WHEN TO VOTE, WHEN TO REWRITE: DISAGREEMENT-GUIDED STRATEGY ROUTING FOR TEST-TIME SCALING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Reasoning Models (LRMs) achieve strong performance on mathematical reasoning tasks but remain unreliable on challenging instances. Existing test-time scaling methods, such as repeated sampling, self-correction, and tree search, improve performance at the cost of increased computation, yet often exhibit diminishing returns on hard problems. We observe that output disagreement is strongly correlated with instance difficulty and prediction correctness, providing a useful signal for guiding instance-level strategy selection at test time. Based on this insight, we propose a training-free framework that formulates test-time scaling as an instance-level routing problem, rather than allocating more computation within a single strategy, dynamically selecting among different scaling strategies based on output disagreement. The framework applies lightweight resolution for consistent cases, majority voting for moderate disagreement, and rewriting-based reformulation for highly ambiguous instances. Experiments on seven mathematical benchmarks and three models show that our method improves accuracy by 3%–7% while reducing sampling cost compared to existing approaches.

1 INTRODUCTION

Large Reasoning Models (LRMs) have achieved high performance on mathematical reasoning benchmarks such as Olympiad problems (Yang et al., 2024b; Shao et al., 2024; DeepSeek-AI et al., 2024), yet struggle with challenging instances. A common direction is test-time scaling, which allocates additional computation at inference time to improve output quality. Representative approaches 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; Aggarwal et al., 2023), and tree search methods (Yao et al., 2023; Zhang et al., 2024a; Bi et al., 2024; Gao et al., 2024). While effective, these approaches often suffer from diminishing returns as computation increases (Xue et al., 2023), suggesting that computation is not effectively allocated across instances of varying difficulty. Most existing methods primarily focus on adaptively allocating computation within a single strategy, with limited attention to selecting among different strategies.

Therefore, a key challenge is not only how to allocate computation, but also how to select appropriate inference strategies for different instances. Among various test-time scaling strategies, rewriting has been shown to be an effective approach for enhancing model performance without retraining, based on the observation that model outputs are highly sensitive to variations in problem formulation. Prior work has explored this idea from different perspectives. Zhou et al. (2024) improves robustness via rewriting combined with a voting mechanism. Zhang et al. (2025b) applies rewriting as a preprocessing step before code generation. More recently, Lai et al. (2025) further enforces consistency across prompt formulation, expression, and language style to stabilize model behavior.

However, while rewriting is effective, it is not universally beneficial. It may alter the semantic meaning of the original instance or introduce incorrect assumptions during reformulation. As shown in Figure 1, rewriting degrades performance on easy instances, while it consistently improves accuracy on hard instances. In contrast, majority voting achieves strong performance on easy instances, but degrades on hard instances. We further observe a strong relationship between instance difficulty, output disagreement, and prediction correctness: hard instances tend to exhibit higher output disagreement,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

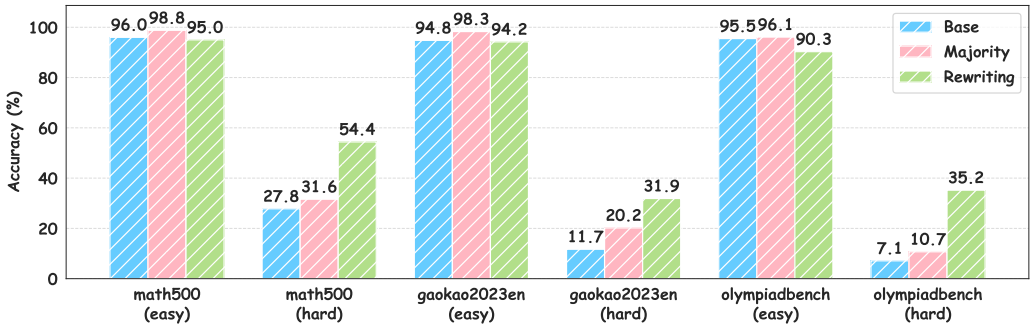


Figure 1: The effect of rewriting and majority voting (6 samplings) for Qwen3-8B across instances of different difficulty levels. Easy instances are defined as those correctly answered in a single sampling, while hard instances are those incorrectly answered in a single sampling.

whereas easy instances usually produce more consistent results. Moreover, instances with severe disagreement under majority voting, characterized by a flat distribution over multiple answers, are highly correlated with incorrect predictions. This observation indicates that output disagreement reflects the model's uncertainty under stochastic sampling, providing a natural signal for guiding strategy selection. In particular, when model outputs are highly consistent, instances are typically easier and majority voting is sufficient; whereas under high disagreement, instances are more likely to be challenging, making rewriting more effective. This reveals a clear performance trade-off between majority voting and rewriting-based strategies and motivates selecting appropriate inference strategies based on model disagreement.

Based on this finding, we design a training-free framework without fine-tuning or external models. Rather than introducing new reasoning modules, our framework formulates test-time scaling as an instance-level decision problem, determining which strategies should be applied under different uncertainty conditions. The framework operates by routing instances to different strategies based on the degree of model disagreement, which we interpret as a proxy for uncertainty under stochastic sampling. In the first stage **(Disagreement Filter)**, we identify no-disagreement instances via a two-sampling consistency check. These instances are directly resolved without additional sampling or rewriting, avoiding unnecessary computation. In the second stage **(Vote Resolve)**, instances with minor disagreement are further sampled and resolved via majority voting over multiple generations, leveraging increased sampling to improve reliability. In the final stage **(Rewrite & Rethink)**, instances with persistent disagreement are rewritten and re-sampled to escape inconsistent reasoning. This framework selects appropriate strategies for different uncertainty levels, improving efficiency by avoiding redundant computation on easy cases while focusing resources on ambiguous ones.

We conduct experiments on seven mathematical benchmarks and three LRMs. The results demonstrate that our method improves mathematical reasoning accuracy by 3%–7% absolute, with particularly strong gains on challenging benchmarks where the base models perform poorly. Moreover, our method achieves these improvements while using fewer samplings on the evaluated benchmarks. We further validate the generalization ability of our method on code generation tasks. Finally, we provide an analysis of each component, highlighting the effectiveness of the proposed design and the role of rewriting in improving robustness.

2 Related Work

2.1 Test-Time Scaling For Math Reasoning

Test-time scaling increases inference-time computation to improve answer quality and has been shown effective for mathematical reasoning (Ji et al., 2025; Zhang et al., 2025a). Representative approaches include repeated sampling (Wang et al., 2022; Li et al., 2024a; Lin et al., 2024; Gui et al., 2024; Zhang et al., 2024b), self-correction (Gao et al., 2025; 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) among

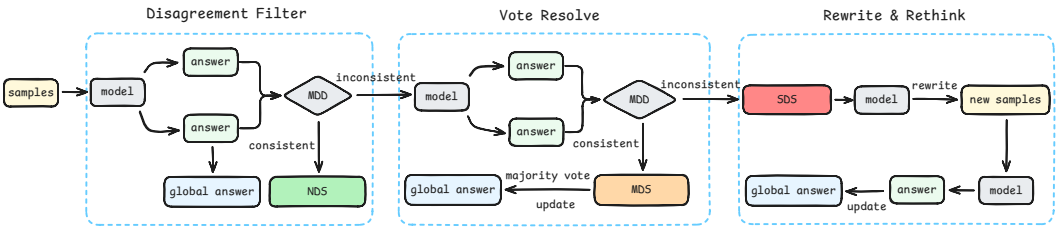


Figure 2: This figure illustrates the overall workflow of our method, where MDD, NDS, MDS, and SDS stand for Minimal Disagreement Detector, No Disagreement Samples, Minor Disagreement Samples, and Severe Disagreement Samples, respectively. The four model blocks in the figure all refer to the same model.

others (Bi et al., 2024; Gao et al., 2024). These methods improve reliability via multiple outputs or iterative refinement. Recent work (Agarwal et al., 2025; Son et al., 2025; Zhao et al., 2026; Xiang et al., 2026) also studies adaptive test-time compute allocation and compute-accuracy trade-offs, which are complementary to our focus on routing among inference strategies.

2.2 Rewriting

Rewriting has been widely studied in both information systems and language model reasoning. In database systems, query rewriting is used to improve retrieval efficiency (Liu & Mozafari, 2024). Recent work extends this idea to LLMs. RewriteLM (Shu et al., 2024) combines instruction tuning and reinforcement learning to optimize rewrite quality, while LLM-R2 (Li et al., 2024c) learns to distinguish semantically equivalent but more effective rewrites via contrastive training. Other studies explore prompt optimization and preference-based rewriting (Kong et al., 2024; Chen et al., 2025).

In mathematical reasoning, Zhou et al. (2024) demonstrate that surface form significantly affects model performance and propose SCoP, which applies paraphrasing with self-consistency. Zhang et al. (2025b) adopt a rewrite-then-execute strategy to improve robustness, while Lai et al. (2025) enforce multi-dimensional consistency across prompts and expressions. These works highlight rewriting as an effective way to improve reasoning stability and generalization. However, these methods primarily focus on improving single-pass generation quality, and do not explicitly address uncertainty-aware computation allocation during inference.

3 Methodology

Our method is illustrated in Figure 2, and consists of three stages: **Disagreement Filter**, **Vote Resolve**, and **Rewrite & Rethink**. We employ two minimal disagreement detectors to partition samples into three categories based on the degree of divergence, namely No Disagreement Samples, Minor Disagreement Samples, and Severe Disagreement Samples. Majority voting is applied to Minor Disagreement Samples, while rewriting is used for Severe Disagreement Samples, and the corresponding results are updated in the global answer.

Algorithm 1 provides a step-by-step procedure of how each sample is processed in our framework. Section 3.1 defines the minimal disagreement detector and the sample categorization. Section 3.2 describes the first two stages, including how samples are filtered and partitioned, as well as how iteration steps are determined. Section 3.3 details how final answers for Severe Disagreement Samples are obtained via rewriting.

3.1 Sample Categorization via the Minimal Disagreement Detector

We sample each instance n times and obtain n model-generated answers, denoted as $\{a_1, a_2, \ldots, a_n\}$. When the distribution of sampled answers is highly concentrated, i.e., one answer A dominates, majority voting can reliably select A as the final prediction, indicating strong model consensus. In

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^*

$a_1, a_2 \leftarrow M(S, P_{\text{reason}})$
if $a_1 = a_2$ **then**
  return $a^* \leftarrow a_1$
else if $a_1 \neq a_2$ **then**
  $a_3, a_4 \leftarrow M(S, P_{\text{reason}})$
  if $a_3 = a_4$ **then**
    return $a^* \leftarrow Maj(a_1, a_2, a_3, a_4)$
  else if $a_3 \neq a_4$ **then**
    $S_{\text{rewrite}} \leftarrow M(S, P_{\text{rewrite}})$, $a_{\text{rewrite}} \leftarrow M(S_{\text{rewrite}}, P_{\text{reason}})$
    return $a^* \leftarrow a_{\text{rewrite}}$
  end if
end if

contrast, a flat distribution with no dominant answer suggests significant disagreement among model outputs, implying unreliable predictions and reduced effectiveness of majority voting.

To exploit this observation, we propose a minimal disagreement detector (MDD), which performs two samplings to obtain a_1 and a_2. If $a_1 \neq a_2$, the sample is identified as exhibiting disagreement; otherwise, it is considered consistent. Answer equivalence is determined using a normalized matching procedure that accounts for numerical, structural, and symbolic equivalence (see Appendix D). As shown in Figure 2, consistent samples are directly accepted without rewriting. This design is motivated by the observation that rewriting is beneficial for hard instances, while providing limited gains for easy ones (Figure 1). Therefore, MDD is designed to maintain high recall of correct predictions while providing a lightweight decision rule for choosing between majority voting and rewriting based on disagreement degree. Table 7 further shows that increasing the number of consistency checks reduces recall of correct instances, motivating the use of a two-sampling design for efficiency.

We perform multiple independent evaluations of each sample using MDD. Let d_i denote the number of detected disagreements for the i-th sample S_i. Each disagreement is computed independently within one stage from a single pairwise comparison, without comparing answers across different stages. Based on d_i, samples are categorized into three groups:

No Disagreement Samples (NDS): $d_i = 0$. No disagreement is observed, indicating stable model behavior; these samples are directly accepted without additional computation.

Minor Disagreement Samples (MDS): $d_i = 1$. A single disagreement is observed, indicating moderate uncertainty; additional sampling is required to improve reliability.

Severe Disagreement Samples (SDS): $d_i \geq 2$. Multiple disagreements indicate highly unstable predictions. The answer distribution is flat, making it difficult to obtain a reliable result by simply increasing the number of samplings. In such cases, sampling alone is insufficient, and rewriting is required to reformulate the input before re-reasoning.

To better illustrate the relationship between disagreement level and correctness, we further quantify the relationship between disagreement level and correctness via $P(\text{correct} \mid \text{NDS/MDS/SDS})$ in Table 8, showing a strong negative correlation between disagreement and accuracy. This suggests that disagreement is a reliable signal for distinguishing samples with different levels of difficulty.

3.2 Iterative Filtering Strategy

To efficiently allocate computational resources, we adopt an iterative filtering strategy that categorizes samples into NDS, MDS, and SDS, and applies rewriting only to SDS, where majority voting is insufficient. To justify this strategy allocation, we compare baseline answering, majority voting, and rewriting within each partition in Table 9, showing that different disagreement levels favor different strategies. We further analyze rewriting behavior in Figure 1, where rewriting is most effective on hard samples. Since rewriting is applied to SDS, these samples should retain a high recall of incorrect

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, dynamic voting, SCoP, BoN, and our method. Standard deviations in the Avg column are computed over 3 runs with different random seeds.

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$\pm$0.71
Qwen3-8B (Majority)	87.6	79.7	50.7	95.8	85.0	33.3	50.0	68.9$\pm$0.26
Qwen3-8B (DV)	87.6	79.7	49.5	95.7	85.0	30.0	43.3	67.3$\pm$0.33
Qwen3-8B (SCoP)	88.2	79.5	52.6	95.4	82.5	33.3	40.0	67.4$\pm$0.34
Qwen3-8B (BoN)	85.8	75.6	49.0	95.1	65.0	33.3	40.0	63.4$\pm$0.57
Qwen3-8B (Ours)	**92.8**	**82.9**	**61.9**	**95.8**	**87.5**	**43.3**	**66.7**	**75.8$\pm$0.43**
Qwen3-4B	85.4	75.6	49.0	94.2	65.0	23.3	30.0	60.4$\pm$0.62
Qwen3-4B (Majority)	88.6	79.5	53.2	**95.5**	82.5	26.7	50.0	68.0$\pm$0.27
Qwen3-4B (DV)	88.6	78.4	51.3	95.1	80.0	30.0	46.7	67.2$\pm$0.31
Qwen3-4B (SCoP)	87.8	80.3	55.6	93.5	82.5	26.7	40.0	66.6$\pm$0.36
Qwen3-4B (BoN)	87.4	76.4	50.1	94.1	72.5	26.7	50.0	65.3$\pm$0.59
Qwen3-4B (Ours)	**90.6**	**81.8**	**57.9**	94.5	**87.5**	**43.3**	**56.7**	**73.2$\pm$0.55**
DS-Llama-8B	80.2	74.0	46.4	85.0	67.5	30.0	40.0	60.4$\pm$0.80
DS-Llama-8B (Majority)	84.6	79.8	51.6	87.5	80.0	**33.3**	46.7	66.2$\pm$0.42
DS-Llama-8B (DV)	85.8	76.9	51.0	87.5	77.5	33.3	46.7	65.5$\pm$0.43
DS-Llama-8B (SCoP)	88.2	78.2	51.9	89.0	85.0	**33.3**	40.0	66.5$\pm$0.53
DS-Llama-8B (BoN)	79.0	78.8	55.3	88.8	75.0	30.0	43.3	64.3$\pm$0.76
DS-Llama-8B (Ours)	**88.4**	**81.3**	**54.7**	**91.2**	**87.5**	**33.0**	**50.0**	**69.4$\pm$0.68**

cases. As shown in Figure 6, increasing iterations reduces error recall while increasing cost, so we use two iterations 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 as a provisional answer in the global answer, which maintains the current best answer for each sample and is updated via stage-wise overwriting. 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 via rewriting followed by re-reasoning.

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 additional reasoning paths, the model is prompted to reformulate the problem and reattempt solving it from an alternative formulation of the problem.

We apply the rewriting prompt template shown in Appendix C to each SDS for a single rewrite, generating a rewritten set $S_{rewrite}$. The rewriting step aims to reformulate the surface expression of the problem while preserving its underlying semantics and final answer. For each sample in $S_{rewrite}$, we perform a single sampling to obtain an answer $a_{rewrite}$, which is recorded in the global answer. By altering the surface form of the problem, this process encourages the model to reconsider the task from a different perspective, helping it escape previously incorrect reasoning trajectories. With minimal additional computation, it can significantly improve accuracy.

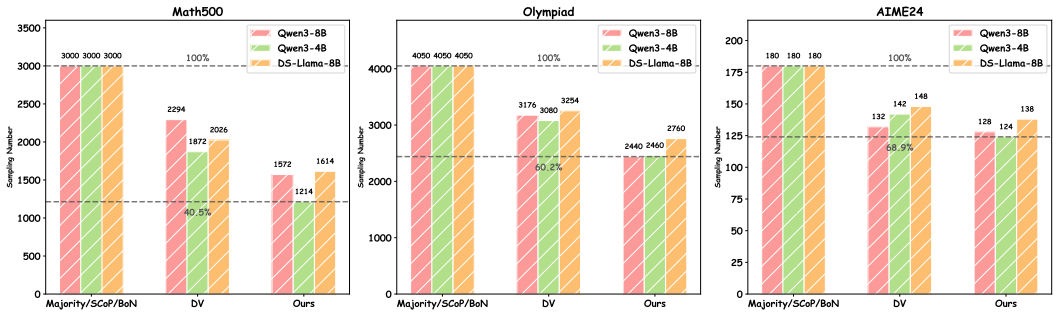


Figure 3: The number of samplings used by different methods. The dashed line indicates the percentage of the maximum sampling budget.

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)). We use accuracy of the final answers as the evaluation metric.

To better understand the performance of our method, we compare with four representative methods: majority voting, dynamic voting (Xue et al., 2023), Best-of-N with an outcome reward model, and SCoP (Zhou et al., 2024). The first three are abbreviated as Majority, DV, and BoN. For fair comparison, we define a sampling as any base model generation, including both reasoning and rewriting, and constrain all methods to six samplings per instance. Our method uses at most six samplings per instance (2 for stage-1, 2 for stage-2, 1 for rewriting, and 1 for re-reasoning). The Majority method samples six times per instance and selects the final answer through simple voting; the DV method allows up to six samplings per instance and selects the final answer through a confidence-based voting mechanism with a threshold of 0.7, selected based on a threshold sensitivity analysis (Table 10); BoN 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, applies one round of problem rewriting to all instances, followed by five rounds of sampling, and selects the final answer through simple voting. We use temperature 0.6, top-p 0.95, and top-k 20, and run three seeds (0, 42, 777). Experiments are conducted under 8K and 16K contexts (Table 1 and 5) on the Qwen2.5-Math[1] framework with 8 NVIDIA A100 GPUs. We further evaluate our method on a larger-scale model (Qwen3-32B) on a subset of benchmarks to verify scalability, with results shown in Table 11.

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 Table 6, for Qwen3-4B and Qwen3-8B on the GSM8K benchmark, 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.

[1] https://github.com/QwenLM/Qwen2.5-Math

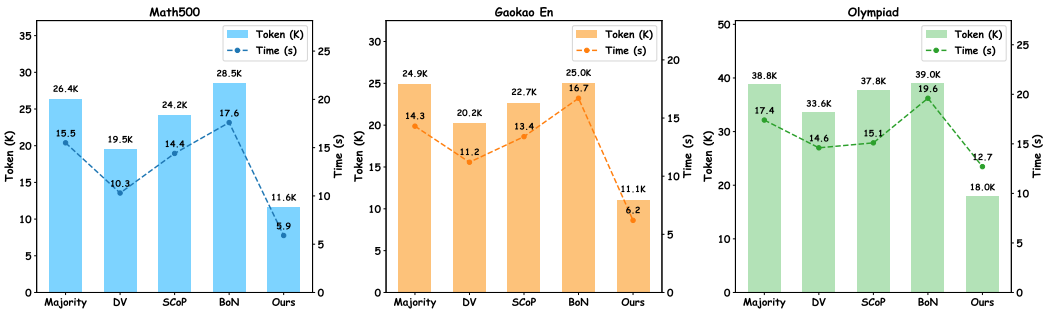


Figure 4: Average tokens and wall-clock time per sample for Qwen3-8B on Math500, Gaokao En, and Olympiad. Reported costs cover the entire pipeline: for SCoP and our method, rewriting costs are included; for BoN, reward model costs are included. Each rewriting step is counted as one sampling.

Table 2: This table presents the evaluation results of our method on code generation tasks.

Model / Method	HumanEval	HumanEval+	MBPP	MBPP+	Avg
Qwen3-8B	84.8	80.5	70.5	60.8	74.2
Qwen3-8B (Majority)	85.5	79.9	71.0	60.5	74.2
Qwen3-8B (DV)	85.3	79.5	71.0	60.3	74.0
Qwen3-8B (SCoP)	83.7	79.0	69.9	60.1	73.2
Qwen3-8B (BoN)	85.4	80.7	72.0	61.4	74.9
Qwen3-8B (Ours)	**87.8**	**81.1**	**71.7**	**61.7**	**75.6**
DS-Llama-8B	44.5	37.8	31.7	26.7	35.2
DS-Llama-8B (Majority)	48.2	42.9	32.8	27.1	37.8
DS-Llama-8B (DV)	47.9	41.7	32.8	26.6	37.3
DS-Llama-8B (SCoP)	43.7	38.3	31.2	25.9	34.8
DS-Llama-8B (BoN)	52.8	44.8	34.0	28.1	40.0
DS-Llama-8B (Ours)	**54.3**	**47.0**	**38.4**	**31.2**	**42.7**
Qwen2.5-Coder-7B-Instruct	86.6	82.3	82.5	70.1	80.4
Qwen2.5-Coder-7B-Instruct (Majority)	88.1	82.6	83.9	69.5	81.0
Qwen2.5-Coder-7B-Instruct (DV)	88.1	82.1	83.6	69.1	80.7
Qwen2.5-Coder-7B-Instruct (SCoP)	87.1	82.2	81.7	68.4	79.9
Qwen2.5-Coder-7B-Instruct (BoN)	87.9	82.8	85.2	73.0	82.2
Qwen2.5-Coder-7B-Instruct (Ours)	**89.0**	**83.5**	**87.8**	**73.5**	**83.5**

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 three benchmarks (Math500, Olympiad, and AIME24) for illustration and visualize the results in Figure 3. On Math500 and Olympiad, our method uses approximately 40% to 60% of the maximum sampling budget, and even on the more challenging AIME24, it uses only about 70% of the maximum sampling budget. We also show in Figure 4 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 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 consistent comparison. We adopt EvalPlus[2] as the evaluation framework and assess three models (Qwen3-8B, DeepSeek-R1-Distill-Llama-8B, and Qwen2.5-Coder-7B-Instruct (Yang et al., 2024a)), 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+).

[2] https://github.com/evalplus/evalplus

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.

Table 3: This table presents an ablation study of our method on the Qwen3-8B model across multiple mathematical benchmarks, where accuracy (%) is reported per component.

Method	Math500	Olympiad	AIME24
Baseline	84.4	46.4	36.7
OnlyRewrite	90.4	59.0	56.7
OnlyMajority	87.6	48.6	40.0
Full Method	**92.8**	**61.9**	**66.7**

Table 4: Performance on Math500 with Qwen3-8B under different sampling budgets (4, 6, and 8).

Method	4 sampling	6 sampling	8 sampling
Majority	87.4	87.6	87.8
DV	86.1	87.6	87.3
SCoP	85.6	88.2	87.2
BoN	86.2	85.8	85.8
Ours	**91.6**	**92.8**	**91.8**

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.

To further evaluate the robustness of our method under different sampling budgets, we conduct additional experiments on Math500 with Qwen3-8B using 4 and 8 samplings, as shown in Table 4. To accommodate different budgets, we accordingly adjust the number of iterations in our framework. Specifically, under the 4-sampling setting, our method performs one iteration (two samplings) followed by a Rewrite & Rethink step, while under the 8-sampling setting, it performs three iterations (six samplings) followed by a Rewrite & Rethink step.

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 E. We further analyze this categorization, as shown in Figure 5 (Left).

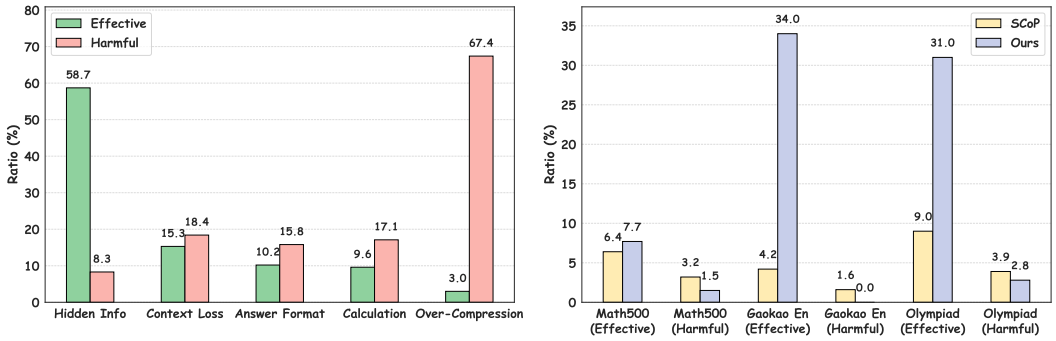


Figure 5: (Left) Effective and harmful rewriting ratios across different error types. (Right) Comparison between SCoP and our method on Qwen3-8B across Math500, Gaokao English, and OlympiadBench. Our method achieves higher effective and lower harmful rewriting rates.

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 Rewriting can also turn correct answers into incorrect ones. We find that this often occurs when a simple original question is excessively compressed during rewriting, removing or obscuring key conditions (**over-compression error**). Avoiding rewriting for short and clear questions therefore helps reduce such failures. We further compare rewriting effectiveness and harmfulness between SCoP and our method in Figure 5 (Right). Our disagreement-based routing identifies more appropriate cases for rewriting, increasing effective rewrites while reducing harmful ones.

6 Conclusion

This work proposes a training-free framework that formulates test-time scaling as an instance-level strategy routing problem, using model disagreement as a lightweight signal to guide the selection of inference strategies. By dynamically routing instances to different strategies, the framework applies more appropriate strategies to uncertain samples while avoiding unnecessary or harmful processing on easier ones. We evaluate our method on seven mathematical reasoning benchmarks using three models. The results show that it consistently improves accuracy while requiring fewer sampling runs than existing methods. We further examine the applicability of the framework to code generation tasks and provide an analysis of when rewriting is beneficial or detrimental.

AI USE STATEMENT

In this work, we used generative AI tools solely for language polishing and improving the readability of the manuscript. All AI-assisted revisions were reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide the implementation details necessary to reproduce our experiments throughout the paper and appendix. Section 4.1 specifies the evaluated models, datasets, baselines, sampling budgets, decoding hyperparameters, random seeds, context lengths, and computational resources. Section 3 and Algorithm 1 describe the complete inference procedure of our framework. Appendix C provides the reasoning and rewriting prompt templates, while Appendix D details the answer equivalence checking procedure.

REFERENCES

Aradhye Agarwal, Ayan Sengupta, and Tanmoy Chakraborty. The art of scaling test-time compute for large language models, 2025. URL https://arxiv.org/abs/2512.02008.

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with llms, 2023. URL https://arxiv.org/abs/2305.11860.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

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

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug, 2023. URL https://arxiv.org/abs/2304.05128.

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

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian

Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

Kuofeng Gao, Huanqia Cai, Qingyao Shuai, Dihong Gong, and Zhifeng Li. Embedding self-correction as an inherent ability in large language models for enhanced mathematical reasoning, 2025. URL https://arxiv.org/abs/2410.10735.

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

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

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

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

Yixin Ji, Juntao Li, Yang Xiang, Hai Ye, Kaixin Wu, Kai Yao, Jia Xu, Linjian Mo, and Min Zhang. A survey of test-time compute: From intuitive inference to deliberate reasoning, 2025. URL https://arxiv.org/abs/2501.02497.

Weize Kong, Spurthi Hombaiah, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. Prewrite: Prompt rewriting with reinforcement learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 594–601, 2024.

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

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

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need, 2024a. URL https://arxiv.org/abs/2402.05120.

Loka Li, Zhenhao Chen, Guangyi Chen, Yixuan Zhang, Yusheng Su, Eric Xing, and Kun Zhang. Confidence matters: Revisiting intrinsic self-correction capabilities of large language models, 2024b. URL https://arxiv.org/abs/2402.12563.

Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. Llm-r2: A large language model enhanced rule-based rewrite system for boosting query efficiency. *arXiv preprint arXiv:2404.12872*, 2024c.

Lei Lin, Jiayi Fu, Pengli Liu, Qingyang Li, Yan Gong, Junchen Wan, Fuzheng Zhang, Zhongyuan Wang, Di Zhang, and Kun Gai. Just ask one more time! self-agreement improves reasoning of language models in (almost) all scenarios, 2024. URL https://arxiv.org/abs/2311.08154.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, Yang Liu, and Yahui Zhou. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*, 2025.

Jie Liu and Barzan Mozafari. Query rewriting via large language models. *arXiv preprint arXiv:2403.09060*, 2024.

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

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Lei Shu, Liangchen Luo, Jayakumar Hoskere, Yun Zhu, Yinxiao Liu, Simon Tong, Jindong Chen, and Lei Meng. Rewritelm: An instruction-tuned large language model for text rewriting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 18970–18980, 2024.

Guijin Son, Jiwoo Hong, Hyunwoo Ko, and James Thorne. Linguistic generalizability of test-time scaling in mathematical reasoning, 2025. URL https://arxiv.org/abs/2502.17407.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

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

Yang Xiang, Yixin Ji, Ruotao Xu, Dan Qiao, Zheming Yang, Juntao Li, and Min Zhang. When is thinking enough? early exit via sufficiency assessment for efficient reasoning, 2026. URL https://arxiv.org/abs/2604.06787.

Mingfeng Xue, Dayiheng Liu, Wenqiang Lei, Xingzhang Ren, Baosong Yang, Jun Xie, Yidan Zhang, Dezhong Peng, and Jiancheng Lv. Dynamic voting for efficient reasoning in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3085–3104, 2023.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394*, 2024a.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu, Zhihan Guo, Yufei Wang, Niklas Muennighoff, Irwin King, Xue Liu, and Chen Ma. A survey on test-time scaling in large language models: What, how, where, and how well?, 2025a. URL https://arxiv.org/abs/2503.24235.

Ruiqi Zhang, Momin Haider, Ming Yin, Jiahao Qiu, Mengdi Wang, Peter Bartlett, and Andrea Zanette. Accelerating best-of-n via speculative rejection. In *ICML 2024 Workshop on Structured Probabilistic Inference {\&} Generative Modeling*, 2024b.

Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. Evaluating the performance of large language models on gaokao benchmark. *arXiv preprint arXiv:2305.12474*, 2023.

Yu Zhang, Shujun Peng, Nengwu Wu, Xinhan Lin, Yang Hu, and Jie Tang. Rm-pot: Reformulating mathematical problems and solving via program of thoughts. *arXiv preprint arXiv:2502.12589*, 2025b.

James Xu Zhao, Bryan Hooi, and See-Kiong Ng. Test-time scaling in reasoning models is not effective for knowledge-intensive tasks yet, 2026. URL https://arxiv.org/abs/2509.06861.

Yue Zhou, Yada Zhu, Diego Antognini, Yoon Kim, and Yang Zhang. Paraphrase and solve: Exploring and exploiting the impact of surface form on mathematical reasoning in large language models. *arXiv preprint arXiv:2404.11500*, 2024.

A LIMITATIONS

This work leaves room for further exploration in certain aspects.

- Our framework relies on output disagreement as a proxy for uncertainty, which may not fully capture all failure modes. In particular, cases with consistent but incorrect reasoning (false agreement) remain challenging.
- The effectiveness of rewriting is instance-dependent. While our routing strategy reduces harmful rewriting, it cannot fully eliminate cases where rewriting introduces deviations from correct reasoning.
- Our method currently focuses on routing among a limited set of test-time scaling strategies. Extending the framework to incorporate a broader range of strategies (e.g., search-based or verification-based methods) remains an open direction.

B EXPERIMENTAL RESULTS

Table 5: This table presents the evaluation results of three models on seven mathematical benchmarks in the 16K-context setting. The models include Qwen3-8B, Qwen3-4B, and DS-Llama-8B. Each model is evaluated under different reasoning enhancement strategies: baseline, majority voting, dynamic voting, SCoP, BoN, and our method.

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 (DV)	92.9	84.4	64.5	95.9	92.5	50.0	70.0	78.6
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 (DV)	95.0	84.7	66.3	94.9	92.5	55.0	66.7	79.3
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 (DV)	88.8	80.8	57.9	89.0	85.0	36.7	53.3	70.2
DS-Llama-8B (SCoP)	88.4	78.2	55.3	89.3	87.5	33.3	43.3	67.9
DS-Llama-8B (BoN)	87.2	79.4	57.1	90.5	85.0	36.7	50.0	69.4
DS-Llama-8B (Ours)	**91.0**	**81.2**	**58.5**	**90.8**	**92.5**	**43.3**	**60.0**	**73.9**

Table 6: Distribution of samples across NDS, MDS, and SDS for three models on seven benchmarks under the 8K-context setting. Values are reported in percentages.

Model / Samples	Math500	Gaokao En	Olympiad	GSM8K	AMC23	AIME25	AIME24
Qwen3-8B (NDS)	66.4	62.9	55.3	97.6	65.0	26.7	40.0
Qwen3-8B (MDS)	10.0	10.1	8.7	0.9	10.0	13.3	6.7
Qwen3-8B (SDS)	23.6	27.0	36.0	1.5	25.0	60.0	53.3
Qwen3-4B (NDS)	87.4	81.3	55.3	96.0	67.5	26.7	43.3
Qwen3-4B (MDS)	3.8	7.0	7.3	2.0	2.5	6.7	6.7
Qwen3-4B (SDS)	8.8	11.7	37.4	2.0	30.0	66.6	50.0
DS-Llama-8B (NDS)	63.2	54.3	43.1	69.6	70.0	20.0	26.7
DS-Llama-8B (MDS)	12.2	17.1	13.5	14.3	10.0	0.0	16.7
DS-Llama-8B (SDS)	24.6	28.6	43.4	16.1	20.0	80.0	56.6

Table 7: 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.

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

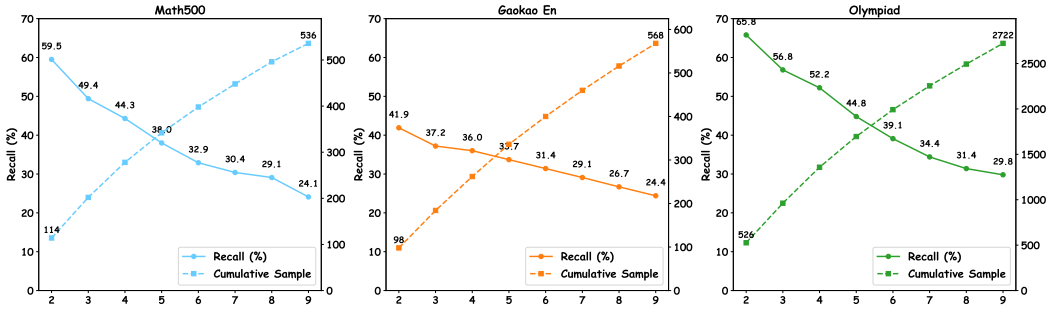


Figure 6: The figure shows the recall (%) of incorrect samples as a function of the number of iterations N on three datasets: Math500, Gaokao En, and Olympiad. Solid lines represent the recall achieved by our iterative filtering method, while dashed lines indicate the cumulative number of samples used at each iteration. The figure shows that as the number of iterations increases, the recall decreases while the cumulative number of samples increases.

Table 8: This table reports conditional accuracies $P(\text{correct} \mid \text{NDS/MDS/SDS})$ on Olympiad, Gaokao En, and Math500 using Qwen3-8B. The results show a clear accuracy gradient across partitions: NDS instances achieve higher accuracy, while SDS instances exhibit substantially lower accuracy, indicating that disagreement is strongly associated with the likelihood of correctness.

Datasets	NDS	MDS	SDS	NDS → SDS Drop
Olympiad	63.4	11.1	4.1	↓ 59.3
Gaokao En	87.3	52.6	20.0	↓ 67.3
Math500	94.2	65.2	33.3	↓ 60.9

Table 9: This table compares the effectiveness of different strategies across disagreement-based partitions on Olympiad, Gaokao En, and Math500 using Qwen3-8B. The results show that different disagreement levels favor different strategies: NDS require little additional sampling, MDS benefit more from majority voting, and SDS benefit more from rewriting.

Method	Olympiad			Gaokao En			Math500		
	NDS	MDS	SDS	NDS	MDS	SDS	NDS	MDS	SDS
Baseline	63.4	11.1	4.1	87.3	52.6	20.0	94.2	65.2	33.3
Majority	65.4	44.8	4.1	88.0	63.2	28.0	96.1	68.0	39.7
Rewrite	61.1	38.1	27.8	85.8	55.2	58.0	93.4	66.5	68.3

Table 10: Accuracy (%) of dynamic voting (DV) under different thresholds on three benchmarks.

Threshold	Math500	Olympiad	Gaokao En
0.5	84.0	44.9	75.1
0.6	87.4	49.2	79.2
0.7	87.6	49.5	79.7
0.8	87.6	49.2	79.7
0.9	87.6	49.0	79.5

Table 11: Comparison of different methods on three benchmarks using Qwen3-32B.

Method	Math500	Olympiad	Gaokao En
Baseline	87.8	50.5	79.5
Majority	88.2	54.3	80.7
DV	88.2	54.0	80.0
SCoP	89.5	57.7	82.3
BoN	88.6	55.6	79.7
Ours	**93.1**	**62.8**	**84.2**

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 Answer Equivalence Checking

To determine whether two answers are equivalent, we adopt a hierarchical matching procedure that goes beyond exact string matching.

Normalization. We first normalize both prediction and reference by removing formatting artifacts such as extra whitespace, brackets, and casing differences. Structured outputs are converted into consistent canonical forms.

Numerical Equivalence. When both answers can be parsed as numeric values, we perform tolerance-based comparison. We additionally account for common variations such as percentage and scale transformations (e.g., x, $x/100$, and $100x$).

Structural Matching. For structured outputs such as tuples, lists, and matrices, we perform element-wise comparison recursively.

Symbolic Equivalence. When numerical comparison is not applicable, we apply symbolic matching. In particular, equations are first normalized into residual forms (e.g., $x = y$ to $x - y = 0$) before symbolic equivalence checking.

This hierarchical procedure allows us to capture a broad class of mathematically equivalent answers beyond strict string matching, while remaining efficient in practice.

E CASE STUDY

Case	Original	Rewritten
hidden info error	**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 $kS_1 \geq S$. **Answer:** 1 **Output:** $\cdots$ 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 $\cdots$ Final Answer $\boxed{\frac{1}{2}}$	**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 $k\,S_1 \geq S$ holds for all configurations. **Answer:** 1 **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 $ABCD$ $\cdots$ Final Answer $\boxed{1}$
answer format error	**Question:** Given sets $M = \{x\|x + 2 \geq 0\}, N = \{x\|x-1 < 0\}$, find $M \cap N$. **Answer:** $\{x\| - 2 \leq x < 1\}$ **Output:** To find the $\cdots$ this is: $[-2, 1)$ Final Answer $\boxed{[-2, 1)}$	**Question:** $M = \{x\|x+2 \geq 0\}, N = \{x\|x-1 < 0\}$, find $M \cap N$. **Answer:** $\{x\| - 2 \leq x < 1\}$ **Output:** To find the $\cdots$ but not including, Final Answer $\boxed{[-2, 1)}$
context loss error	**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 $x > A - 1$. **Answer:** $\frac{79}{128}$ **Output:** Let the triangle $\cdots$ For a fixed y, let $x_{min} = 1 \cdots$ Since x ranges from 1 to 17, we compute $P(x > A - 1) = \frac{17-(A-1)}{16}$ $\cdots$	**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 $(17, 1)$. **Answer:** $\frac{79}{128}$ **Output:** Let (x, y) be uniformly $\cdots$ The area of the triangle is $S = \frac{1}{2}(17 - 1)(7 - 1) = 48$ $\cdots$ we compute $P(x > A - 1) = \frac{17-(A-1)}{16}$ $\cdots$
calculation error	**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 $n = 6$. **Answer:** 6.62 **Output:** $\cdots$ Sixth energy level ($n = 6$): $E_6 = -\frac{13.6}{36} \approx -0.3778$eV. Energy difference required to excite the electron $\Delta E = E_6 - E_1 = -0.3778 - (-13.6) = 13.2222$eV $\cdots$ Solving for V: $V = 13.22/2 = 6.61V$. Final Answer $\boxed{6.61}$	**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 ($n = 6$). **Answer:** 6.62 **Output:** $\cdots$ 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 $\cdots$ Final Answer $\boxed{6.61}$
over-compression error	**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? **Answer:** 0.8 **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. $\cdots$ Final Answer $\boxed{0.8}$	**Question:** If someone signs up for the football club, what is the probability they also signed up for the table tennis club? **Answer:** 0.8 **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. $\cdots$ However, no specific data or probabilities are provided in the question. Without knowing $\cdots$ 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 $\cdots$ the probability would be: $\boxed{\frac{1}{3}}$

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 $r_1, r_2, \ldots, r_n$ 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 $\boxed{16}$

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 $r_1, r_2, \ldots, r_n$ 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:

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

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 $n \geqslant 3$ is given. We call an n-tuple of real numbers $(x_1, x_2, \ldots, x_n)$ *Shiny* if for each permutation $y_1, y_2, \ldots, y_n$ of these numbers we have $\sum_{i=1}^{n-1} y_i y_{i+1} = y_1y_2 + y_2y_3 + \cdots + y_{n-1}y_n \geqslant -1$. Find the largest constant $K = K(n)$ such that $\sum_{1 \leqslant i < j \leqslant n} x_i x_j \geqslant K$ holds for every Shiny n-tuple $(x_1, x_2, \ldots, x_n)$.

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

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

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 $(x_1, x_2, \ldots, x_n)$ *Shiny* if for every permutation $(y_1, y_2, \ldots, y_n)$ of these numbers, the following inequality holds: $y_1y_2 + y_2y_3 + \cdots + y_{n-1}y_n \geq -1$. For every Shiny n-tuple, consider the sum of all pairwise products $S = \sum_{1 \leq i < j \leq n} x_ix_j$. 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_ix_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 $(x_1, x_2, \ldots, x_n)$ 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:

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

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

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

Rearranging gives:

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

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 $\sum x_ix_j$ 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 $(\varepsilon, \varepsilon, \ldots, \varepsilon, -c)$, with $n-1$ copies of ε and one $-c$. We analyze the sum of pairwise products:

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

After simplification, this becomes: $\sum x_ix_j = \frac{n-1}{2} \left(\varepsilon^2 - 1 \right)$ 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 $\varepsilon \to 0$, and we can set c such that the sum of adjacent products is exactly -1. Asymptotic Behavior and Conclusion. As $\varepsilon \to 0$, the sum of pairwise products approaches: $\sum x_ix_j \to \frac{n-1}{2} \cdot (-1) = -\frac{n-1}{2}$ 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 $\boxed{-\frac{n-1}{2}}$