

RIRM: Reflection Inhibition Reward Mechanism for Mitigating Overthinking in Large Reasoning Models

Anonymous ACL submission

Abstract

Large Reasoning Models (LRMs) achieve significant performance gains across various domains by extending the Chain-of-Thought (CoT) length during inference. However, they frequently exhibit a tendency toward overthinking—generating unnecessary reasoning steps after already reaching a correct answer—which undermines both efficiency and performance. To address this issue, we propose the Reflection Inhibition Reward Mechanism (RIRM), a reinforcement learning-based method designed to suppress excessive post-answer reflection. Specifically, RIRM identifies the position of the first correct answer and subsequent reflections within the CoT. It then optimizes a well-designed reward function that guides reinforcement learning toward accurate yet computationally efficient reasoning. Extensive experiments on mathematical and scientific benchmarks show that RIRM reduces token consumption by up to 69.06% while improving accuracy by up to 11.37 percentage points, yielding a more favorable efficiency–accuracy tradeoff. Code will be released shortly.

1 Introduction

Large Reasoning Models (LRMs)—such as OpenAI’s o1 (OpenAI, 2024), DeepSeek-R1 (Guo et al., 2025), QwQ (Team, 2025), and Kimi-k1.5 (Team et al., 2025)—have demonstrated strong reasoning capabilities on complex tasks through Test-Time Scaling (TTS). By dynamically allocating additional computation during inference, these models generate extended Chain-of-Thought (CoT) sequences, leading to notable improvements on challenging mathematical and logical benchmarks.

As research deepens, extensive work (Qu et al., 2025) reveals a critical limitation of LRMs: overthinking. After arriving at a correct solution, these models persistently generate verbose, repetitive, or irrelevant reasoning steps. This behavior not only wastes computational resources but can also

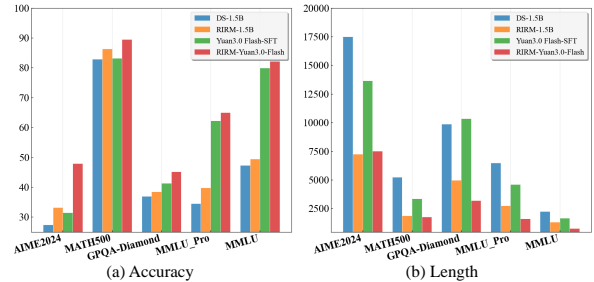


Figure 1: RIRM effectively mitigates overthinking and improves the reasoning performance of DeepseekR1-Distill-Qwen-1.5B (DS-1.5B) and Yuan3.0 Flash-SFT across multiple benchmarks.

degrades model performance. Chen et al. (2024) characterize overthinking as a state in which models expend substantial computation on intermediate steps with diminishing marginal returns. Subsequent work by Wei et al. (2025) and Zhao et al. (2025) further shows that such redundancy typically occurs after the first correct answer, where models enter cycles of repetitive verification.

From a reinforcement learning perspective, overthinking arises from a mis-specified reward signal and imperfect credit assignment along the reasoning trajectory. Standard training objectives primarily reward final answer correctness, providing little supervision on when the model should stop reasoning. Consequently, reasoning steps generated after the first correct answer are not penalized and can even accumulate positive reward through spurious correlations with success, leading to reward hacking in the form of excessive reflection. This suggests that incorporating trajectory-level awareness into reward design can help differentiate solution-relevant reasoning from post-answer reflection, enabling more efficient inference-time computation.

To address this reward misalignment, we propose a Reflection Inhibition Reward Mechanism (RIRM) that explicitly regulates reasoning trajectories by penalizing redundant post-answer reflec-

tion. The core idea of RIRM is to provide fine-grained, trajectory-level reward signals that distinguish solution discovery from unnecessary verification, thereby improving credit assignment during policy optimization. Concretely, RIRM identifies two critical positions in CoT: (i) the step where the correct answer first emerges, and (ii) the boundary of subsequent reflection steps. Based on these positions, the reasoning trajectory is evaluated along three dimensions: successful identification of the first correct answer, the extent of post-answer reflection, and final answer correctness. These criteria are combined into a unified reward signal that guides reinforcement learning toward accurate yet computation-efficient reasoning.

Experiments conducted on DeepSeek-R1-Distill-Qwen-1.5B (Guo et al., 2025) and Yuan3.0 Flash-SFT (Wu et al., 2026) across mathematical and scientific reasoning benchmarks (Figure 1) show that RIRM reduces token length by up to 69.06% while improving accuracy by up to 11.37 percentage points. These results indicate that RIRM effectively mitigates overthinking, enhances baseline performance and generalizes across model scales.

In summary, our contributions are threefold:

- We provide a systematic analysis of the reasoning trajectories of large reasoning models and identify redundant post-answer reflection as the primary cause of overthinking.
- We propose a novel Reflection Inhibition Reward Mechanism (RIRM), which identifies the first correct answer and subsequent reflection steps, and designs a three-dimensional reward for end-to-end optimization under the RLVR framework.
- Extensive experiments across multiple base models and diverse reasoning tasks demonstrate that RIRM effectively reduces overthinking and advances a practical paradigm for efficient reasoning in LRMs.

2 Related Work

Large Reasoning Models (LRMs), such as OpenAI’s o1 (OpenAI, 2024), DeepSeek-R1 (Guo et al., 2025), and Kimi-k1.5 (Team et al., 2025), employ reinforcement learning with verifiable rewards (RLVR) to produce extended reasoning chains. However, RLVR tends to encourage models to generate redundant reasoning steps—a phenomenon

known as overthinking—thereby consuming unnecessary computational resources and potentially degrading performance. Existing research addressing overthinking can be categorized into the following three types (Sui et al., 2025).

Prompt-Based Optimization. Prompt-based optimization guides LRMs toward more concise answers via engineered prompts (Huang et al., 2025; Xu et al., 2025; Han et al., 2025). While straightforward, their effectiveness relies heavily on prompt design, and imposed length constraints often compromise reasoning accuracy.

Inference-Time Reasoning Control. Reasoning output optimization intervenes during inference to induce early termination (Wei et al., 2025; Fan et al., 2025; Yang et al., 2025b). These methods, however, depend on heuristic rules that may lack generality, fail to resolve overthinking fundamentally, and risk interrupting valid reasoning chains.

Post-Training Optimization. Post-training methods can be categorized into supervised fine-tuning (SFT) and reinforcement learning (RL). SFT-based methods instill efficient reasoning capabilities by training on short-chain-of-thought datasets (Ma et al., 2025; Ye et al., 2025; Yu et al., 2025a; Zhao et al., 2025; Liu et al., 2025) but they rely on high-quality curated data and often generalize poorly across diverse tasks. RL-based methods typically incorporate explicit length penalties (Team et al., 2025; Shen et al., 2025; Yeo et al., 2025; Cheng et al., 2025), length-aware rewards (Arora and Zanette, 2025; Yi et al., 2025; Tu et al., 2025), or complex stepwise verification mechanisms (Yue et al., 2025). However, these strategies tend either to oversimplify the optimization objective or to incur high computational cost and low training efficiency.

Different from prior work, we identify overthinking as primarily arising from redundant post-answer verification—where models continue reasoning even after the first correct answer has emerged. Building on this insight, we propose the Reflection Inhibition Reward Mechanism (RIRM). Experiments demonstrate that RIRM achieves up to 69.09% token compression while preserving accuracy, with training efficiency substantially exceeding that of existing methods.

3 Methodology

3.1 LRM Reasoning Process Analysis

In complex reasoning tasks, particularly in mathematical and scientific domains, large reasoning models (LRMs) exhibit pronounced overthinking behavior. As illustrated in Figure 2(a), even after producing a correct answer, these models frequently continue generating reasoning steps in repetitive self-verification loops, a phenomenon commonly referred to as reflection. Empirical analysis on the DeepSeek-R1-Distill-Qwen-1.5B and DeepSeek-R1 models across the AIME 2024 and MATH-500 benchmarks reveals that a substantial portion of inference-time computation—up to 71.6% of total tokens—is consumed during this post-answer reflection stage (Figure 2(b)).

These findings suggest that mitigating overthinking is fundamentally a problem of regulating the reasoning trajectory rather than merely shortening outputs. In particular, effective control requires identifying the **first correct answer**—the earliest point at which the model reaches a valid solution—and distinguishing solution-critical reasoning from subsequent post-answer reflection. Without such differentiation, redundant verification steps are treated on par with productive reasoning and continue to accumulate computation. This motivates a mechanism that explicitly suppresses unproductive reflection after solution discovery, enabling models to terminate reasoning in a principled manner while preserving correctness. Such trajectory-aware control naturally aligns inference-time computation with task-level utility and forms the basis of our proposed Reflection Inhibition Reward Mechanism (RIRM). Figure 3 illustrates the workflow of RIRM, which consists of two main processes: Correct Answers Identification and Reward Calculation. These will be detailed below.

3.2 Reflection Inhibition Reward Mechanism

Correct Answers Identification. To suppress overthinking in a principled manner, it is essential to explicitly model the internal structure of the reasoning trajectory generated by LRMs, i.e., the content enclosed between the `<think>` and `</think>` tags. In particular, effective reward assignment requires precise identification of two critical components: (1) the earliest step at which the model first arrives at a correct solution, referred to as the *first correct answer*, and (2) the boundaries

of subsequent reflection steps that occur after this point. Distinguishing these stages is crucial for separating solution-critical reasoning from redundant post-answer verification, which underlies the overthinking phenomenon.

To achieve robust and scalable segmentation of reasoning trajectories, we adopt a fixed-window text segmentation strategy combined with external model-assisted annotation. This design is motivated by two considerations. First, reasoning chains in LRMs can be long and structurally unregularized, making rule-based parsing brittle and unreliable. Second, directly annotating step-level correctness within a single long sequence is challenging due to context dilution and error accumulation. Segmenting the reasoning text into fixed-length windows enables localized analysis while preserving the temporal order of reasoning steps.

The annotation workflow proceeds as follows: (1) Text Segmentation. The reasoning text is split into consecutive, non-overlapping segments of fixed length (e.g., 1000 characters). Each segment is processed sequentially to maintain the original reasoning order. (2) External Model Annotation. Each segment, together with the original question and its ground-truth answer, is provided to an external reasoning-capable model. Guided by explicit prompt instructions, the external model determines whether the segment contains a valid occurrence of the correct answer. If a segment contains n such occurrences, the model outputs n corresponding statements, and a `</verify>` tag is appended after each statement containing the correct answer. (3) Tag Conversion. After all segments are annotated, the earliest `</verify>` tag in the reconstructed reasoning sequence is replaced with `<1st_answer>`, marking the position of the first correct answer.

This procedure yields a structured and unambiguous annotation of the reasoning trajectory: the `<1st_answer>` tag identifies the transition point from solution discovery to post-answer reasoning, while subsequent `</verify>` tags delineate the boundaries of reflection intervals. Such fine-grained structural annotations provide a reliable operational basis for computing reflection-aware reward signals in RIRM. Detailed prompt templates used for external model annotation are provided in the Appendix C.

Reward Calculation. After annotating the `<1st_answer>` and `</verify>` tags in the reasoning process, we compute the *Reflection Inhibition*

Question:
The sum of the first N positive odd numbers is 121. What is the value of N?

Reasoning:
<think>

Okay, so I need to find the value of N where the sum of the first N positive odd numbers is 121. Hmm, let me think about how to approach this. First, I remember that the sum of the first N odd numbers has a formula. Is it something like N squared? Wait, let me verify that. Yes, if that's the case, then if the sum is 121, then N squared must be 121. Therefore, N is the square root of 121. Let me compute that. Square root of 121 is 11, because 11 times 11 is 121. So, N should be 11.

Is that right? Wait, let me double-check. Let me add up the first 11 odd numbers to make sure I get 121. 1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 + 17 + 19 + 21. Let me add them step by step: 1 + 3 = 4, 4 + 5 = 9, 9 + 7 = 16, 16 + 9 = 25, 25 + 11 = 36, 36 + 13 = 49, 49 + 15 = 64, 64 + 17 = 81, 81 + 19 = 100, 100 + 21 = 121. Yes, that adds up to 121. So, N is indeed 11.

But wait, just to make sure I didn't make a mistake in my initial thought process, let me think about why the sum of the first N odd numbers is N squared. Maybe that's a formula I should recall or derive. So, the first odd number is 1, which is 1 squared. Alternatively, I can write the sum as $\sum_{k=1}^N (2k-1)$. Which is equal to $2\sum_{k=1}^N k - \sum_{k=1}^N 1$, where the first sum is from k=1 to N of k, and the second sum is from k=1 to N of 1. So, that becomes $2 \cdot \frac{N(N+1)}{2} - N = N^2 + N - N = N^2$. So, yes, that works out. So, the sum is N squared. Therefore, if the sum is 121, N is 11.

Wait, another way to think about it is that each odd number can be represented as $2k-1$, so when you add them up, it's like $1 + 3 + 5 + \dots + (2N-1) = N^2$. So, that's another way to see it. So, in this problem, since the sum is 121, which is 11 squared, N is 11.

.....

I think that's thorough enough. I don't see any errors in my reasoning. So, the value of N is 11.

Final Answer The value of N is 11.
</think>

Summary:
The sum of the first N positive odd numbers is given by the formula N^2 . We need to find N such that the sum is 121. First, we recognize that the sum of the first N positive odd numbers is N^2 . Therefore, if the sum is 121, we have $N^2 = 121$. Taking the square root of both sides, we get $N = \sqrt{121}$. Since the square root of 121 is 11, we find $N = 11$. To verify, we can add the first 11 odd numbers: $1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 + 17 + 19 + 21 = 121$. Thus, the value of N is 11.

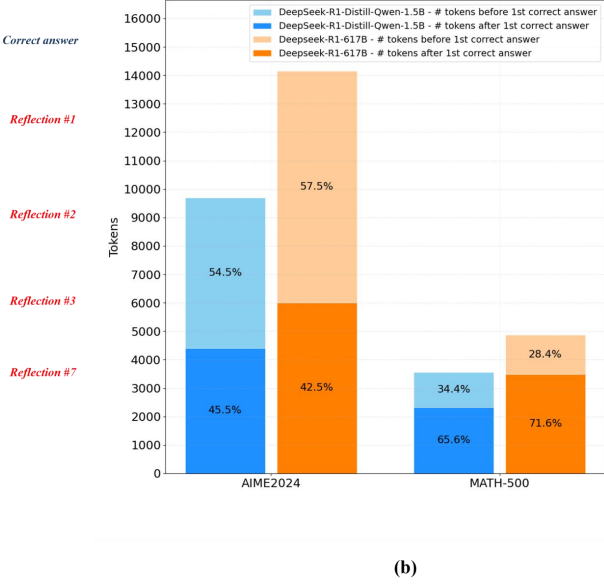


Figure 2: (a) An example of repetitive “reflection” behavior in DeepSeek-R1-Distill-Qwen-1.5B reasoning output after the “first answer” is reached. (b) Breakdown of average token consumption for DeepSeek-R1-Distill-Qwen-1.5B and DeepSeek-R1 on the AIME 2024 and MATH-500 benchmarks.

Reward \mathcal{R} , which provides trajectory-aware supervision for reinforcement learning. To enable effective credit assignment, \mathcal{R} is decomposed into three complementary components: R_{ans} , indicating whether the first correct answer is successfully identified; R_{ver} , measuring the appropriateness of post-answer reflection; and R_{acc} , capturing final answer correctness. This decomposition explicitly separates solution discovery, reflection control, and task success, thereby aligning reasoning efficiency with task-level utility.

If the `<1st_answer>` tag is missing, the reasoning trajectory is treated as invalid, as the model either fails to reach a correct solution or produces unstructured output. In this case, we uniformly penalize the trajectory by assigning $[R_{ans} = -1, R_{ver} = -1, R_{acc} = -1]$. If the reasoning process terminates immediately after the `<1st_answer>` tag—i.e., the output contains `</think>` but no `</verify>`—the model correctly stops reasoning upon solution discovery. In this case, reflection behavior is considered optimal, and we set $R_{ver} = R_{acc}$, yielding $[R_{ans} = 1, R_{ver} = R_{acc}, R_{acc}]$. If the `<1st_answer>` tag appears but the output lacks the closing `</think>` tag, indicating incomplete or truncated reasoning, we assign $[R_{ans} = 1, R_{ver} = -1, R_{acc} = -1]$.

When both `<1st_answer>` and one or more `</verify>` tags are present, we assign $[R_{ans} =$

$1, R_{ver}, R_{acc}]$, where R_{ver} is determined by the number of `</verify>` tags. Let v denote the number of `</verify>` tags. To account for task difficulty and answer ambiguity, we estimate the empirical pass rate p over G sampled responses for the same prompt,

$$p = \frac{1}{G} \sum_{i=1}^G \mathbb{I}(R_{acc}^{(i)} = 1). \quad 300$$

When $p \geq 50\%$, fixed reflection bounds $V_{min} = 2$ and $V_{max} = 10$ are applied, reflecting that excessive verification is unlikely to be beneficial for relatively easy problems. When $p < 50\%$, adaptive bounds are defined using the observed minimum and maximum numbers of `</verify>` tags among correct responses, allowing additional reflection for more challenging cases.

The reflection reward is computed as

$$R_{ver}(v) = \begin{cases} 1, & \text{if } v \leq V_{min}, \\ 1 - \frac{(v - V_{min})}{(V_{max} - V_{min})}, & \text{if } V_{min} < v \leq V_{max}, \\ 0, & \text{if } V_{max} < v. \end{cases} \quad 310$$

This piecewise formulation softly penalizes excessive reflection rather than imposing a hard cutoff, reducing the risk of suppressing necessary reasoning. When $v = 0$, we set $R_{ver} = R_{acc}$ to avoid penalizing trajectories that correctly terminate immediately after solution discovery. If $V_{max} \leq 2$, we directly set $R_{ver} = 1$ to prevent degenerate

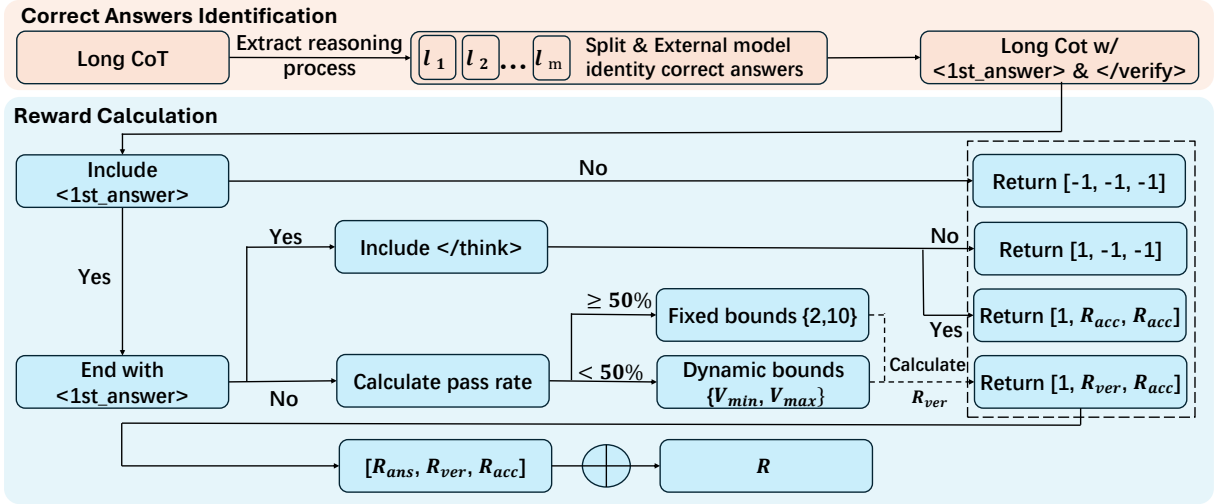


Figure 3: Workflow of Reflection Inhibition Reward mechanism (RIRM). It consists of two main processes: Correct Answers Identification and Reward Calculation. The former identifies the first correct answer and subsequent reflection segments within the reasoning trajectory, while the latter assigns a trajectory-aware reward to penalize redundant reflection during reinforcement learning.

scaling effects. A detailed algorithmic description of R_{ver} computation is provided in Appendix A.

Finally, the overall Reflection Inhibition Reward is defined as the sum of the three components,

$$\mathcal{R} = R_{ans} + R_{ver} + R_{acc}.$$

This unified reward balances answer correctness and reasoning efficiency, guiding the model to suppress redundant post-answer reflection while preserving solution-critical reasoning.

4 Experiments

4.1 Experiments Setup

Models. We use Deepseek-R1-Distill-Qwen-1.5B (Guo et al., 2025) and Yuan3.0 Flash-SFT (Wu et al., 2026) as base models. These two representative LRMs have demonstrated strong performance in mathematical and scientific reasoning tasks.

Dataset. For mathematical training, we use the OpenMathReasoning dataset (Moshkov et al., 2025); for scientific training, the Nemotron-Science-v1 dataset (Corporation, 2025) and the physics, chemistry, and biology subsets of the CAMEL dataset (Li et al., 2023) are employed. We exclude binary-choice, multiple-choice, multi-part, formal-proof, and open-ended questions, retaining only algorithmically parsable numerical problems. To prevent data leakage, similarity-based deduplication is applied against the benchmarks included in

the present study—AIME 2024, MATH-500, GPQA-Diamond, MMLU, and MMLU_Pro. After this process, a total of 60K high-quality mathematical training samples and 40K scientific training samples are obtained.

Evaluation and Metrics. Our evaluation uses mathematical and scientific reasoning benchmarks: AIME 2024, MATH-500, GPQA-Diamond, MMLU, and MMLU_Pro. We adopt an inference length limit of 24,576 tokens (beyond the 16,384-token training limit) to avoid truncation. Hyperparameters follow DeepSeek-R1 (temperature=0.6, top-p=0.95). Metrics are the average pass@1 over N runs, with N=1 for MMLU and MMLU_Pro, 8 for MATH-500, and 64 for AIME and GPQA-Diamond to control variance.

We consider Accuracy(Acc), Token Length(Tok) and Accuracy-Efficiency Score(AES) as evaluation metrics. Accuracy denote the final answer accuracy, we report the pass@1 performance. Token Length denote the average output length per sample in each benchmark. Accuracy-Efficiency Score is a composite metric which balances output brevity against answer accuracy (Luo et al., 2025). It rewards models that maintain high accuracy while reducing output length, with higher values reflecting a more favorable balance. See Appendix B for its formal definition and hyperparameters.

Implementation Details. Our method is implemented on the VeRL framework (Sheng et al., 2025), with the RIRM built upon the DAPO algo-

rithm; this implementation is referred to as RIRM-DAPO in the paper. During training, we sample 8 rollouts per instance with temperature 1.0 and top-p 1.0, and disable the length penalty typically used in DAPO. We use a batch size of 512, a mini-batch size of 64, an output length limit of 16K tokens, and a fixed learning rate of 1e-6 for Deepseek-R1-Distill-Qwen-1.5B and 5e-7 for Yuan3.0 Flash-SFT. The mathematical task is trained for 1 epoch, and the scientific task for 2 epochs. All other hyperparameters follow the original DAPO configuration. For the Deepseek-R1-Distill-Qwen-1.5B model, training is performed on 8 nodes, each equipped with 8 GPUs. The mathematical task requires approximately 19.77 hours, and the scientific task 27.67 hours. For the Yuan3.0 Flash-SFT model, we use 32 nodes (8 GPUs per node), with training times of about 33.65 hours and 30.28 hours for the mathematical and scientific tasks, respectively. We employ the Qwen3-32B model (Yang et al., 2025a) as an external model to annotate the <1st_answer> and </verify> tags. For the Deepseek-R1-Distill-Qwen-1.5B experiment, its inference runs on an additional 8 nodes (8 GPUs per node); for the Yuan3.0 Flash-SFT experiment, inference is conducted on an additional 16 nodes of the same configuration.

Baselines. We compare RIRM-DAPO with the following representative efficient reasoning methods. (1) Deepseek-R1-Distill-Qwen-1.5B (Guo et al., 2025): Model obtained by distillation from the Qwen2.5-1.5B model using larger DeepSeek-R1 models. (2) Yuan3.0 Flash-SFT (Wu et al., 2026): A 40B MoE model (3.7B activated) optimized for enterprise use. Its SFT version integrates high-quality data that unifies deep-thinking and fast-thinking paradigms. (3) DAPO (Yu et al., 2025b): In the DAPO experiments, we apply its built-in Soft Overlong Punishment to control output length, with the output limit set to 16,384 tokens and a punishment interval of 4,096 tokens. (4) VSRM-PPO (Yue et al., 2025): VSRM introduces a verifiable stepwise reward mechanism that assigns stepwise rewards to specific positions based on the performance of intermediate states along the reasoning trajectory. Like RIRM, it is a reward mechanism designed for the reasoning process. We therefore include it as a key baseline for comparison with our method. To ensure a fair comparison, all experiments are conducted using the same training dataset and trained for the same number of epochs. For DAPO and VSRM-PPO, we report the

Table 1: Performance comparison for all methods across AIME 2024 and MATH-500. “Acc” denotes the accuracy, “Tok” denotes the output length in tokens, and “AES” denotes the “Accuracy-Efficiency Score”.

Methods	AIME 2024			MATH-500		
	Acc↑	Tok↓	AES↑	Acc↑	Tok↓	AES↑
Deepseek-R1-Distill-Qwen-1.5B						
Base	27.45	17514	-	82.89	5277	-
VSRM-PPO	30.57	17145	0.36	84.55	5495	0.01
DAPO	34.17	12150	1.04	88.78	3778	0.49
RIRM-DAPO	33.18	7245	1.21	86.33	1877	0.77
Yuan3.0 Flash-SFT						
Base	31.45	13656	-	83.20	3362	-
DAPO	46.35	13781	1.41	89.06	3974	0.03
RIRM-DAPO	47.92	7505	2.02	89.47	1777	0.70

scores from the best-performing checkpoint.

4.2 Main Results

As shown in Table 1, we conduct a comparison among RIRM-DAPO, DAPO, and VSRM-PPO on mathematical benchmarks. Built upon DeepSeek-R1-Distill-Qwen-1.5B, RIRM-DAPO yields a substantially greater reduction in output length on both AIME 2024 and MATH-500 than both VSRM-PPO and DAPO. Notably, on MATH-500, the output length is reduced from 5,227 to 1,877 tokens, corresponding to a 64.09% decrease. When Yuan3.0 Flash-SFT is used as the base model, RIRM-DAPO consistently reduces output length on AIME 2024 and MATH-500 and improves accuracy on AIME 2024 by 11.37 pp over the base model. Across all mathematical benchmarks and base models, RIRM-DAPO achieves substantially higher Accuracy-Efficiency Scores (AES) than the other methods.

Similar trends are observed in Table 2 on scientific benchmarks. With DeepSeek-R1-Distill-Qwen-1.5B as the base model, RIRM-DAPO significantly compresses output length across all tasks, outperforming VSRM-PPO and DAPO, while consistently improving accuracy over the base model. On MMLU_Pro, accuracy increases by 5.1 pp. When Yuan3.0 Flash-SFT is adopted, RIRM-DAPO achieves greater output length reduction than DAPO while maintaining comparable accuracy improvements. The most pronounced reduction occurs on GPQA-Diamond, where the output length decreases from 10,337 to 3,198 tokens (69.06%). Across all scientific benchmarks, RIRM-DAPO consistently attains the highest AES.

Overall, these results indicate that RIRM-DAPO effectively eliminates redundant reflection steps

Table 2: Performance comparison for all methods across GPQA-Diamond, MMLU and MMLU_Pro. “Acc” denotes the accuracy, “Tok” denotes the output length in tokens, and “AES” denotes the “Accuracy-Efficiency Score”.

Methods	GPQA-Diamond			MMLU			MMLU_Pro		
	Acc↑	Tok↓	AES↑	Acc↑	Tok↓	AES↑	Acc↑	Tok↓	AES↑
Deepseek-R1-Distill-Qwen-1.5B									
Base	36.39	9906	-	47.28	2219	-	34.67	6371	-
VSRM-PPO	36.55	9935	-0.02	46.97	2260	-0.05	34.56	6492	-0.02
DAPO	39.11	8374	0.38	49.53	2649	-0.05	40.28	6144	0.04
RIRM-DAPO	38.48	4970	0.67	49.50	1324	0.54	39.77	2752	1.01
Yuan3.0 Flash-SFT									
Base	41.32	10337	-	79.98	1662	-	62.25	4599	-
DAPO	43.36	7085	0.46	82.43	1089	0.44	64.18	3178	0.40
RIRM-DAPO	45.14	3198	0.97	82.22	762	0.63	65.03	1616	0.78

and achieves a superior accuracy–efficiency trade-off, demonstrating its effectiveness in mitigating the overthinking problem.

4.3 Ablation Study

Based on the DeepSeek-R1-Distill-Qwen-1.5B model, we design ablation studies to evaluate the contribution of each component in RIRM. All experiments are trained for 1 epoch on mathematical data and are evaluated on AIME 2024 and MATH-500 benchmarks. The results are summarized in Table 3.

Effect of Reflection-Count Penalty Boundaries.

To examine the effect of the reflection-count penalty boundaries, we perform an ablation comparing three configurations: (1) applying fixed boundaries ($V_{min} = 2$, $V_{max} = 10$) to all samples; (2) using only sample-wise statistically determined dynamic boundaries; and (3) removing the minimum upper-bound constraint from RIRM-DAPO (i.e., “If $V_{max} \leq 2$, $R_{ver} = 1$.”) As shown in Table 3, the fixed-boundary configuration strikes a good balance between accuracy and length reduction, but it tends to over-suppress reflections on harder problems, causing a slight accuracy drop. The fully dynamic boundaries yield substantial length reduction at the cost of severe accuracy degradation, as they penalize reasonable reflection steps even on easier samples. Removing the minimum upper-bound constraint results in higher penalties for difficult samples—whose reflection counts have become low and concentrated in later training—which in turn encourages longer outputs and increases overall length. These observations confirm that our proposed boundary-selection mechanism effectively balances accuracy and efficiency in most scenarios.

Effect of the Pass-Rate Threshold To validate the effectiveness of the pass-rate threshold, we ex-

amine how different threshold values affect model accuracy and output length compression. We adjust the original 50% threshold in RIRM to 40% and 60% for comparison; results are shown in Table 3. Both higher and lower thresholds degrade performance. When the threshold is set to 40% or 60%, an imbalance in reward signals prevents sufficient reflection inhibition for many samples, which in turn reduces both length compression and accuracy.

Effect of the sample range for determining dynamic bounds

In the calculation of dynamic upper and lower bounds for RIRM, we count only the number of `</verify>` tags from positive samples. Table 3 examines the effect on accuracy and length compression when the bounds are instead computed using `</verify>` counts from all samples—both positive and negative. Experiments show that including negative samples makes the reflection-inhibition penalty overly lenient for samples with lower pass rates. This leads to insufficient compression of output length, inflates the final reward, and consequently hinders accuracy improvement.

4.4 Analysis of reflection inhibition impact

Figure 4(a) illustrates that after RIRM training, the DeepSeek-R1-Distill-Qwen-1.5B model learns to halt reasoning naturally once the correct answer is reached (marked in blue), thereby sharply reducing subsequent reflection steps. Additional examples are provided in the Appendix. To quantify the improvement in reasoning efficiency brought by the RIRM, we evaluate the RIRM trained DeepSeek-R1-Distill-Qwen-1.5B model on the AIME 2024 and MATH-500 benchmarks. The results show a pronounced compression in output length compared to the baseline. As seen in Figure 4(b), the fraction of tokens generated after the first correct

Table 3: Ablation study on different configurations of RIRM. "Average" reports the average performance across all benchmarks. The best results are highlighted in bold.

Methods	AIME 2024			MATH-500			Average		
	Acc↑	Tok↓	AES↑	Acc↑	Tok↓	AES↑	Acc↑	Tok↓	AES↑
Base	27.45	17514	-	82.89	5227	-	55.17	11371	-
RIRM-DAPO	33.18	7245	1.21	86.33	1877	0.77	59.76	4561	0.99
w/ fixed bounds	30.83	7943	0.92	85.43	1964	0.72	58.13	4954	0.82
w/ dynamic bounds	27.76	6588	0.66	85.51	1546	0.80	56.64	4067	0.73
w/o min. upper bound	32.34	9300	1.00	86.35	2094	0.72	59.35	5697	0.86
w/ 40% as threshold	31.77	7924	1.02	86.03	2037	0.72	58.90	4981	0.87
w/ 60% as threshold	30.36	8442	0.84	86.79	1998	0.76	58.58	5220	0.80
w/ bounds from all samples	28.39	8361	0.63	85.97	1950	0.74	57.18	5156	0.68

Question:
The sum of the first N positive odd numbers is 121. What is the value of N ?

Reasoning:
<think>

Okay, so I need to find the value of N such that the sum of the first N positive odd numbers is 121. Hmm, let me think about this. I remember something about the sum of the first N odd numbers. Isn't there a formula for that? Wait, yes! The sum of the first N odd numbers is N squared. So, the formula is: $\text{Sum} = 1 + 3 + 5 + \dots + (2N - 1) = N^2$. So, if the sum is 121, then $N^2 = 121$. To find N , I just need to take the square root of 121. $\sqrt{121} = 11$. So, N must be 11.

****Final Answer****
The value of N is $\boxed{11}$.
</think>

Summary:
The sum of the first (N) positive odd numbers is given by the formula (N^2) . We need to find (N) such that this sum is 121. Using the formula: $\text{Sum} = N^2$. Given that the sum is 121, we have: $N^2 = 121$. Taking the square root of both sides, we get: $N = \sqrt{121} = 11$. Thus, the value of (N) is $\boxed{11}$.

(a)

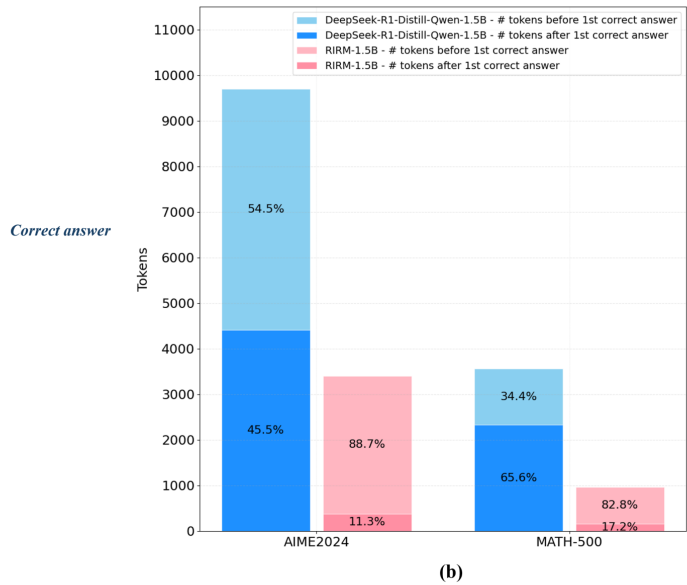


Figure 4: (a) A illustration of DeepSeek-R1-Distill-Qwen-1.5B model’s reasoning process. After training with RIRM, the model naturally stops reasoning upon reaching the correct answer. (b) Breakdown of average token consumption for DeepSeek-R1-Distill-Qwen-1.5B before and after training with RIRM on the AIME 2024 and MATH-500 benchmarks.

answer—which corresponds to reflection steps—is markedly reduced. Specifically, on AIME 2024 this portion drops from 45.5% to 11.3%, and on MATH-500 from 65.6% to 17.2%. Together, these findings indicate that the RIRM effectively suppresses redundant reflections, producing more concise and computationally efficient reasoning trajectories without compromising final accuracy.

5 Conclusion

In this work, we introduce the Reflection Inhibition Reward Mechanism (RIRM) to address the overthinking problem in large reasoning models. By precisely identifying the initial correct answer and subsequent redundant reasoning steps, RIRM evaluates reasoning trajectories against three well-defined criteria and utilizes a unified reward signal to guide policy optimization during reinforcement

learning. Extensive experiments show that RIRM significantly reduces token consumption without sacrificing accuracy across multiple mathematical and scientific reasoning benchmarks. The approach yields marked gains in efficiency while maintaining or even improving predictive performance, thereby establishing a new paradigm to balance computational cost and reasoning quality in large reasoning models.

Limitations

While our proposed Reflection Inhibition Reward Mechanism (RIRM) shows promise in mitigating overthinking, several important directions remain for future work. First, to improve the generalizability and robustness of the reward signal, we intend to explore self-supervised alternatives that reduce reliance on external models for annotation.

572	Second, although RIRM performs well on mathematical and scientific reasoning benchmarks, its efficacy should be further validated in more open-ended, multimodal, or subjectively evaluated domains, where reasoning patterns may differ significantly. Future efforts will therefore focus on developing lightweight training paradigms and conducting broader domain validation to enhance the practicality and applicability of the RIRM framework.	
573		
574		
575		
576		
577		
578		
579		
580		
581		
582		
583	References	
584	Daman Arora and Andrea Zanette. 2025. Training language models to reason efficiently. <i>arXiv preprint arXiv:2502.04463</i> .	
585		
586		
587	Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, and 1 others. 2024. Do not think that much for $2+3=?$ on the overthinking of o1-like llms. <i>arXiv preprint arXiv:2412.21187</i> .	
588		
589		
590		
591		
592		
593	Zhengxiang Cheng, Dongping Chen, Mingyang Fu, and Tianyi Zhou. 2025. Optimizing length compression in large reasoning models. <i>arXiv preprint arXiv:2506.14755</i> .	
594		
595		
596		
597	NVIDIA Corporation. 2025. Nemotron-science-v1. https://huggingface.co/datasets/nvidia/Nemotron-Science-v1 .	
598		
599		
600	Chongyu Fan, Yihua Zhang, Jinghan Jia, Alfred Hero, and Sijia Liu. 2025. Cyclicreflex: Improving large reasoning models via cyclical reflection token scheduling. <i>arXiv preprint arXiv:2506.11077</i> .	
601		
602		
603		
604	Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. <i>arXiv preprint arXiv:2501.12948</i> .	
605		
606		
607		
608		
609		
610	Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware llm reasoning. In <i>Findings of the Association for Computational Linguistics: ACL 2025</i> , pages 24842–24855.	
611		
612		
613		
614		
615	Xin Huang, Tarun Kumar Vangani, Zhengyuan Liu, Bawei Zou, and Ai Ti Aw. 2025. Adacot: Rethinking cross-lingual factual reasoning through adaptive chain-of-thought. <i>arXiv preprint arXiv:2501.16154</i> .	
616		
617		
618		
619	Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for "mind" exploration of large scale language model society. <i>Preprint</i> , arXiv:2303.17760.	
620		
621		
622		
623		
	Yongjiang Liu, Haoxi Li, Xiaosong Ma, Jie Zhang, and Song Guo. 2025. Think how to think: Mitigating overthinking with autonomous difficulty cognition in large reasoning models. <i>arXiv preprint arXiv:2507.02663</i> .	624 625 626 627 628
	Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. 2025. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. <i>arXiv preprint arXiv:2501.12570</i> .	629 630 631 632 633
	Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025. Cot-valve: Length-compressible chain-of-thought tuning. <i>arXiv preprint arXiv:2502.09601</i> .	634 635 636 637
	Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. 2025. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. <i>arXiv preprint arXiv:2504.16891</i> .	638 639 640 641 642 643
	OpenAI. 2024. Openai o1: Reasoning-enhanced model. https://openai.com/o1/ .	644 645
	Xiaoye Qu, Yafu Li, Zhaochen Su, Weigao Sun, Jianhao Yan, Dongrui Liu, Ganqu Cui, Daizong Liu, Shuxian Liang, Junxian He, Peng Li, Wei Wei, Jing Shao, Chaochao Lu, Yue Zhang, Xian-Sheng Hua, Bowen Zhou, and Yu Cheng. 2025. A survey of efficient reasoning for large reasoning models: Language, multimodality, and beyond. <i>CoRR</i> , abs/2503.21614.	646 647 648 649 650 651 652
	Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. 2025. Dast: Difficulty-adaptive slow-thinking for large reasoning models. <i>arXiv preprint arXiv:2503.04472</i> .	653 654 655 656 657
	Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient RLHF framework. In <i>EuroSys</i> , pages 1279–1297.	658 659 660 661 662
	Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. 2025. Stop overthinking: A survey on efficient reasoning for large language models. <i>arXiv preprint arXiv:2503.16419</i> .	663 664 665 666 667 668
	Kimi Team, Angang Du, Bofei Gao, Bawei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. <i>arXiv preprint arXiv:2501.12599</i> .	669 670 671 672 673
	Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.	674 675
	Songjun Tu, Jiahao Lin, Qichao Zhang, Xiangyu Tian, Linjing Li, Xiangyuan Lan, and Dongbin Zhao. 2025. Learning when to think: Shaping adaptive reasoning	676 677 678

679	in r1-style models via multi-stage rl. <i>arXiv preprint arXiv:2505.10832</i> .	Haoran Zhao, Yuchen Yan, Yongliang Shen, Haolei Xu, Wenqi Zhang, Kaitao Song, Jian Shao, Weiming Lu, Jun Xiao, and Yueting Zhuang. 2025. Let llms break free from overthinking via self-braking tuning. <i>arXiv preprint arXiv:2505.14604</i> .	732
680			733
681	Zihao Wei, Liang Pang, Jiahao Liu, Jingcheng Deng, Shicheng Xu, Zenghao Duan, Jingang Wang, Fei Sun, Xunliang Cai, Huawei Shen, and Cheng Xueqi. 2025. Stop spinning wheels: Mitigating llm overthinking via mining patterns for early reasoning exit. <i>arXiv preprint arXiv:2508.17627</i> .		734
682			735
683			736
684			
685			
686			
687	Shawn Wu, Sean Wang, Louie Li, Darcy Chen, Allen Wang, Jiangang Luo, Xudong Zhao, Joseph Shen, Gawain Ma, Jasper Jia, Marcus Mao, Claire Wang, Hunter He, Carol Wang, Zera Zhang, Jason Wang, Chonly Shen, Leo Zhang, Logan Chen, and 6 others. 2026. Yuan3.0 flash: An open multimodal large language model for enterprise applications. <i>Preprint, arXiv:2601.01718</i> .		
688			
689			
690			
691			
692			
693			
694			
695	Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025. Chain of draft: Thinking faster by writing less. <i>arXiv preprint arXiv:2502.18600</i> .		
696			
697			
698	An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. <i>arXiv preprint arXiv:2505.09388</i> .		
699			
700			
701			
702			
703	Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. 2025b. Dynamic early exit in reasoning models. <i>arXiv preprint arXiv:2504.15895</i> .		
704			
705			
706			
707	Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. <i>arXiv preprint arXiv:2502.03387</i> .		
708			
709			
710	Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. Demystifying long chain-of-thought reasoning in llms. <i>arXiv preprint arXiv:2502.03373</i> .		
711			
712			
713			
714	Jingyang Yi, Jiazheng Wang, and Sida Li. 2025. Shorterbetter: Guiding reasoning models to find optimal inference length for efficient reasoning. <i>arXiv preprint arXiv:2504.21370</i> .		
715			
716			
717			
718	Bin Yu, Hang Yuan, Haotian Li, Xueyin Xu, Yuliang Wei, Bailing Wang, Weizhen Qi, and Kai Chen. 2025a. Long-short chain-of-thought mixture supervised fine-tuning eliciting efficient reasoning in large language models. <i>arXiv preprint arXiv:2505.03469</i> .		
719			
720			
721			
722			
723	Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gao-hong Liu, Lingjun Liu, and 1 others. 2025b. Dapo: An open-source llm reinforcement learning system at scale. <i>arXiv preprint arXiv:2503.14476</i> .		
724			
725			
726			
727			
728	Chuhuai Yue, Chengqi Dong, Yinan Gao, Hang He, Jiajun Chai, Guojun Yin, and Wei Lin. 2025. Promoting efficient reasoning with verifiable stepwise reward. <i>arXiv preprint arXiv:2508.10293</i> .		
729			
730			
731			

A Pseudocode of the Algorithm

Algorithm 1 Calculation of R_{ver}

Input:

acc_score: accuracy of the response
 marked_txt: annotated reasoning process
 V_{min} : lower bound of </verify> counts
 V_{max} : upper bound of </verify> counts

Output:

Total verification reward R_{ver}

$v \leftarrow \text{marked_txt.count}('</verify>')$

if $v = 0.0$ **then**

$R_{ver} \leftarrow \text{acc_score}$

else if $V_{max} \leq 2$ **then**

$R_{ver} \leftarrow 1.0$

else

if $0 < v \leq V_{min}$ **then**

$R_{ver} \leftarrow 1.0$

else if $V_{min} < v \leq V_{max}$ **then**

$R_{ver} \leftarrow 1 - \frac{v - V_{min}}{V_{max} - V_{min}}$

else

$R_{ver} \leftarrow 0.0$

return R_{ver}

B Accuracy-Efficiency Score (AES)

The Accuracy–Efficiency Score (AES) (Luo et al., 2025) is a composite metric that assesses whether a model can reduce its output length without compromising accuracy. It is defined as:

$$\text{AES} = \begin{cases} \phi \cdot \Delta\text{Length} + \eta \cdot |\Delta\text{Acc}|, & \text{if } \Delta\text{Acc} \geq 0 \\ \phi \cdot \Delta\text{Length} - \theta \cdot |\Delta\text{Acc}|, & \text{if } \Delta\text{Acc} < 0 \end{cases} \quad (1)$$

where

$$\Delta\text{Length} = \frac{\text{Length}_{base} - \text{Length}_{model}}{\text{Length}_{base}}$$

and

$$\Delta\text{Acc} = \frac{\text{Acc}_{model} - \text{Acc}_{base}}{\text{Acc}_{base}}$$

denote the relative changes in output token length and accuracy, respectively, compared to a base model. A positive AES indicates that the model produces shorter outputs while preserving or improving accuracy, whereas a negative score reflects a penalty for accuracy degradation. The hyperparameter ϕ weights length reduction, η rewards accuracy gains, and θ penalizes accuracy losses. In

our experiments, following (Luo et al., 2025) and (Yi et al., 2025), we adopt the following hyperparameter values:

$$\phi = 1, \quad \eta = 3, \quad \theta = 5$$

Setting $\theta > \eta$, imposes a stronger penalty for accuracy loss than the reward for accuracy gain, aligning with the practical preference for avoiding performance deterioration.

C Prompts to Locate Correct Answer

This section presents the prompt templates used to locate correct answers within reasoning trajectories. The template in Figure 5 is employed for mathematical tasks, while those in Figure 6 and Figure 7 are used for scientific tasks. The second prompt template in Figure 7 is more stringent than the one in Figure 6, making it particularly suitable for identifying correct answers in reasoning trajectories where the final answer is incorrect.

D Examples of Reasoning Trajectories

This section showcases reasoning trajectories generated by the Deepseek-R1-Distill-Qwen-1.5B and Yuan3.0 Flash-SFT models after RIRM training. The examples, Figure 8 to Figure 11 are selected from the AIME 2024 and GPQA-Diamond evaluation datasets, illustrate that the models maintain fluent and coherent reasoning paths. Notably, the occurrence of reflection steps remains infrequent, and once a correct answer is reached, the reasoning process tends to terminate naturally.

Prompt to locate correct answer in mathematical task

```
""""[Original Question]:
...
{Question_text}
...

** [Standard Answer to the Original Question]: **
...
{Solution_text}
...

**Traverse the text provided in Input, and extract all parts of the text that provide the exact complete [Standard Answer] to the given
[Question] as above.**
**Notes:**
- **Do not modify the original text.**
- **Must scan from the beginning of the text, referencing the standard answer of the original question, to find all positions providing
identical answers. Ensure no omissions, ensure no omissions, ensure no omissions.**
- **Must ignore differences in formats such as LaTeX, and judge based on the consistency of mathematical content. For example,
coordinate (7,10) is equivalent to x=7,y=10, sqrt(5) is equivalent to  $\sqrt{5}$ , and pay attention to differences in interval openness/closedness,
etc.**
- If the answer of the original question contains multiple values or formulas (i.e., more than one numerical value/content), the original
text may present the complete answer across multiple consecutive sentences. In such cases, all these consecutive sentences must be
completely extracted.
- **Whenever the same [Standard Answer to the Question] appears (ignoring non-mathematical differences such as LaTeX formatting),
check whether the content at that location (could be a single sentence or multiple consecutive sentences) completely and accurately
presents the answer to the question. If yes, return the corresponding content based on the following conditions: if the [Standard Answer
to the Question] is a single value/formula, return the sentence containing the answer; if the [Standard Answer to the Question] includes
multiple values/formulas, return all consecutive sentences in the original text that collectively form the complete answer (note: the
extracted content must include all answers provided, not just partial answers).**
- **If the [Standard Answer to the Question] contains multiple answers (i.e., more than one value/formula/point/interval, etc.), the
content at the identified position must contain all answers simultaneously. Content that only includes part of the answers does not meet
the criteria.**

**Output Format:**
- If qualifying content exists, list all qualifying content in order of appearance as follows:
...
1. ANSWER location 1: [The first block of content that fully presents the answer (extracted single sentence or consecutive sentences from
the original text), **do not modify the language, structure, wording, etc. of the original text.**]<get_answer>
...
n. ANSWER location n: [The n-th block of content that fully presents the answer (extracted single sentence or consecutive sentences
from the original text), **do not modify the language, structure, wording, etc. of the original text.**]<get_answer>
...
- If no qualifying content exists, output:
...
[InputText does not contain content providing the final answer to the question]
...

**Input:**
...
{InputText}
...

**Re-emphasizing: Whenever content identical to the [final answer] appears (ignoring LaTeX formatting and other non-mathematical
meaning differences), check whether the content at that position (which can be a single sentence or multiple consecutive sentences)
completely summarizes all answers to the question.**

**Output:**
""""
```

Figure 5: Prompt template used to locate correct answers in mathematical reasoning trajectories.

Prompt to locate correct answer in scientific task # 1

```
""""
**Traverse the text in [InputText] and extract all text fragments that provide the exact complete [Standard Answer] to the [Question]. The
extracted text must meet the following requirements:**
- **Do not modify the original text**: When extracting, must use the exact words and sentences from the original text. Do not
rewrite, summarize, paraphrase, or alter any wording, structure, or format (including LaTeX formatting).
- Must fully scan the entire [InputText]: Refer to the [Standard Answer] of the original question, extract all content that provides the
standard answer to the question, and ensure no fragments containing the answer are omitted.
- If the [Question] is a multiple-choice question and the [Standard Answer] is the correct option: Identify the content corresponding
to the correct option in the question. As long as the complete content of the option is clearly mentioned in the input text (no need to
associate it with the option label), it is considered to provide the correct answer and needs to be extracted. Note: Merely listing the
[Standard Answer] or the content corresponding to the correct option without providing positive descriptions such as correctness,
affirmation, or potential correctness, or without deriving it through reasoning, analysis, or calculation, does not meet the requirements.
- Must ignore differences in formats such as LaTeX: Use the consistency of mathematical content as the judgment criterion. For
example, the coordinate (7,10) is equivalent to  $x=7,y=10$ ,  $\sqrt{5}$  is equivalent to  $\sqrt{5}$ , and attention should be paid to differences in
interval openness/closedness (e.g., the interval (0, 1) is different from [0, 1]), etc.
- Complete matching: The found text fragment (which may be a single sentence or consecutive sentences) must completely
contain the entire standard answer. If the answer includes multiple parts (such as multiple options, multiple values, multiple formulas,
multiple points, multiple intervals, etc.), the extracted fragment must simultaneously contain all these parts. Fragments that only
include part of the answer do not meet the requirements.
- As long as content identical to the [Standard Answer] or the content corresponding to the option appears (ignoring non-
mathematical differences such as LaTeX formatting): Check whether the content at that position (which can be a single sentence or
consecutive sentences) accurately and completely presents the answer to the question. If yes, return the corresponding fragment
according to the following conditions:
- If the [Standard Answer] is a single element such as a single option, value, or formula, extract the sentence(s) that provide the answer
or the content corresponding to the option.
- If the [Standard Answer] contains multiple parallel elements such as multiple options, values, formulas, points, or intervals, extract
one or more consecutive sentences from the original text that together exactly and completely contain all answer elements or all content
corresponding to the options. (Note: The found content must include all provided answers; those containing only part of the answers do
not meet the conditions.)

Output Format:
- If there is content that meets the requirements, list them in the order of appearance as follows:
...
1. ANSWER location 1: [The first content that fully presents the answer (a single sentence or consecutive sentences extracted from the
original text). When extracting, must use the exact words and sentences from the original text. Do not rewrite, summarize, paraphrase, or
alter any wording, structure, or format.]<get_answer>
...
n. ANSWER location n: [The nth content that fully presents the answer (a single sentence or consecutive sentences extracted from the
original text). When extracting, must use the exact words and sentences from the original text. Do not rewrite, summarize, paraphrase, or
alter any wording, structure, or format.]<get_answer>
...
- If there is no content that meets the requirements, output:
...
[The input text does not contain content that provides the final answer to the question]
...

[Question]:**
...
{Question_text}
...

[Standard Answer]:**
...
{Solution_text}
...

[InputText]:**
...
{InputText}
...

Note: Do not expand the input text; only search for all fragments that meet the requirements within [InputText].**

Output:**
""""
```

Figure 6: Prompt template used to locate correct answers in scientific reasoning trajectories.

Prompt to locate correct answer in scientific task # 2

```
****
**Traverse the text in [InputText] to find all complete and accurate text fragments that provide the exact complete [Standard Answer] to the [Question].**

**Key Rules and Requirements:**
- **Do not modify the original text:** When extracting, must use the exact words and sentences from the original text. Do not rewrite, summarize, paraphrase, or alter any wording, structure, or format (including LaTeX format).
- **Must fully scan the entire [InputText]:** Refer to the standard answer of the original question, extract all content that provides the standard answer to the question, and ensure no fragments containing the answer are omitted. Ensure no omissions. Ensure no omissions.
- **If the [Question] is a multiple-choice question and [Standard Answer] is the correct option:** Identify the content corresponding to the correct option in the question. As long as the complete content of the option is clearly mentioned in the input text (no need to associate it with the option label), it is considered to provide the correct answer. **Note:** Merely listing the [Standard Answer] or the content corresponding to the correct option without providing descriptions such as correctness, affirmation, or potential correctness, or without deriving it through reasoning, analysis, or calculation, does not meet the requirements.
- **Must ignore differences in formats such as LaTeX:** Judge based on the consistency of mathematical content. For example, the coordinate (7,10) is equivalent to  $x=7,y=10$ ,  $\sqrt{5}$  is equivalent to  $\sqrt{5}$ , and note differences in interval openness and closedness (e.g., the interval (0, 1) is different from [0, 1]), etc.
- **Complete matching:** The found text fragment (which may be a single sentence or consecutive sentences) must completely contain the entire standard answer. If the answer includes multiple parts (such as multiple options, multiple values, multiple formulas, multiple points, multiple intervals, etc.), the extracted fragment must simultaneously contain all these parts. Fragments that only include part of the answer do not meet the requirements.
- **As long as content identical to the [Standard Answer] or the content corresponding to the option appears (ignoring non-mathematical differences such as LaTeX formatting):** Check whether the content at that position (which can be a single sentence or consecutive sentences) accurately and completely presents the answer to the question. If yes, return the corresponding fragment according to the following conditions:
  - If the [Standard Answer] is a single element such as a single option, value, or formula, extract the sentence(s) that provide the answer or the content corresponding to the option.
  - If the [Standard Answer] contains multiple parallel elements such as multiple options, values, formulas, points, or intervals, extract one or more consecutive sentences from the original text that together exactly and completely contain all answer elements or all content corresponding to the options. (Note: The found content must include all provided answers; those containing only part of the answers do not meet the conditions.)
  - **Note: Content that fully presents the answer or the content of the correct option but does not explicitly confirm the correctness of the answer (e.g., using ambiguous words such as "may," "not sure if correct," "perhaps," etc.) does not meet the requirements.**

**Output Format:**
- If there is content that meets the requirements, list them in the order of appearance as follows:
...
  1. ANSWER location 1: [The first content that fully presents the answer (a single sentence or consecutive sentences extracted from the original text). When extracting, must use the exact words and sentences from the original text. Do not rewrite, summarize, paraphrase, or alter any wording, structure, or format.]<get_answer>
  ...
  n. ANSWER location n: [The nth content that fully presents the answer (a single sentence or consecutive sentences extracted from the original text). When extracting, must use the exact words and sentences from the original text. Do not rewrite, summarize, paraphrase, or alter any wording, structure, or format.]<get_answer>
...
- If there is no content that meets the requirements, output:
...
[The input text does not contain content that provides the final answer to the question]
...

**[Question]:**
...
{Question_text}
...

**[Standard Answer]:**
...
{Solution_text}
...

**[InputText]:**
...
{InputText}
...

**Re-emphasize:**
- **For multiple-choice questions: If [InputText] contains the content corresponding to the standard correct option, it is also considered to provide the correct answer to the question. Note: Content that fully presents the answer or the content of the correct option but does not explicitly confirm that the answer is correct, meets the conditions, or is uncertain about its correctness (e.g., containing ambiguous words such as "may," "perhaps," etc.) does not meet the requirements.**
- **Note: Do not expand the input text; only search for all fragments that meet the requirements in [InputText].**

**Output:**
****
```

Figure 7: Prompt template used to locate correct answers in scientific reasoning trajectories. No.2 prompt template is more stringent than No.1 template and is suitable for identifying correct answers in reasoning trajectories where the final answer is incorrect.

Example of DS-1.5B # 1

There exist real numbers x and y , both greater than 1, such that $\log_x \left(\log_y (x^{4y}) \right) = 10$. Find xy .

<think>
 Okay, so I have this problem here: There exist real numbers x and y , both greater than 1, such that \log_b base x of (y^{4x}) equals \log_b base y of (x^{4y}) equals 10. I need to find the product xy .
 Hmm, let me write down what I know. The equations given are:
 $\log_x (y^{4x}) = 10$ and $\log_y (x^{4y}) = 10$
 I remember that $\log_b (a^c) = c \cdot \log_b (a)$, so maybe I can use that property to simplify these equations.
 Starting with the first equation: $\log_x (y^{4x}) = x \cdot \log_x (y) = 10$
 Similarly, the second equation: $\log_y (x^{4y}) = 4y \cdot \log_y (x) = 10$
 So now I have:
 $1) x \cdot \log_x (y) = 10$ and $2) 4y \cdot \log_y (x) = 10$
 Hmm, maybe I can express $\log_x (y)$ in terms of $\log_y (x)$. Since $\log_x (y)$ is the reciprocal of $\log_y (x)$, right? Because $\log_x (y) = 1 / \log_y (x)$. Let me denote $\log_y (x)$ as k for simplicity. Then $\log_x (y)$ would be $1/k$.
 So substituting into equation 1: $x \cdot (1/k) = 10$ and equation 2: $4y \cdot k = 10$
 So now I have:
 From equation 1: $x = 10k$ and from equation 2: $4y \cdot k = 10 \Rightarrow y = 10 / (4k) = 5 / (2k)$
 So now I have $x = 10k$ and $y = 5 / (2k)$
 Now, I need to find xy . Let's compute that:
 $xy = (10k) \cdot (5 / (2k)) = (10 \cdot 5) / (2 \cdot k / k) = 50 / 2 = 25$
 So the product xy is 25.
 Final Answer
 \boxed{25}

</think>
 Given the equations:
 $\log_x (y^{4x}) = 10$ and $\log_y (x^{4y}) = 10$
 We start by using the logarithm property $\log_b (a^c) = c \cdot \log_b (a)$. This transforms the equations into:
 $1. x \cdot \log_x (y) = 10$ and $2. 4y \cdot \log_y (x) = 10$
 Let $k = \log_y (x)$. Then, $\log_x (y) = 1/k$. Substituting these into the equations, we get:
 $1. x \cdot (1/k) = 10 \Rightarrow x = 10k$ and $2. 4y \cdot k = 10 \Rightarrow y = 5 / (2k)$
 Now, we find the product xy :
 $xy = (10k) \cdot (5 / (2k)) = 50 / 2 = 25$
 Thus, the product xy is \boxed{25}

Figure 8: This example presents a reasoning trajectory from the Deepseek-R1-Distill-1.5B model trained with RIRM, based on a question from AIME 2024.

Example of DS-1.5B # 2

"Oh, I know you," the SRP says to the nascent chain as they meet in the cytosol. "Let me grab you. I'll take you to the ER; you really need some sugar."
 "It seems somewhat rough. I guess this is goodbye; I won't see you anymore", the protein replies. Where is the protein headed?
 A. to the proteasome.
 B. to the extracellular space.
 C. to apoptosis.
 D. to the nucleus.

<think>
 Okay, so I've got this question here, and I'm trying to figure out where the protein is going. Let me read it again.
 The question is about the SRP, which is the SRP-7 protein complex. It says, "Oh, I know you," to the nascent chain as they meet in the cytosol. "Let me grab you. I'll take you to the ER; you really need some sugar." Then the protein replies, "It seems somewhat rough. I guess this is goodbye; I won't see you anymore." The question is, where is the protein headed?
 The options are:
 A. to the proteasome.
 B. to the extracellular space.
 C. to apoptosis.
 D. to the nucleus.
 Hmm, okay. I remember that the SRP-7 complex is involved in protein synthesis, specifically in the translation of mRNA into proteins. It helps bring the mRNA to the ER for processing.
 So, the nascent chain is in the cytosol, which is the organelle where ribosomes are, right? So when the SRP-7 complex arrives, it grabs the mRNA and brings it to the ER. In the ER, the mRNA is processed, including the translation machinery, and then exported to the Golgi apparatus, which is in the Golgi bag.
 So, the protein itself, being the part of the ER, would be sent where? Well, the ER is part of the cell membrane, which is the extracellular space. So the protein would go to the extracellular space. That's option B.
 Let me think about the other options. Option A, the proteasome. The proteasome is involved in breaking down proteins, not in the synthesis or transport of proteins. So that's probably not it. Option C, apoptosis. The SRP-7 isn't involved in cell death or apoptosis, so that's out. Option D, the nucleus. The SRP-7 is in the cytoplasm, so it wouldn't go to the nucleus, where proteins are transcribed.
 So, putting it all together, the protein is heading to the extracellular space. So the answer should be B.
 </think>
 The correct answer is B. to the extracellular space.
 \boxed{B}

Figure 9: This example presents a reasoning trajectory from the Deepseek-R1-Distill-1.5B model trained with RIRM, based on a question from GPQA-Diamond.

Example of Yuan3.0 Flash # 1

Jen enters a lottery by picking 4 distinct numbers from $S = \{1, 2, 3, \dots, 9, 10\}$. 4 numbers are randomly chosen from S . She wins a prize if at least two of her numbers were 2 of the randomly chosen numbers, and wins the grand prize if all four of her numbers were the randomly chosen numbers. The probability of her winning the grand prize given that she won a prize is $\frac{m}{n}$ where m and n are relatively prime positive integers. Find $m+n$.

<think>
 Okay, so I have this probability problem about Jen entering a lottery. Let me try to understand it step by step.
 Jen picks 4 distinct numbers from the set $S = \{1, 2, 3, \dots, 9, 10\}$. Then, 4 numbers are randomly chosen from S . She wins a prize if at least two of her numbers are among the randomly chosen numbers. She wins the grand prize if all four of her numbers are chosen. We need to find the probability that she wins the grand prize given that she won a prize. The answer should be in the form m/n where m and n are coprime, and then find $m+n$.
 Hmm, so this is a conditional probability problem. The formula for conditional probability is $P(A|B) = P(A \cap B) / P(B)$. In this case, event A is winning the grand prize, and event B is winning a prize. Since winning the grand prize is a subset of winning a prize, $A \cap B$ is just A . So, $P(A|B) = P(A) / P(B)$.
 First, I need to find $P(A)$, the probability of winning the grand prize. That's the probability that all four of her numbers are chosen. The total number of ways to choose 4 numbers from 10 is $C(10, 4)$. The number of favorable outcomes is 1, since there's only one way to choose all four numbers. So, $P(A) = 1 / C(10, 4)$.
 Calculating $C(10, 4) = 10! / (4! \cdot (10-4)!) = (10 \cdot 9 \cdot 8 \cdot 7) / (4 \cdot 3 \cdot 2 \cdot 1) = 210$. So, $P(A) = 1/210$.
 Next, I need to find $P(B)$, the probability of winning a prize, which is winning at least two of her numbers. It might be easier to calculate the complementary probability (i.e., the probability of winning fewer than two numbers) and subtract it from 1.
 Winning fewer than two numbers means winning 0 or 1 number.
 Calculating $P(0 \text{ numbers})$: The number of ways to choose 4 numbers with none of Jen's. Since she has 4 numbers, the remaining 6 numbers are chosen. So, $C(6, 4) = 15$. So, $P(0) = 15 / 210$.
 Calculating $P(1 \text{ number})$: She chooses 1 number from her 4, and 3 from the remaining 6. So, $C(4, 1) \cdot C(6, 3) = 4 \cdot 20 = 80$. So, $P(1) = 80 / 210$.
 Therefore, $P(B) = 1 - P(0) - P(1) = 1 - (15 + 80) / 210 = 1 - 95 / 210 = (210 - 95) / 210 = 115 / 210$.
 Simplify $115 / 210$: Divide numerator and denominator by 5: $23 / 42$. So, $P(B) = 23/42$.
 Now, going back to the conditional probability: $P(A|B) = P(A) / P(B) = (1/210) / (23/42)$.
 Dividing fractions: $(1/210) \cdot (42/23) = (42) / (210 \cdot 23)$. Simplify $42/210$: divide numerator and denominator by 42: $1/5$. So, it becomes $(1/5) / 23 = 1 / (5 \cdot 23) = 1/115$.
 So, the probability is $1/115$. Therefore, $m = 1$, $n = 115$, and $m+n = 1 + 115 = 116$.
 </think>
 The probability of winning the grand prize given that she won a prize is \boxed{116}.

Figure 10: This example presents a reasoning trajectory from the Yuan3.0 Flash-SFT model trained with RIRM, based on a question from AIME 2024.

Example of Yuan3.0 Flash # 2

Consider a system with Hamiltonian operator $H = \epsilon \sum_{i=1}^n \sigma_i$. Here, σ_i is an arbitrary unit vector, ϵ is a constant of dimension energy, and components of σ_i are the Pauli spin matrices. What are the eigenvalues of the Hamiltonian operator?
 A. $+1, -1$.
 B. $\epsilon, -\epsilon$.
 C. $\sqrt{2}\epsilon, -\sqrt{2}\epsilon$.
 D. $1/3\epsilon, 2/3\epsilon$.

<think>
 Okay, so I need to find the eigenvalues of the Hamiltonian operator given by $H = \epsilon \sum \sigma_i$, where σ_i is the Pauli matrices and n is a unit vector. Hmm, let me think. I remember that the Pauli matrices are $\sigma_x, \sigma_y, \sigma_z$, and each has eigenvalues $+1$ and -1 . But when you take a linear combination like $\sum \sigma_i$, which is $\sigma_x + \sigma_y + \sigma_z$, the eigenvalues of that combination are still ± 1 , right? Because the Pauli matrices are traceless and their eigenvalues are fixed. So multiplying by a scalar ϵ , which has dimension energy, the eigenvalues should just scale by ϵ . So the eigenvalues of H should be $\pm \epsilon$. Let me check the options. Option B is $+\epsilon$ and $-\epsilon$, which matches. The other options don't make sense. Option A is $+1$ and -1 , which would be the case if $\epsilon = 1$, but since ϵ is a general energy constant, the eigenvalues should be scaled by ϵ . So the correct answer must be B.
 </think>
 The Hamiltonian $H = \epsilon \sum \sigma_i$ involves the Pauli matrices σ_i , which have eigenvalues ± 1 and ± 1 when acting on any unit vector \vec{n} . Multiplying by the scalar ϵ scales these eigenvalues by ϵ , resulting in eigenvalues $\pm \epsilon$ and $\pm \epsilon$.
 Answer \boxed{B}

Figure 11: This example presents a reasoning trajectory from the Yuan3.0 Flash-SFT model trained with RIRM, based on a question from GPQA-Diamond.