation of the tuple, the sum of adjacent products is at least -1 . That is:

$$\min_{\text{permutations } \pi} \left(\sum_{i=1}^{n-1} \pi_i \pi_{i+1} \right) \geq -1$$

This condition ensures that no matter how we rearrange the elements, the sum of adjacent products never drops below -1 . Goal: Find the Largest Lower Bound $K(n)$. We are to find the largest constant $K(n)$ such that for every Shiny n -tuple, the sum of all pairwise products is at least $K(n)$, that is:

$$\sum_{1 \leq i < j \leq n} x_i x_j \geq K(n)$$

We can relate this sum to the square of the sum of the elements:

$$\left(\sum_{i=1}^n x_i \right)^2 = \sum_{i=1}^n x_i^2 + 2 \sum_{1 \leq i < j \leq n} x_i x_j$$

Rearranging gives:

$$\sum_{1 \leq i < j \leq n} x_i x_j = \frac{1}{2} \left(\left(\sum x_i \right)^2 - \sum x_i^2 \right)$$

This expression will be key in analyzing the behavior of the sum of pairwise products. Constructing a Candidate Shiny Tuple. To find the minimum value of $\sum x_i x_j$ across all Shiny n -tuples, we consider a specific configuration: Let $n - 1$ elements be ε (a small positive number) Let the last element be $-c$, where $c > 0$. This gives us a tuple of the form $(\varepsilon, \varepsilon, \dots, \varepsilon, -c)$, with $n - 1$ copies of ε and one $-c$. We analyze the sum of pairwise products:

$$\sum_{1 \leq i < j \leq n} x_i x_j = \frac{1}{2} \left(((n-1)\varepsilon - c)^2 - ((n-1)\varepsilon^2 + c^2) \right)$$

After simplification, this becomes: $\sum x_i x_j = \frac{n-1}{2} (\varepsilon^2 - 1)$ To ensure the tuple is Shiny, we must also satisfy the condition that the minimum sum of adjacent products over all permutations is at least -1 . By analyzing the worst-case permutation (when $-c$ is adjacent to two ε 's), we find that this condition is satisfied as $\varepsilon \rightarrow 0$, and we can set c such that the sum of adjacent products is exactly -1 . Asymptotic Behavior and Conclusion. As $\varepsilon \rightarrow 0$, the sum of pairwise products approaches: $\sum x_i x_j \rightarrow \frac{n-1}{2} \cdot (-1) = -\frac{n-1}{2}$ This is the smallest possible value of the sum of pairwise products that still satisfies the Shiny condition. Since we can construct such a tuple that approaches this value, and no tuple can have a smaller sum without violating the Shiny condition,

we conclude that: $\boxed{-\frac{n-1}{2}}$ is the largest constant $K(n)$ such that for every Shiny n -tuple, the

sum of all products of pairs is at least $K(n)$. Final Answer $\boxed{-\frac{n-1}{2}}$