RETRAIL: HESITATION-GUIDED BACKTRACKING FOR TEST-TIME REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large reasoning models (LRMs) often revise earlier steps through reflection, but insufficient reflection leaves mistakes uncorrected, whereas excessive reflection leads to repetitive reasoning. Existing approaches that regulate reflection using fixed schedules cannot adequately address this difficulty because they do not adapt to whether the model needs to reconsider its reasoning or move forward. To address this limitation, we propose ReTrail, a training-free decoding method that uses the model's own reflection-token probabilities to guide when and how it revises its reasoning. To determine when to intervene, ReTrail monitors the hesitation mass, defined as the total probability the model assigns to reflection tokens at the end of each reasoning step. This probability measures the model's tendency to reflect, and it remains available even when no reflection token is sampled into the output text. When this signal rises sharply above its earlier values, ReTrail selects an earlier step with high hesitation mass as the starting point for revision. It rolls back to that step and regenerates the continuation. During regeneration, ReTrail first raises reflection-token logits to encourage the model to reconsider that step, and then lowers them to prevent repetitive reflection. The method operates on a single generation trajectory without requiring an external verifier model or multi-candidate search. Across four reasoning models and four benchmarks in mathematics and code generation, ReTrail achieves the highest or tied-highest mean accuracy in all sixteen evaluated settings under identical retained-output token limits. It improves accuracy over unmodified decoding by an average of 3.25 and up to 8.08 percentage points, with a 15.2% average increase in total generated tokens, including tokens discarded during rollback. Component ablations show that rollback improves accuracy while reducing generation length compared to applying reflection bias in place. Furthermore, controlled comparisons of revision timing and restart location show higher mean accuracy with hesitation mass than with representative alternatives like entropy. Code is available at `https://anonymous.4open.science/r/retrail-8FC5/`.

1 INTRODUCTION

Large reasoning models (LRMs) such as DeepSeek-R1 answer a hard problem only after generating a long chain of thought (DeepSeek-AI, 2025). Giving the model more compute at test time often raises its accuracy on such problems, an effect known as test-time scaling (Snell et al., 2024; Muennighoff et al., 2025). There are two ways to make the model think more. One is to let a single chain of thought, or trajectory, run longer so that the model works through more steps and revisits more of them (Muennighoff et al., 2025). The other is to sample several trajectories in parallel and choose among their answers by majority vote or by a scorer, as in self-consistency and Best-of-N sampling (Wang et al., 2023; Brown et al., 2024). In the first way, the value of the extra steps depends on how the model uses them. A longer trajectory helps when the model revisits a questionable earlier step and revises it, but it can also carry an early mistake through the remaining reasoning and spend the extra tokens without correcting it (Chen et al., 2026). Such a revisit is often visible in the text through words such as *Wait*, *But*, and *Alternatively*. These words are known as reflection tokens, and the revisiting they mark is known as reflection (Fan et al., 2026). Excessive reflection can lead to repeated reconsideration without progress (over-reflection), whereas insufficient reflection can leave errors uncorrected (under-reflection) (Fan et al., 2026). In either case, an early reasoning error

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.

can influence subsequent steps and remain uncorrected despite later reflection (Chen et al., 2026). However, models can correct reasoning errors more effectively when the error location is provided (Tyen et al., 2024). These findings motivate a decoding method that uses the evolving trajectory to determine when to intervene and from which earlier point to resume generation.

Existing methods regulate reflection using preset rules. They encourage the model to continue reasoning (Muennighoff et al., 2025), discourage early changes of strategy (Wang et al., 2025b), or alternate between promoting and suppressing reflection (Fan et al., 2026). These rules govern how generation continues from the current prefix. Revising an earlier step also requires choosing where to resume. Other approaches select this restart point using a separately trained model to score reasoning steps (Singh et al., 2025) or predict where correction is needed (Chen et al., 2026). This raises a question: *can the reasoning model itself determine both when to intervene and where to resume?*

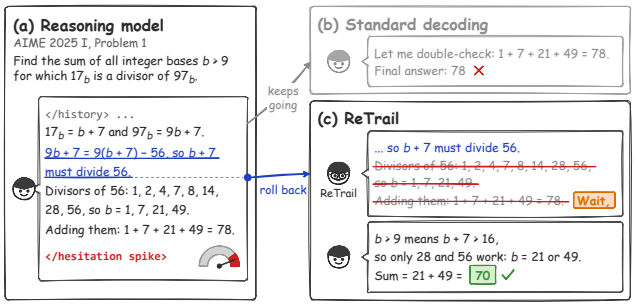

Figure 1: ReTrail on Problem 1 of AIME 2025 I. (a) The hesitation mass, the probability that a model assigns to reflection tokens such as *Wait* at the end of each step, spikes two steps after the step at which the model was least sure (underlined). (b) Standard decoding keeps going and answers 78. (c) ReTrail rolls back to the underlined step, deletes what followed (struck through), briefly raises the probability of reflection tokens, and the model answers 70.

To answer this question, we propose ReTrail, a training-free decoding method that uses the model's own reflection-token probabilities to guide when and where to revise a single reasoning trajectory. We quantify the model's tendency to begin the next step with reflection through the hesitation mass, the total probability assigned to reflection tokens at the end of each reasoning step. Because these probabilities are available even when no reflection token is sampled, ReTrail can use them to guide revision throughout generation. A sharp rise relative to earlier values prompts ReTrail to return to a recent step with high hesitation mass and regenerate from there. This localized revision gives the model an opportunity to reconsider the reasoning that precedes the detected rise (Figure 1).

To encourage a different continuation from the selected step, ReTrail combines rollback with bidirectional hedging, a temporary sequence of reflection promotion and suppression. It first promotes reflection to encourage reconsideration of earlier reasoning and then suppresses reflection to support progress toward an answer. By placing this sequence immediately after rollback, ReTrail concentrates reflection control where a new continuation is formed. The revised trajectory provides the basis for subsequent interventions, allowing the method to adapt its revisions as reasoning develops.

We evaluate ReTrail against unmodified decoding and two methods that regulate reflection (Wang et al., 2025b; Fan et al., 2026) on four reasoning models. The benchmarks cover competition mathematics through AIME 2025, AIME 2026, and HMMT 2026, and code generation through LiveCodeBench. All methods share a retained-output token limit within each model–benchmark setting. Total generation includes tokens deleted by rollback and is reported as generation cost. ReTrail achieves the highest mean accuracy in fourteen settings and ties for the highest in the remaining two. Its gain over unmodified decoding averages 3.25 percentage points and reaches 8.08 points on DeepSeek-V3.2-Speciale–HMMT 2026, where total generation increases by only 4.1%. In the component ablation on GPT-OSS-120B and Qwen-7B, the full method achieves the highest accuracy in all four settings. Adding localized rollback improves accuracy while reducing total generated tokens in each setting (Table 2). At a fixed trigger, the selected restart location also yields higher mean accuracy than entropy-based or distance-matched random choices (Table 4).

Our contributions are:

- We formulate the regulation of reflection as two connected decisions, when to intervene and where to resume, and base both on the hesitation mass of the trajectory itself.

- We propose ReTrail, a training-free decoding method that intervenes at a sharp rise in the hesitation mass and rolls back to the step of highest mass shortly before it. After each rollback it hedges by first raising and then lowering reflection-token logits (Section 4).
- We demonstrate accuracy gains across four reasoning models and four mathematics and coding benchmarks. Component ablations establish the benefits of localized rollback and quantify the contribution of the subsequent probability bias (Section 5).

2 Related Work

2.1 Test-Time Scaling and Adaptive Computation

Test-time scaling allocates additional inference computation across sampled solutions or along individual reasoning trajectories. Self-consistency aggregates sampled answers (Wang et al., 2023), while repeated sampling increases the fraction of problems for which a correct solution is generated (Brown et al., 2024). Along a single trajectory, s1 uses budget forcing to prolong or terminate reasoning (Muennighoff et al., 2025). The effectiveness of these strategies depends on problem difficulty and how computation is allocated (Snell et al., 2024). Adaptive-Consistency adjusts sample counts using answer agreement (Aggarwal et al., 2023), and DEER uses confidence in trial answers to decide when to stop an ongoing trajectory (Yang et al., 2026). ReTrail reallocates computation within a trajectory by selectively replacing a suffix and regenerating from an earlier step.

2.2 Reflection Regulation in Reasoning Models

Reflection regulation addresses redundant reconsideration and premature changes of strategy (Chen et al., 2025; Wang et al., 2025b). TIP penalizes tokens associated with thought switching (Wang et al., 2025b), while NoWait suppresses selected reflection tokens throughout generation (Wang et al., 2025a). CyclicReflex alternates promotion and suppression through a periodic logit schedule (Fan et al., 2026). Adaptive methods condition this control on the model's current state. CGRS uses the entropy of trial answers to set the probability of suppressing reflection (Huang et al., 2026). PathCal aggregates marker probabilities to assess competition between continuing a reasoning path and opening another, then adjusts marker logits to favor continuation (Jiang et al., 2026). ReTrail uses changes in reflection-token probability to initiate a rollback. Its promotion–suppression waveform begins at the selected earlier step and runs once per intervention.

2.3 Self-Correction and Backtracking for Reasoning

Self-correction can use model-generated feedback to iteratively revise an answer (Madaan et al., 2023). Its reliability depends on recognizing when and where correction is needed. Studies report failures without external feedback (Huang et al., 2024) and improved repair when an error location is supplied (Tyen et al., 2024). Backtracking localizes revision by retaining a prefix and replacing its continuation. Process reward models guide backtracking through step-level value estimates (Singh et al., 2025), while TAAR trains a diagnostic model to predict restart locations and intervention strength (Chen et al., 2026). Entropy-gated branching selects uncertain positions for branching and ranks alternatives with a process reward model (Li et al., 2026). Thought-ICS combines structured error localization with self- or oracle verification (Samanta et al., 2026). Entropy-Cut Metropolis–Hastings uses entropy jumps to select suffix-resampling positions and accepts or rejects revisions using model probabilities (Zhou et al., 2026). ReTrail derives intervention timing and the rollback point from reflection-token probabilities, then guides regeneration with a short bidirectional bias.

3 Preliminaries

Reasoning trajectories and step boundaries. Given a problem x, a large reasoning model π generates a trajectory y consisting of a chain of thought followed by an answer. We write $\pi(\cdot \mid x, y)$ for the next-token sampling distribution after any logit bias, temperature scaling, and top-p truncation. Reasoning steps are separated by blank lines outside code blocks (Chen et al., 2026). Their ending positions $b_1, b_2, \ldots$ become boundaries once further text follows them; Appendix A gives the segmentation rule. The prefix through a boundary b is denoted by $y_{\leq b}$.

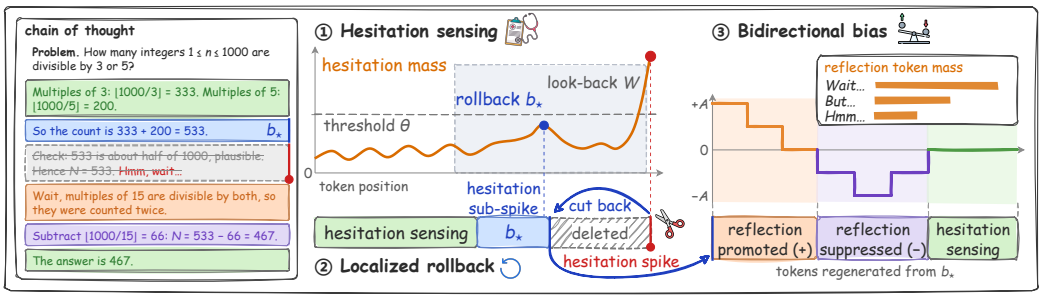

Figure 2: Overview of ReTrail; colors connect the reasoning steps to the three stages. ① Hesitation sensing detects a rise above the trajectory's threshold θ (red). ② Localized rollback selects an earlier boundary $b_\star$ with high mass within the preceding W tokens (blue). ③ Bidirectional hedging guides regeneration with reflection-token bias from $+A$ through $-A$ to zero. Detection resumes after a further cooldown of R unbiased tokens, subject to the intervention limit.

Reflection tokens. Words such as *Wait*, *But*, *Alternatively*, and *However* often mark reconsideration of an earlier step. Following Lin et al. (2025), we call them reflection tokens. The set $\mathcal{T}$ contains the first token of each word, tokenized with and without a leading space and deduplicated for each model (Appendix B). The same set is used to measure reflection probability and to bias generation.

Rollback, output limit, and generation cost. A rollback to b retains $y_{\le b}$, deletes the subsequent tokens, and continues generation from that prefix. Each run has a retained-output token limit B and the model's context limit L, enforcing $|y| \le B$ and $|x| + |y| \le L$. Total generation q counts every generated token, including those later deleted by rollback, and measures generation cost. Deleted tokens remain in q but no longer occupy the retained output, so total generation can exceed B. Section 4 defines how ReTrail selects rollback points and biases their continuations. We assume access to next-token probabilities, reflection-token logit bias, and continuation from a chosen prefix. Appendix C describes how the hosted implementation supplies these operations.

4 Method: ReTrail

4.1 Overview

ReTrail revises a reasoning trajectory during generation using the model's own reflection-token probabilities. Figure 2 illustrates the procedure in three stages. At each reasoning-step boundary, ReTrail measures the total probability of reflection tokens, which we call hesitation mass. A rise above the trajectory's reference level triggers revision (①). To choose where revision begins, ReTrail selects an earlier boundary with high hesitation mass among the recent steps. The trajectory is then rolled back to the selected boundary. The prefix through that boundary is retained, and all text generated after that boundary up to the current end of the trajectory is discarded (②).

Generation then resumes from the retained prefix with a temporary bias on reflection-token logits (③). This bias first promotes reflection and then suppresses it before returning to zero, a sequence we call bidirectional hedging. Detection pauses during this guided regeneration and a further period of unbiased generation. It can resume on the revised trajectory, subject to a limit on the number of interventions. The next sections specify the detection rule, rollback selection, and regeneration bias.

4.2 Hesitation Mass and Detection

At a reasoning-step boundary b_i, we define the hesitation mass as the total probability assigned to the reflection-token set $\mathcal{T}$:

$$m_i = \sum_{t \in \mathcal{T}} \pi(t \mid x, y_{\le b_i}), \tag{1}$$

where $y_{\leq b_i}$ is the generated prefix through b_i. This quantity measures the model's tendency to begin the next step with reflection and is available whether or not a reflection token is sampled. The mass sums all reflection-token probabilities in the sampling distribution defined in Section 3.

Since typical hesitation levels vary across models and problems, ReTrail compares each mass with earlier values from the same trajectory. Let $\mathcal{H}_i$ contain the masses at earlier boundaries of the retained trajectory, excluding the guided regeneration periods described in Section 4.4. This exclusion prevents the regeneration bias from directly shifting the reference level for detection. The detection threshold accounts for both the average hesitation mass and its variation across earlier boundaries:

$$\theta_i = \mu(\mathcal{H}_i) + \lambda\sigma(\mathcal{H}_i). \tag{2}$$

Here, $\mu(\mathcal{H}_i)$ is the historical mean, and $\sigma(\mathcal{H}_i)$ is the population standard deviation. A larger λ raises the threshold, requiring a greater departure from the historical mean to trigger rollback. This comparison requires at least $H_{\min}$ history values with nonzero variance. During detection, the reasoning trajectory is rolled back when the current hesitation mass m_i exceeds θ_i.

4.3 Localized Rollback

The rise detected at b_τ may follow an earlier step that needs reconsideration. We use the hesitation masses at preceding boundaries to choose where regeneration begins. Let p denote the previous rollback point, initially zero. The candidate set $\mathcal{W}_\tau$ contains indices of boundaries represented in $\mathcal{H}_\tau$ that satisfy $\max(p, b_\tau - W) < b_i < b_\tau$. This restricts rollback to the preceding W tokens and keeps each rollback point after the previous one.

A higher recorded mass indicates a stronger tendency to begin the next step with reflection. We therefore rank candidates by m_i. A tolerance ϵ groups candidates whose masses are close to the maximum, analogous to the tie margin in Singh et al. (2025). Let $M_\tau = \max_{i\in\mathcal{W}_\tau} m_i$. We select the latest candidate within ϵ of this maximum:

$$b_\star = \max\{b_i : i \in \mathcal{W}_\tau,\ M_\tau - m_i \leq \epsilon\}. \tag{3}$$

This choice preserves the longest prefix among the candidates within the tolerance.

ReTrail retains $y_{\leq b_\star}$ and deletes the continuation after $b_\star$. It removes the deleted boundaries from the reference history and sets $p = b_\star$. Generation then resumes from the retained prefix with the bias described in Section 4.4. Appendix J gives the exact history and candidate definitions and specifies how an empty candidate set is handled.

4.4 Bidirectional Hedging

Resuming from the retained prefix can reproduce the discarded reasoning. ReTrail therefore applies a short sequence of reflection promotion followed by suppression. We use promotion to encourage reconsideration after rollback and suppression to limit repetitive reflection as generation continues.

Let s count tokens generated since rollback and z_t be the logit of token t. Before sampling, we apply

$$\tilde{z}_t = z_t + \delta(s)\,\mathbf{1}[\,t \in \mathcal{T}\,]. \tag{4}$$

After each rollback, $\delta(s)$ follows a fixed schedule, which we call the waveform, over C generated tokens. It begins at $+A$, raising the logits of reflection tokens relative to the rest of the vocabulary to encourage reflection. The bias then decreases through zero to $-A$, lowering those logits to suppress reflection tokens before returning to zero. Thus, A sets the maximum absolute logit adjustment and C sets the duration of the schedule. Each rollback starts one application of this schedule from the retained prefix. After the C tokens, generation continues without logit bias until another rollback starts a new sequence. Appendix J specifies the exact bias values and durations of the eight phases.

Because the bias changes the probabilities used for detection, ReTrail excludes boundaries within these C tokens from the reference history and future rollback candidates. We call these boundaries *dirty* and all others *clean*. The exclusion includes the schedule's zero-bias phases. The reference history contains only earlier clean boundaries that remain in the retained trajectory.

Detection is paused during the C-token schedule and a further R tokens of unbiased generation, called the cooldown. Clean boundaries observed during the cooldown enter the reference history

alongside those retained after rollback. Detection can resume at a boundary after the latest rollback point once $C + R$ tokens have been generated and fewer than K interventions have occurred. Algorithm 1 gives the full execution and termination rules; Appendix D lists the parameter settings.

5 EXPERIMENTS

We evaluate end-to-end accuracy and generation cost, then isolate the roles of rollback and reflection bias. Controlled comparisons test which signals help determine when to revise and where to resume.

5.1 EXPERIMENTAL SETUP

Models and tasks. We evaluate `DeepSeek-R1-Distill-Qwen-7B` (Qwen-7B) (DeepSeek-AI, 2025), `DeepSeek-V3.2-Speciale`, `openai/gpt-oss-120b` (OpenAI, 2025), and `Kimi-K2.5-Thinking` (Kimi Team, 2026). Mathematics tasks comprise AIME 2025, AIME 2026, and HMMT 2026; LiveCodeBench (Jain et al., 2025) evaluates code generation. Component ablations cover two model sizes, GPT-OSS-120B and Qwen-7B, on AIME 2026 and LiveCodeBench. The trigger and location comparisons use Qwen-7B on these two benchmarks. Table 1 averages three seeds, whereas these controlled comparisons use the first seed, fixed before evaluation. Because a single run can differ from the three-seed mean, we compare their accuracies only within each table. In a single run, each AIME problem accounts for 3.33 accuracy points and each LiveCodeBench problem for 1.25. Appendix E lists model configurations and dataset versions.

Protocol and metrics. Complete-policy comparisons share problems, prompts, and sampling parameters within each model–benchmark setting. Reasoning length varies across problems and decoding policies. Enforcing identical realized token counts can terminate a solution before its final answer. To avoid such premature termination solely for length matching, we preserve each method's stopping rules under shared context and retained-output limits. We report final-answer accuracy alongside total generated tokens, including those discarded by rollback, to quantify the accuracy–generation trade-off. Accuracy uses exact match for mathematics and execution-based pass@1 for coding. Code tests are used only for final grading, so no decoding decision depends on test outcomes. Table 1 reports retained output length, while Figure 3 includes deleted tokens. All component and signal comparisons report total generation. Appendix E gives sampling settings and output limits; Appendix L specifies the continuation allowance for the fixed-trigger experiment.

Baselines and variants. We compare against unmodified decoding (Original), the thought-switching penalty (TIP) (Wang et al., 2025b), and CyclicReflex (Fan et al., 2026). TIP penalizes tokens associated with thought switching, whereas CyclicReflex alternates promotion and suppression of reflection tokens through a periodic logit schedule. Both methods apply preset rules as generation continues from the current prefix and never return to an earlier step. The comparison therefore tests whether deciding when to intervene and where to resume improves on preset reflection regulation. TIP and CyclicReflex use the values recommended in their original papers for their own parameters and the ReTrail configuration for shared settings such as the context limit. The component ablation changes one design choice at a time. The w/o rollback variant applies the full waveform at the detection boundary and tests whether revision needs to resume from an earlier step. The w/o regeneration bias variant keeps rollback but regenerates without bias over the same C tokens. This variant tests whether sampling again from the selected prefix is sufficient. The w/ promotion-only bias and w/ suppression-only bias variants retain rollback and use $+|\delta(s)|$ and $-|\delta(s)|$, respectively, to test whether the bias needs both directions. All variants share the detector and pre-intervention sampling settings. The two single-direction variants also preserve the waveform duration and integrated absolute bias of ReTrail, so their comparison with ReTrail isolates the bias direction (Appendix D).

5.2 END-TO-END ACCURACY AND GENERATION COST

ReTrail achieves fourteen wins and two ties for the highest mean accuracy across the sixteen settings (Table 1). Gains over Original average 3.25 percentage points and range from 1.11 to 8.08 points. Appendix F analyzes gain reliability with seed-level variation and paired confidence intervals. All

Table 1: **End-to-end accuracy and generation length.** Acc. (%): exact match for mathematics and pass@1 for LiveCodeBench. Len.: mean retained output tokens per problem, in thousands; Figure 3 includes deleted tokens. All accuracies average three runs. Each run contains 30 problems per AIME benchmark, 33 HMMT problems, and 80 LiveCodeBench problems. Bold and underline indicate the best and second-best mean accuracy within each model–benchmark pair.

	AIME 2025		AIME 2026		HMMT 2026		LiveCodeBench	
Method	Acc. ↑	Len. ↓	Acc. ↑	Len. ↓	Acc. ↑	Len. ↓	Acc. ↑	Len. ↓
DeepSeek-R1-Distill-Qwen-7B								
Original	<u>48.89</u>	12.79k	<u>46.67</u>	14.59k	36.36	19.58k	<u>37.08</u>	11.88k
TIP (Wang et al., 2025b)	46.67	12.65k	42.22	11.26k	<u>37.37</u>	18.84k	36.25	11.28k
CyclicReflex (Fan et al., 2026)	48.89	11.41k	38.89	12.40k	**38.38**	19.16k	35.83	12.47k
▶ ReTrail	**51.11**	14.25k	**47.78**	17.91k	**38.38**	21.59k	**38.75**	14.17k
DeepSeek-V3.2-Speciale								
Original	84.44	20.80k	82.22	22.72k	79.80	33.14k	79.17	32.41k
TIP (Wang et al., 2025b)	<u>87.78</u>	18.69k	75.56	21.76k	74.75	31.79k	78.75	30.24k
CyclicReflex (Fan et al., 2026)	85.56	23.68k	83.33	22.84k	82.83	32.41k	80.42	32.81k
▶ ReTrail	**88.89**	23.02k	**90.00**	22.88k	**87.88**	32.72k	**83.75**	34.43k
GPT-OSS-120B								
Original	<u>72.22</u>	5.18k	<u>78.89</u>	5.08k	<u>72.73</u>	5.19k	<u>77.92</u>	4.42k
TIP (Wang et al., 2025b)	**73.33**	4.77k	77.78	4.46k	71.72	4.53k	68.75	4.27k
CyclicReflex (Fan et al., 2026)	72.22	5.14k	78.89	5.35k	70.71	4.98k	75.83	3.62k
▶ ReTrail	**73.33**	5.36k	**82.22**	5.32k	**75.76**	5.10k	**79.58**	3.71k
Kimi-K2.5-Thinking								
Original	86.67	15.16k	<u>92.22</u>	20.42k	82.83	17.97k	86.25	15.35k
TIP (Wang et al., 2025b)	84.44	14.76k	82.22	20.37k	80.81	16.86k	83.33	14.28k
CyclicReflex (Fan et al., 2026)	88.89	16.24k	90.00	22.10k	83.84	18.74k	86.67	14.64k
▶ ReTrail	**92.22**	17.29k	**93.33**	21.21k	**85.86**	19.24k	**87.50**	16.18k

models and benchmarks share one ReTrail configuration, selected by a small grid search on MATH-500 (Lightman et al., 2024), which is disjoint from the evaluation benchmarks (Appendix D).

The mean accuracy gain accompanies a 15.2% average increase in total generation, including tokens later deleted by rollback (Figure 3). On DeepSeek-V3.2-Speciale, accuracy rises by 7.78 points on AIME 2026 and 8.08 points on HMMT 2026 with less than 5% additional generation. GPT-OSS-120B–LiveCodeBench improves both measures: accuracy rises from 77.92% to 79.58%, while total generation falls from 4.42k to 3.94k tokens. Here the shorter final output more than offsets the 0.23k tokens discarded by rollback. When s1 budget forcing (Muennighoff et al., 2025) generates as many tokens as ReTrail on each problem, its first-seed accuracy on Qwen-7B remains 13.34 points lower on AIME 2026 and 5.00 points lower on LiveCodeBench (Appendix H).

5.3 Component Ablation

Rollback. ReTrail and w/o rollback share the detection rule and waveform, so their comparison isolates the effect of resuming from an earlier step. Rollback raises accuracy by 3.75–23.34 points in all four settings and reduces total generation in each. The largest gain appears on Qwen-7B–AIME 2026, from 33.33% to 56.67%, and its paired 95% confidence interval lies above zero (Appendix F). On GPT-OSS-120B–LiveCodeBench, total generation falls by 36.2%. These results support the premise of Section 1 that revision is more effective when generation returns to an earlier step.

Regeneration bias. ReTrail gains 3.33–13.34 accuracy points over w/o regeneration bias. Both use the same detector and rollback rule, so the gain comes from the regeneration bias. Without this bias, regeneration from the retained prefix can reproduce the discarded reasoning, which motivates bidirectional hedging (Section 4.4). The gains in all four settings show that guiding regeneration improves on resampling alone.

Bias direction. Combining promotion and suppression gives the highest accuracy in all four settings. The w/ promotion-only bias variant falls below w/o regeneration bias in every setting, whereas the w/ suppression-only bias variant matches it on AIME 2026 and exceeds it on LiveCodeBench. Without the subsequent suppression, promoted reflection persists on some problems until the trajectory reaches the context limit and is truncated before a final answer. In ReTrail, the suppression phase that follows promotion limits repetitive reflection and supports progress toward an answer

Table 2: **Component ablation on GPT-OSS-120B and Qwen-7B.** First-seed responses on 30 AIME 2026 and 80 LiveCodeBench problems. Variant names specify the change from ReTrail. Variants share the detection rule and retained-output token limit. Acc. (%): exact match for AIME and pass@1 for LiveCodeBench. Len.: mean total generated tokens per problem in thousands, including deleted suffixes. Bold and underline mark the best and second-best distinct accuracies within each model–benchmark pair; ties share the same marking.

	AIME 2026		LiveCodeBench	
Method	Acc. ↑	Len. ↓	Acc. ↑	Len. ↓
GPT-OSS-120B				
▶ ReTrail	**80.00**	5.49k	**82.50**	3.58k
w/o rollback	70.00	6.73k	77.50	5.61k
w/ promotion-only bias	73.33	4.52k	68.75	3.70k
w/ suppression-only bias	<u>76.67</u>	5.77k	<u>78.75</u>	4.81k
w/o regeneration bias	<u>76.67</u>	4.24k	75.00	3.36k
DeepSeek-R1-Distill-Qwen-7B				
▶ ReTrail	**56.67**	18.21k	**41.25**	13.16k
w/o rollback	33.33	18.59k	<u>37.50</u>	14.96k
w/ promotion-only bias	36.67	18.06k	32.50	14.37k
w/ suppression-only bias	<u>43.33</u>	19.80k	<u>37.50</u>	15.75k
w/o regeneration bias	<u>43.33</u>	18.31k	35.00	13.94k

(Section 4.4). The combination also uses fewer tokens than suppression alone in every setting. On GPT-OSS-120B–LiveCodeBench, it exceeds w/ suppression-only bias by 3.75 accuracy points with 25.6% fewer tokens.

5.4 Trigger and Rollback Controls

Table 3 changes the trigger while holding mass-based rollback and regeneration fixed. The entropy trigger replaces hesitation mass with next-token entropy, a standard measure of model uncertainty, and the random trigger fires with a fixed probability at each eligible boundary. Both are calibrated on separate AIME 2025 trajectories to match the candidate trigger count of the hesitation rule. This calibration equalizes how often each rule proposes an intervention, so the comparison tests when to intervene. Calibration uses no correctness labels, and both settings are fixed before evaluation (Appendix K).

Hesitation-based triggering improves accuracy over both alternatives by 6.67 points on AIME 2026 and 3.75 points on LiveCodeBench. On AIME 2026, ReTrail and the random trigger intervene 2.40 and 2.37 times per problem on average. On LiveCodeBench, ReTrail intervenes least, 1.93 times per problem versus 2.80 for the entropy trigger and 2.55 for the random trigger (Table 8). Hesitation-based triggering thus improves accuracy with comparable or fewer interventions, so its advantage lies in choosing when to intervene.

Table 8 also varies the rollback rule. Under the hesitation trigger, replacing fixed-distance rollback of about 512 tokens with mass-based selection raises accuracy from 43.33% to 56.67% on AIME

Table 3: **Trigger controls on Qwen-7B.** First-seed responses on 30 AIME 2026 and 80 LiveCodeBench problems. All three rules share mass-based rollback, the waveform, and cooldown.

	AIME 2026		LiveCodeBench	
Method	Acc. ↑	Total (k)	Acc. ↑	Total (k)
ReTrail	**56.67**	18.21	**41.25**	13.16
w/ entropy trigger	50.00	19.77	37.50	14.19
w/ random trigger	50.00	17.26	37.50	12.39

Acc.: exact match (AIME) or pass@1 (LiveCodeBench), in percent. Total: mean generated tokens in thousands, including deleted tokens. Table 8 reports all eight variants and the token and intervention statistics.

Table 4: **Restart location at a fixed first trigger on Qwen-7B.** Three continuations per location and problem: 90 answers on AIME 2026 and 240 on LiveCodeBench. All groups share the first trigger and subsequent bias.

	AIME 2026		LiveCodeBench	
Method	Acc. ↑	Total (k)	Acc. ↑	Total (k)
ReTrail (mass sub-spike)	**52.22**	14.00	**37.92**	13.14
w/ entropy-peak rollback	44.44	15.38	35.83	12.99
w/ distance-matched random rollback	42.22	18.26	35.83	12.81
w/o rollback	48.89	14.46	35.00	12.61

Acc.: mean accuracy (%; AIME exact match, LiveCodeBench pass@1). Total: mean generated tokens, including deleted tokens; the shared prefix is counted once per continuation. Appendix L gives the full protocol.

2026 and from 36.25% to 41.25% on LiveCodeBench. Under the random trigger, the same replacement raises AIME 2026 accuracy from 36.67% to 50.00% and also improves LiveCodeBench accuracy. Mass-based rollback therefore improves accuracy under both triggers. Section 5.5 isolates this location choice at a fixed trigger.

5.5 Restart Location at a Fixed Trigger

We fix the first trigger on each first-seed Original Qwen-7B trajectory, the first boundary where ReTrail intervenes. The mass sub-spike is the rollback point that ReTrail selects by Equation 3. We compare it with three alternatives: the entropy peak, a random candidate in the same 512-token distance bin, and the current trigger boundary. All locations share the trajectory, trigger, and subsequent bias and differ only in where generation resumes. Each location receives three continuations with matched seeds, and all continuations enter the mean with equal weight per problem (Appendix L).

The mass sub-spike yields the highest mean accuracy on both tasks (Table 4). On AIME 2026, it exceeds the entropy peak by 7.78 points and the random location by 10.00 points, and it also uses the fewest tokens. On LiveCodeBench, it exceeds the current boundary by 2.92 points. The paired 95% confidence intervals of these three gains lie above zero (Appendix F). With the trigger and bias fixed, these results directly support using hesitation mass to choose where revision begins.

Supplementary analyses. Cases in Appendix I.1 show successful and unsuccessful revision. In an arithmetic problem, regeneration restores a missed positivity constraint and corrects the answer. In a permutation problem, it rechecks a convention but leaves an incomplete enumeration unrepaired. A third case shows a local error followed by recovery. Appendix I relates these observations to the hesitation signal and compares it with entropy. On 40 Original Qwen-7B trajectories, the two signals are correlated, with median within-trajectory rank correlations of 0.77 on AIME 2026 and 0.62 on LiveCodeBench (Appendix M.1), yet their top 10% positions overlap by only 46.3% and 32.8% (Appendix M.2). A strong preference for reflection tokens therefore differs from uncertainty across the vocabulary.

6 Conclusion

We presented ReTrail, a training-free decoding method that uses the model's own reflection-token probabilities to decide when to revise and where to resume. Localized rollback returns generation to an earlier step, and a short bidirectional bias guides its reconsideration within the same trajectory. The method requires no separately trained model and supports local and hosted implementations.

Across four reasoning models and four mathematics and coding benchmarks, ReTrail achieves the highest or tied-highest mean accuracy in every setting. The average gain over unmodified decoding is 3.25 percentage points, with a maximum of 8.08 points. The full method also achieves the highest accuracy in all four ablation settings. Rollback improves accuracy and reduces total generation in each setting, while bidirectional hedging improves accuracy over unbiased regeneration and exceeds suppression alone with fewer generated tokens. These results support using reflection-token probabilities to guide when and where to revise a single reasoning trajectory.

AI USE STATEMENT

We used generative AI tools to implement parts of the method, provide feedback on the experimental design, translate text into English, and help interpret experimental results. We did not use them to generate data, develop the conceptual framework, formulate mathematical definitions or claims, propose hypotheses, clean datasets, or perform qualitative analysis; the paper contains no proofs. Additionally, we used these tools to brainstorm research directions, search for and summarize related work, write and edit experiment and plotting code, draft figure sketches, suggest the paper structure, draft parts of the method and experimental analysis sections, polish the writing, and give simulated reviewer feedback on drafts. The authors reviewed and tested all AI-assisted code, checked every cited reference against its source, and revised all AI-drafted text; all reported numbers come from actual experiment runs. The authors take full responsibility for the final content of this paper, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12375–12396. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.761. URL https://aclanthology.org/2023.emnlp-main.761/.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Kang Chen, Fan Yu, Junjie Nian, Shihan Zhao, Zhuoka Feng, Zijun Yao, Heng Wang, Minshen Yu, and Yixin Cao. Thinking traps in long chain-of-thought: A measurable study and trap-aware adaptive restart. *arXiv preprint arXiv:2601.11940*, 2026. URL https://arxiv.org/abs/2601.11940.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do NOT think that much for 2+3=? On the overthinking of long reasoning models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 9487–9499. PMLR, 2025. URL https://proceedings.mlr.press/v267/chen25bx.html.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Chongyu Fan, Yihua Zhang, Jinghan Jia, Alfred O. Hero, and Sijia Liu. CyclicReflex: Improving reasoning models via cyclical reflection token scheduling. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=4oOF4J2xSy.

Jiameng Huang, Baijiong Lin, Guhao Feng, Jierun Chen, Di He, and Lu Hou. Efficient reasoning for large reasoning language models via certainty-guided reflection suppression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(37):31176–31184, 2026. doi: 10.1609/aaai.v40i37.40379. URL https://ojs.aaai.org/index.php/AAAI/article/view/40379.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2403.07974.

Lingyu Jiang, Zirui Li, Shuo Xing, Peiran Li, Tsubasa Takahashi, Dengzhe Hou, Zhengzhong Tu, Kazunori Yamada, and Fangzhou Lin. PathCal: State-aware reflection-marker calibration for efficient reasoning. *arXiv preprint arXiv:2605.23074*, 2026. URL https://arxiv.org/abs/2605.23074.

Kimi Team. Kimi K2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026. URL https://arxiv.org/abs/2602.02276.

Xianzhi Li, Ethan Callanan, Abdellah Ghassel, and Xiaodan Zhu. Entropy-gated branching for efficient test-time reasoning. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5054–5069, 2026. doi: 10.18653/v1/2026.eacl-long.235. URL https://aclanthology.org/2026.eacl-long.235/.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Weizhe Lin, Xing Li, Zhiyuan Yang, Xiaojin Fu, Hui-Ling Zhen, Yaoyuan Wang, Xianzhi Yu, Wulong Liu, Xiaosong Li, and Mingxuan Yuan. TrimR: Verifier-based training-free thinking compression for efficient test-time scaling. *arXiv preprint arXiv:2505.17155*, 2025. URL https://arxiv.org/abs/2505.17155.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL https://arxiv.org/abs/2303.17651.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. URL https://arxiv.org/abs/2508.10925.

Ankur Samanta, Akshayaa Magesh, Ayush Jain, Kavosh Asadi, Youliang Yu, Daniel Jiang, Boris Vidolov, Kaveh Hassani, Paul Sajda, Jalaj Bhandari, and Yonathan Efroni. Structure enables effective self-localization of errors in LLMs. *arXiv preprint arXiv:2602.02416*, 2026. URL https://arxiv.org/abs/2602.02416.

Anikait Singh, Kushal Arora, Sedrick Keh, Jean Mercat, Tatsunori Hashimoto, Chelsea Finn, and Aviral Kumar. Improving test-time search for LLMs with backtracking against in-context value verifiers. In *ICLR 2025 Workshop on Self-Improving Foundation Models Without Human Supervision*, 2025. URL https://iclr.cc/virtual/2025/10000498.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Gladys Tyen, Hassan Mansoor, Victor Cărbune, Peter Chen, and Tony Mak. LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Chenlong Wang, Yuanning Feng, Dongping Chen, Zhaoyang Chu, Ranjay Krishna, and Tianyi Zhou. Wait, we don't need to "wait"! removing thinking tokens improves reasoning efficiency. *arXiv preprint arXiv:2506.08343*, 2025a. URL https://arxiv.org/abs/2506.08343.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Thoughts are all over the place: On the underthinking of long reasoning models. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 30591–30611. Curran Associates, Inc., 2025b. doi: 10.52202/085713-1027. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/2bfc4acb2e5c4af1576d14b0df9a664c-Paper-Conference.pdf.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models. In *International Conference on Learning Representations*, volume 2026, pp. 88170–88210, 2026. URL https://openreview.net/forum?id=NpU7ZXafRi.

Felix Zhou, Anay Mehrotra, and Quanquan C. Liu. Reasoning with sampling: Cutting at decision points. *arXiv preprint arXiv:2605.30327*, 2026. URL https://arxiv.org/abs/2605.30327.

A Step-Boundary Segmentation

Step boundaries give ReTrail a consistent unit for measuring hesitation mass and selecting a prefix for regeneration. The implementation decodes the generated tokens into text and splits at double newlines outside triple-backtick code blocks. An unclosed code block extends to the end of the available text. Each separator is mapped to the first token whose start is at or after the separator's end; the boundary lies immediately before that token. Scanning from left to right, any step shorter than five tokens is merged into the preceding step when one exists. The retained boundaries are the distinct, strictly positive ends of internal steps after merging. The final step end is excluded. Segmentation is recomputed on the retained trajectory after every rollback.

During local generation, a separator is confirmed as a boundary once the following segment contains at least five tokens. A shorter segment is merged backward under the rule above, so generation may end before a boundary is confirmed. The probability recorded at the boundary prefix is then used for detection. Tokens generated while confirming the boundary count toward total generation, including those later deleted by rollback. A hosted query can read probabilities at a specified prefix, but applying the same segmentation rule still requires confirming that prefix's boundary from the continuation. Appendix C describes the query procedure and its cost accounting.

B Reflection Token Set

The reflection tokens are built from the word list

$$\{\textit{Wait}, \textit{But}, \textit{Alternatively}, \textit{However}\}.$$

Each word is tokenized both without a leading space and with one leading space. The local implementation requires each form to encode as a single token and to decode back to the original string. These checks yield eight distinct IDs for the DeepSeek-R1-Distill-Qwen-7B tokenizer used in the local experiments. Its revision is `916b56a44061fd5cd7d6a8fb632557ed4f724f60`. Thus the first-token construction in Section 3 selects the complete word token in these experiments. For another tokenizer, the resolved IDs and any words split across multiple tokens must be recorded before generation. Hesitation mass and logit bias use the same set $\mathcal{T}$ throughout a run. The Original-trajectory visualizations in Appendix M use the same eight word forms.

C Hosted-API Feasibility

An interface study tested DeepSeek-V3.2-Speciale through a hosted chat API. It examined continuation from a retained prefix and reflection-token logit bias, together with recovery of reflection-token probabilities. The recovered probabilities precede temperature scaling and top-p truncation. We distinguish this raw probability mass from the sampling-distribution hesitation mass defined in Section 3.

C.1 Resuming from a prefix

Structural-regex forced-prefix replay. The tested chat endpoint accepts no supplied assistant prefix for direct continuation. Instead, a regular-expression constraint requires the output to reproduce the retained prefix before generating an arbitrary suffix. This constrains the text, while the model selects among tokenizations allowed by the constraint. On the tested self-generated prefixes, replay preserved the token sequence and matched the next free token's log probability. Checks with reflection-token bias also preserved the forced prefix while changing the free continuation. Reuse on other deployments requires checking these properties with the intended bias settings.

Server-side `stop` is disabled because a double newline in the replayed prefix would terminate the request prematurely. The client instead truncates the free suffix at the chosen checkpoint. Boundary confirmation follows the segmentation conditions in Appendix A.

Physical versus logical accounting. The service decodes the replayed prefix and includes it in completion-token usage. A continuation request therefore adds the prefix length to the requested suffix allowance in `max_tokens`. Subtracting the actual replay length from the returned completion count gives the number of newly generated suffix tokens. All such tokens count toward total generation q, including those removed by rollback or client-side truncation. The retained-output limit B applies to the resulting trajectory. Prefix replay and probability-probe requests are recorded separately as service usage; probe outputs are excluded from the trajectory. Service usage and execution time include these operations and any retries.

C.2 Recovering raw reflection-token probability mass

The tested service repeated its top-logprob table across consecutive positions, consistent with reuse within speculative-decoding blocks. Only tables verified to refer to the queried prefix support direct reading. The diagnostic calls a position offset 0 when its table is current; it checks agreement with the selected token's scalar log probability. Direct summation additionally requires every resolved reflection-token ID to appear in the table. Otherwise, the following recovery uses two biased queries at the same prefix.

Let p_t be the full-vocabulary softmax probability before the probe bias, temperature scaling, and top-p truncation. For $S = \mathcal{T}$, define $m_{\mathrm{raw}} = \sum_{t \in S} p_t$. The derivation assumes identical base logits across the two requests and returned probabilities measured after the probe bias but before sampling transformations. Use greedy selection at the queried position and choose $\beta > 0$ so that Request A selects $t^\star \in S$. Request A adds β to every token in S, giving

$$p'_{t^\star} = \frac{p_{t^\star} e^\beta}{1 + m_{\mathrm{raw}}(e^\beta - 1)}. \tag{5}$$

Request B adds the same bias only to $t^\star$. This token remains a maximizer because the competing tokens receive no greater bias than in Request A. Both requests must return the scalar probability for this same token. Request B gives

$$p''_{t^\star} = \frac{p_{t^\star} e^\beta}{1 + p_{t^\star}(e^\beta - 1)}. \tag{6}$$

The base token probability and raw mass then have the closed forms

$$p_{t^\star} = \frac{p''_{t^\star}}{e^\beta - p''_{t^\star}(e^\beta - 1)}, \tag{7}$$

$$m_{\mathrm{raw}} = \frac{p_{t^\star} e^\beta / p'_{t^\star} - 1}{e^\beta - 1}. \tag{8}$$

For a valid β, these equations recover the full set's raw mass from two requests regardless of $|S|$. Additional requests used to adjust β count as probes. The algebra is exact under the stated assumptions; returned floating-point probabilities introduce numerical error.

Bias selection and numerical checks. The probe increases β when Request A selects a token outside S. A smaller bias is needed when $p''_{t^\star}$ approaches one, where rounding can make the inversion

unstable. Reliable recovery requires finite probabilities with $0 < p'_{t^\star}, p''_{t^\star} < 1$ and recovered values satisfying $0 \leq p_{t^\star} \leq m_{\mathrm{raw}} \leq 1$. Agreement with direct summation at readable positions provides an independent numerical check. Each measurement record should identify its reading route and bias, together with probe and replay costs.

Scope of the validation. The interface checks support prefix continuation with logit bias and raw probability recovery on the tested deployment. Recovering the sampling-distribution mass used by ReTrail additionally requires temperature scaling and top-p truncation over the vocabulary. The scalar m_{raw} alone is insufficient for those operations. Thus equivalence to the local hesitation signal requires a separate validation of the processed probabilities. The interface study establishes these capabilities for the tested deployment; each other deployment requires its own checks.

D Primary Settings and Ablation Variants

The primary configuration fixes the detection threshold, rollback range, and duration of each intervention. An alarm needs at least $H_{\min} = 3$ clean boundaries of history and fires when the hesitation mass exceeds the clean mean by $\lambda = 1.5$ standard deviations. The rollback point is searched within a look-back of $W = 2048$ tokens with a tie tolerance of $\epsilon = 0.05$. The waveform has amplitude $A = 5$ and lasts $C = 304$ tokens. After the waveform ends, a cooldown of $R = 512$ tokens must pass before the next alarm may fire, and each trajectory allows at most $K = 3$ interventions. We selected these values by a small grid search on MATH-500 (Lightman et al., 2024), which is disjoint from the evaluation benchmarks.

The clean-history rule and cooldown connect successive interventions. Boundaries within the C biased tokens are excluded from the reference history used for detection. During the following R unbiased tokens, new clean boundaries enter that history before detection is enabled again. Together with the requirement that rollback points advance along the trajectory, these rules specify how ReTrail can revise multiple portions of a single generation (Section 4.4).

The component variants are named by their change from ReTrail: w/o rollback, w/ promotion-only bias, w/ suppression-only bias, and w/o regeneration bias. Apart from the named change, they share the same settings: the detector, alarm threshold, rollback-point rule, waveform amplitude and duration, clean-history rule, cooldown, intervention limit, retained-output token limit, and pre-action random stream. The w/o rollback variant runs the full waveform $\delta(s)$ at the alarm boundary. The w/ promotion-only bias and w/ suppression-only bias variants keep the rollback and replace $\delta(s)$ by $+|\delta(s)|$ and $-|\delta(s)|$; both are sign-rectified copies of the primary waveform, so duration and integrated absolute bias match ReTrail phase by phase and the contrast isolates direction alone. The w/o regeneration bias variant keeps the rollback and applies no bias over the same C tokens, which remain dirty. The w/o rollback and regeneration bias variant stays at the alarm boundary without bias and provides an implementation check: its tokens must match segmented unbiased decoding until an intervention would occur.

D.1 Implementation checks

Before scoring, deterministic replay must cover: no alarm; one alarm with and without suffix deletion; each ablation variant under the same pre-alarm stream; strictly increasing rollback points; saturation at K; output-length, context, and EOS termination; dirty-history exclusion; and hosted mass recovery. A no-action coordinate must remain token-identical across Original-compatible segmented execution and w/o rollback and regeneration bias until a legal action would occur. Any parity, clean-history, or token-accounting failure must be resolved before scoring the corresponding model–benchmark setting.

E Evaluation Protocol

Model and decoding configuration. GPT-OSS uses its native Harmony format and medium reasoning effort; Kimi K2.5 runs in its thinking mode. LiveCodeBench uses `release_v6` and the problem-date range recorded in the evaluation manifest. On coding problems, ReTrail intervenes in the natural-language reasoning stream before the completed program is graded. Code execution

tests are used only for final grading. The default temperature is 0.6 and top-p is 0.95, with any checkpoint-specific compatibility setting fixed before evaluation. Termination follows Algorithm 1.

Table 1 crosses four models with AIME 2025, AIME 2026, HMMT 2026, and LiveCodeBench. Accuracy for each model is averaged over three runs. Each run contains 30 problems per AIME benchmark, 33 HMMT problems, and 80 LiveCodeBench problems. Table 2 reports the component comparison for GPT-OSS-120B and DeepSeek-R1-Distill-Qwen-7B on 30 AIME 2026 problems and 80 LiveCodeBench problems, with one response per problem for each variant. Comparisons between variants use the settings and samples of that table.

For the fixed-trigger location experiment, we approximate hesitation mass using reflection tokens among the top 20 token probabilities after temperature scaling and top-p truncation. This reduces probability-logging overhead, as tokens outside the top 20 have small individual probabilities.

Mathematics accuracy uses exact match after answer extraction; LiveCodeBench accuracy uses execution-based pass@1 after code extraction. Reported gains are differences in percentage points. The overall accuracy gain averages these differences equally across the sixteen model–benchmark settings, giving each task and model the same weight. This yields a mean gain of 3.25 points over Original from the values in Table 1.

The retained-output limit is $B = L - |x|$, the model's context limit minus the input length. Methods evaluated on the same problem share this limit. Every sampled token, including deleted suffixes, is counted in total generation to measure generation cost. In Table 1, Len. measures retained output; in Table 2, it measures total generation. Figure 3 supplies the corresponding total generation for the main comparison. Hosted prefix replay and probability probes are accounted for as separate service operations (Appendix C).

For the local implementation, a counter-based sampler keys draws by checkpoint, problem, seed, and cumulative generated tokens. Methods share the stream until their first different action, then use substreams indexed by variant and intervention. Segmented unbiased decoding shares the boundary detection and generation checkpoints with ReTrail and provides the implementation-matched reference for runs with no intervention. Retries for transport or execution failures reuse the original run settings. The evaluation manifest specifies the model revision and tokenizer together with the prompt, task version, sampling settings, output limit, and grader. It also records the repetitions and any incomplete runs for each comparison.

Evaluating revision decisions through continuation outcomes. We assess a revision decision by the success of the reasoning that follows it. CyclicReflex measures self-correction by re-answering questions with truncated traces from incorrect solutions (Fan et al., 2026). PathCal compares final-answer success after forcing different markers following the same prefix (Jiang et al., 2026). We apply this experimental logic to the timing and location of revision.

The complete-policy comparisons measure the effect of each rule throughout generation. The fixed-trigger experiment holds the decision to intervene and the subsequent regeneration rule fixed while changing the restart location. Mean accuracy over repeated continuations estimates the success rate from each selected prefix. Reference answers and code execution tests provide the outcome labels, directly evaluating the usefulness of the restart choice for solving the problem.

F Accuracy Gains and Statistical Uncertainty

Tables 5 and 6 quantify the size and uncertainty of the Qwen-7B accuracy gains. They distinguish end-to-end comparisons with the strongest baselines from controlled comparisons of revision decisions and method components. Each gain is ReTrail accuracy minus the comparator accuracy on the same benchmark, in percentage points (pp). Displayed gains follow the rounded accuracies in the source tables; SD and confidence intervals (CIs) summarize the matched observations.

Table 5: **Qwen-7B gains across repeated seeds.** Mean gain and sample standard deviation (SD) are computed across paired seed-level accuracy differences. N is the number of problems and S the number of repeats. All gain statistics are in percentage points.

Benchmark	Comparator	$N \times S$	Mean gain $\pm$ SD	95% CI
End-to-end accuracy (Table 1)				
AIME 2025	Original	30×3	$+2.22 \pm 1.92$	$[0, 5.56]$
AIME 2025	CyclicReflex	30×3	$+2.22 \pm 3.85$	$[-2.22, 6.67]$
AIME 2026	Original	30×3	$+1.11 \pm 1.92$	$[0, 3.33]$
HMMT 2026	CyclicReflex	33×3	0.00 ± 0.00	$[0.00, 0.00]$
LiveCodeBench	Original	80×3	$+1.67 \pm 2.60$	$[-0.42, 4.17]$
Restart location at a fixed trigger (Table 4)				
AIME 2026	w/ entropy-peak rollback	30×3	$+7.78 \pm 3.85$	$[2.22, 13.33]$
LiveCodeBench	w/ entropy-peak rollback	80×3	$+2.09 \pm 2.89$	$[-0.42, 4.58]$
AIME 2026	w/ distance-matched random rollback	30×3	$+10.00 \pm 3.33$	$[3.33, 16.67]$
LiveCodeBench	w/ distance-matched random rollback	80×3	$+2.09 \pm 2.89$	$[-0.42, 4.58]$
AIME 2026	w/o rollback	30×3	$+3.33 \pm 3.33$	$[-1.11, 7.78]$
LiveCodeBench	w/o rollback	80×3	$+2.92 \pm 2.60$	$[0.83, 5.42]$

The first block includes both tied strongest baselines on AIME 2025 and the tied best method on HMMT 2026. In the second block, seeds vary the continuation after a fixed first trigger on the same Original trajectory; its SD measures continuation variability conditional on that trajectory.

Table 6: **Qwen-7B gains in single-run comparisons.** Gain is the observed accuracy difference in one run, in percentage points. Across-seed SD is not defined for these comparisons.

Benchmark	Comparator	Gain (pp)	95% CI (pp)
Component contributions (Table 2)			
AIME 2026	w/o rollback	$+23.34$	$[6.67, 40.00]$
LiveCodeBench	w/o rollback	$+3.75$	$[-1.25, 10.00]$
AIME 2026	w/o regeneration bias	$+13.34$	$[0.00, 30.00]$
LiveCodeBench	w/o regeneration bias	$+6.25$	$[-1.25, 13.75]$
Trigger choice (Table 3)			
AIME 2026	w/ entropy trigger	$+6.67$	$[0.00, 16.67]$
LiveCodeBench	w/ entropy trigger	$+3.75$	$[-1.25, 10.00]$
AIME 2026	w/ random trigger	$+6.67$	$[0.00, 16.67]$
LiveCodeBench	w/ random trigger	$+3.75$	$[-1.25, 10.00]$

Uncertainty measures. For matched run s, let $\Delta_s = \mathrm{Acc}(ReTrail, s) - \mathrm{Acc}(\text{comparator}, s)$. Mean gain and SD in Table 5 summarize these paired differences across the three observed seeds. Both tables report two-sided 95% percentile CIs from paired question-level bootstrap resampling. Each sampled problem carries both methods' results from all available repeats together. Thus, repeated answers to one problem remain grouped when estimating uncertainty over problems. The SD describes variation across the observed runs, whereas the CI describes question-sampling uncertainty conditional on those generations. For the fixed-trigger comparisons, the repeated seeds vary only the continuation from the shared Original trajectory. Table 6 provides question-level CIs for a single run; across-seed SD requires repeated runs. The intervals are comparison-specific, with inference about new generation seeds limited by the available repeats.

Sampling variability motivates paired evaluation. Different generations from the same model can produce different reasoning paths and final answers for the same problem. Best-of-N (BoN) exploits this variation by sampling multiple candidates and selecting one with a verifier (Snell et al., 2024); self-consistency instead aggregates answers across sampled reasoning paths (Wang et al., 2023). Repeated-sampling studies further show that additional attempts can solve problems missed by an initial generation (Brown et al., 2024). This evidence motivates measuring how a decoding method changes success rates in the presence of sampling variability. Our paired differences directly address that question, and the reported means include every evaluated repeat with equal weight.

End-to-end gains against strong baselines. The first block of Table 5 reports gains of 2.22 points on AIME 2025, 1.11 on AIME 2026, and 1.67 on LiveCodeBench against Original. On HMMT 2026, ReTrail ties CyclicReflex. These differences favor ReTrail or preserve the strongest baseline's accuracy, but their size makes uncertainty consequential. On a 30-problem AIME evaluation, one additional correct answer changes single-run accuracy by 3.33 points. Averaging three runs reduces that increment to 1.11 points, the size of the AIME 2026 gain over Original.

All end-to-end intervals in this block include zero. An interval containing zero, including at an endpoint, leaves a positive population gain unresolved at the stated confidence level; it does not establish equivalence or absence of benefit. For example, the LiveCodeBench interval $[-0.42,\ 4.17]$ spans a small loss and a larger positive gain around the observed $+1.67$ points. The appropriate reading therefore retains both the favorable estimate and its uncertainty. The HMMT result is an observed tie: its $[0,\ 0]$ interval describes the empirical resampling distribution, not a guarantee of identical outcomes under future generations.

Controlled comparisons provide clearer evidence for restart selection. The second block of Table 5 holds the first trigger and regeneration bias fixed while changing where generation resumes. On AIME 2026, hesitation-based selection improves over the entropy peak by 7.78 points with CI $[2.22,\ 13.33]$, and over the distance-matched random location by 10.00 points with CI $[3.33,\ 16.67]$. On LiveCodeBench, its gain over continuing at the trigger boundary is 2.92 points with CI $[0.83,\ 5.42]$. These strictly positive intervals provide evidence for useful restart selection conditional on the shared trajectories. The remaining location comparisons also have positive mean gains, with intervals that include zero. Together, the estimates support selecting a prefix from which revision is more likely to succeed, with the clearest separation on AIME 2026 against alternative location rankings.

Component and trigger comparisons support the same direction. In Table 6, the AIME 2026 gain over w/o rollback is 23.34 points with CI $[6.67,\ 40.00]$. This supplies additional evidence for the contribution of rollback under the complete decoding policy. Removing regeneration bias or replacing the hesitation trigger also reduces observed accuracy on both tasks. Those intervals include zero, so the data estimate their contributions less precisely. These comparisons share problems and ReTrail outputs, so their agreement provides complementary evidence within the same evaluation.

What the supplementary analysis adds. CyclicReflex reports accuracy and generation length in its main comparisons (Fan et al., 2026), while PathCal explicitly reports main results using a single shared seed without averaging over multiple seeds (Jiang et al., 2026). Tables 5 and 6 extend such comparisons by making the uncertainty of the paired gain explicit. The combination of favorable end-to-end estimates and clearer gains in controlled comparisons supports the usefulness of hesitation-guided revision. The uncertainty analysis identifies where that evidence is strongest: rollback and selected restart comparisons provide clearer support than the smaller end-to-end margins.

G Realized Cost behind Table 1

Figure 3 relates the accuracy gains in Table 1 to total generation. Original generates a single pass, so its output length is also its total token use. For ReTrail, total generation is the retained output plus every token discarded by rollback. The hatched portion makes this additional work visible for each model–benchmark pair. Deleted suffixes contribute 0.23k–2.27k tokens per problem.

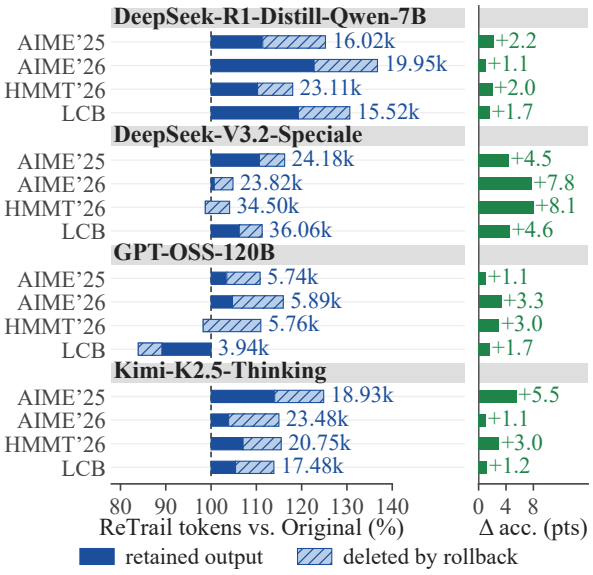

Figure 3: Realized token spend behind Table 1. Each bar starts at the Original spend (dashed line, 100%) and runs to the output ReTrail retains; the hatched part adds the tokens deleted by rollback, so the bar ends at the total spend, printed after it in thousands of tokens. Right: accuracy change over Original from Table 1, in points. LCB is LiveCodeBench.

Across the sixteen settings, the mean of the percentage changes in total generation is $+15.2\%$, accompanying the 3.25-point mean accuracy gain. DeepSeek-V3.2-Speciale gives the largest average accuracy improvement, 6.22 points, with 9.1% more generated tokens on average. On its AIME 2026 and HMMT 2026 evaluations, token increases stay below 5% while accuracy improves by 7.78 and 8.08 points.

Qwen-7B gains 1.76 points with 27.7% more tokens on average. GPT-OSS-120B gains 2.28 points with 6.7% more tokens, and Kimi-K2.5-Thinking gains 2.74 points with 17.3% more tokens. GPT-OSS-120B–LiveCodeBench improves both measures: accuracy rises by 1.66 points and total generation falls by 10.9%. This reduction shows that the benefit of revising an earlier step can offset the tokens discarded by rollback.

The figure measures newly generated tokens. Hosted execution additionally requires prefix replay and probability probes, whose accounting is described in Appendix C.

H Comparison at Matched Token Counts

Table 1 compares methods under a shared retained-output limit. ReTrail nevertheless generates more tokens in total, because rollback deletes part of the trajectory before generation continues (Appendix G). This section compares ReTrail with a method that generates the same number of tokens without rollback. We use s1 budget forcing (Muennighoff et al., 2025) for this comparison. Like ReTrail, s1 operates on a single trajectory without an external verifier. It spends additional tokens by extending the current reasoning, whereas ReTrail spends them on rollback and guided regeneration. The comparison therefore fixes the token count and changes how the additional tokens are used.

Protocol. We use Qwen-7B on AIME 2026 and LiveCodeBench with the first seed, the same ReTrail responses as in Table 2. s1 uses the same prompt and sampling settings as ReTrail. We run s1 with two per-problem token targets taken from the ReTrail response to the same problem. The Output target equals the retained output length of ReTrail. The Total target equals its total generation, including tokens deleted by rollback. Let T_j denote the target for problem j. If the model ends its reasoning before generating T_j tokens, s1 removes the end-of-thinking delimiter and appends *Wait* so that reasoning continues. Once T_j tokens have been generated, s1 appends the end-of-thinking delimiter and the model writes its final answer. The total generation of s1 therefore

approximately matches T_j on each problem. s1 deletes no tokens, so its output length equals its total generation. Both methods stop at the retained-output limit B when a response reaches it.

Table 7: **Accuracy at matched total generation on Qwen-7B.** First-seed responses on 30 AIME 2026 and 80 LiveCodeBench problems. s1 forces each response to the retained output length (Output target) or total generation (Total target) of ReTrail on the same problem. Bold and underline mark the best and second-best accuracies within each benchmark; ties share the same marking.

	AIME 2026			LiveCodeBench		
Method	Acc. ↑	Output (k)	Total (k)	Acc. ↑	Output (k)	Total (k)
▶ ReTrail	**56.67**	17.36	18.21	**41.25**	12.11	13.16
s1 (Output target)	<u>50.00</u>	17.44	17.44	<u>36.25</u>	12.18	12.18
s1 (Total target)	43.33	18.41	18.41	<u>36.25</u>	13.23	13.23

Acc.: exact match (AIME) or pass@1 (LiveCodeBench), in percent. Output: mean retained output tokens per problem, in thousands. Total: mean generated tokens per problem, in thousands, including tokens deleted by rollback.

Results. ReTrail achieves the highest accuracy on both benchmarks at matched token counts (Table 7). With the Total target, s1 generates slightly more tokens than ReTrail on both benchmarks. Its accuracy is nevertheless lower by 13.34 points on AIME 2026 and 5.00 points on LiveCodeBench, a difference of four problems on each benchmark. With the Output target, s1 trails ReTrail by 6.67 points on AIME 2026 and 5.00 points on LiveCodeBench. Raising the s1 target from Output to Total lowers its AIME 2026 accuracy from 50.00% to 43.33% and leaves its LiveCodeBench accuracy at 36.25%. In this single-seed comparison on Qwen-7B, additional tokens spent on extending the reasoning therefore do not recover the accuracy of ReTrail.

I Interpreting the Hesitation Signal

Reflection probability and vocabulary uncertainty. Hesitation mass tracks reflection-token probability, whereas entropy measures uncertainty over the full vocabulary. On 40 selected Original Qwen-7B trajectories, their median within-trajectory rank correlation is 0.77 for AIME 2026 and 0.62 for LiveCodeBench (Appendix M.1). The signals often rise together but emphasize different high-value positions. The median overlap between their top 10% of positions is 46.3% on AIME 2026 and 32.8% on LiveCodeBench (Appendix M.2). Tables 3 and 4 test how using the signals to select revision points affects final-answer accuracy.

In the saved AIME example, hesitation mass is near one before *Wait* while entropy is only 0.305 nats. In the coding example, entropy reaches 3.505 nats before *conflicting* while hesitation mass is 7.13×10^{-8}. These examples show how a strong preference for reflection tokens differs from uncertainty about the next token (Appendix M.3).

Earlier positions for regeneration. Appendix M plots detected rises alongside selected earlier positions, connecting the detection signal to the history used for rollback. These retrospective annotations use token-level readings on unchanged Original trajectories; ReTrail operates at reasoning-step boundaries and regenerates from the selected prefix. Table 2 evaluates the resulting complete decoding policies.

I.1 Case Studies: Successful and Unsuccessful Revision

The Qwen-7B cases illustrate how the retained prefix and changes during regeneration affect the final answer. Tables 10–12 and 14 show further ReTrail trajectories that reach the correct answer or pass the saved tests.

Revising a missed constraint. In the arithmetic-sequence problem (Table 13), Original and ReTrail share the first 497 generated tokens. Both initially include negative common differences among the possible divisors of 20 and 30. ReTrail rolls back to the step establishing this divisibility relation. Regeneration then uses the requirement that 24 and 34 follow the first term 4 to restrict the common

difference to positive values. The completed trajectory returns 178, while Original retains the negative cases and answers 32. The retained prefix supplies the correct divisibility relation; regeneration revises which divisors are admissible.

Reconsideration without repairing the omission. In the permutation-counting problem (Table 15), the third rollback retains a prefix that has already concluded 296 from an incomplete enumeration. Regeneration checks whether fixed points are allowed but leaves the list of cycle structures unchanged. The omitted types contribute 100 permutations, so the final answer remains 296 instead of 396. Here the reflection revisits a counting convention while the missing cases remain unexamined. In contrast to the arithmetic example, reconsideration leaves the decisive earlier choice intact.

A local error followed by self-correction. The hemisphere example (Table 16) shows a different outcome. After the third rollback, regeneration briefly reverses the correct inequality $23{,}200 < 40{,}000$. It immediately corrects this comparison and eventually reaches the correct answer 79. This trajectory shows how a local regression and subsequent recovery can occur within the same completed solution.

J Decoding Details and Complete Algorithm

This section gives the exact definitions and execution rules for Section 4. Step segmentation and the reflection-token set are specified in Appendices A and B; parameter values are listed in Appendix D.

J.1 History and rollback candidates

A boundary is active while it remains in the retained trajectory. Boundaries within the C tokens after a rollback are dirty, including those in zero-bias phases of the waveform; all other boundaries are clean. The clean history contains the masses at earlier active, clean boundaries:

$$\mathcal{H}_i = \{m_j : b_j < b_i,\ j \text{ active},\ j \text{ clean}\}. \tag{9}$$

Thus the current mass is compared with earlier values before entering the history. Boundaries generated during the following R-token cooldown are clean and enter the history even though detection remains disabled.

At a detection boundary b_τ, the candidate indices are

$$\mathcal{W}_\tau = \{i : \max(p, b_\tau - W) < b_i < b_\tau,\ i \text{ active and clean}\}, \tag{10}$$

where $p = 0$ before the first rollback and otherwise denotes the previous rollback point. Boundaries within the C-token bias period are excluded from rollback candidates, including its zero-bias phases. Equation 3 selects the latest candidate within ϵ of the maximum hesitation mass m_i. If $\mathcal{W}_\tau$ is empty, the run raises an error and terminates.

J.2 Waveform and recorded mass

Let s count tokens generated since the latest rollback. The bias on reflection-token logits follows

$$\delta(s) = \begin{cases} A\, h_{\lfloor 8s/C \rfloor}, & 0 \le s < C, \\ 0, & s \ge C, \end{cases} \qquad (h_0, \dots, h_7) = \left(1, \tfrac{1}{2}, 0, -\tfrac{1}{2}, -1, -\tfrac{1}{2}, 0, 0\right). \tag{11}$$

The C tokens are divided into eight equal phases, with C a multiple of eight in all evaluated settings. Positive and negative phases provide promotion and suppression, respectively. The last two phases have zero bias but remain part of the dirty interval. The waveform runs once per intervention; before the first rollback and after the C tokens, the bias is zero.

At every boundary b_i, we record m_i from the sampling distribution after logit bias, temperature scaling, and top-p truncation (Equation 1).

J.3 State, execution, and termination

The inference engine must provide the next-token distribution at a boundary, a bias on reflection-token logits, and continuation from a chosen prefix. These operations are available directly in local inference; Appendix C describes the hosted implementation. The local boundary-recognition rule can require a short continuation beyond a boundary (Appendix A), whose generated tokens enter the total-generation count q.

Between passes, ReTrail retains the trajectory y, the generated-token count q including deleted tokens, and the boundary record $\mathcal{Q}$ containing masses and s_i values. It also tracks the previous rollback point p, intervention count n, and number s of tokens generated since rollback. Initially y and $\mathcal{Q}$ are empty, $q = p = n = 0$, and $s = \perp$ denotes that no rollback has occurred. From a retained prefix of length b, continuation allows at most $\min(B - b, L - |x| - b)$ new tokens. The cumulative count q records generation cost and does not determine this allowance.

Each pass generates under the current bias until a waveform phase changes or the cooldown ends. A pass also ends upon generating an end-of-sequence token or reaching the retained-output limit B or context limit L. New boundaries are processed in increasing order. Detection is enabled at b_i exactly when $n < K$, $b_i > p$, and either $s_i = \perp$ or $s_i \ge C + R$. A defined threshold and $m_i > \theta_i$ then trigger rollback at the first qualifying boundary.

On rollback, ReTrail retains $y_{\le b_\star}$, marks deleted boundaries inactive, and sets $p \leftarrow b_\star$, $n \leftarrow n + 1$, and $s \leftarrow 0$. The next pass starts from this prefix. Deleted tokens remain in q, and deleted boundaries

leave the clean history. An end-of-sequence token terminates generation only if it survives this processing. Generation also stops when $|y| = B$ or $|x| + |y| = L$. After intervention K, detection remains disabled while the final waveform completes, subject to these termination conditions. If no rise is detected, the procedure coincides with segmented unbiased decoding. Appendix D lists the implementation checks.

J.4 COMPLETE ALGORITHM

Algorithm 1 combines the rules above and returns one reasoning trajectory.

Algorithm 1 ReTrail

Require: problem x, model π, retained-output token limit B, context limit L; parameters $H_{\min}$, λ, W, ϵ, A, C, R, K
Ensure: one trajectory y

$y \leftarrow \emptyset,\ q, p, n \leftarrow 0,\ s \leftarrow \perp,\ \mathcal{Q} \leftarrow \emptyset$ ▷ $s = \perp$: no rollback yet
while $|y| < B$ and $|x| + |y| < L$ **do**
  choose δ from Eq. 11 ▷ $\delta = 0$ when $s = \perp$
  $(z, e) \leftarrow \textsc{ResumeSegment}(x, y, B - |y|, L, \delta)$ ▷ e: the stretch ended with an end-of-sequence token
  $y \leftarrow y \,\|\, z;\ q \leftarrow q + |z|$
  record s_i for each boundary in z; if $s \neq \perp$, set $s \leftarrow s + |z|$
  rolled $\leftarrow$ false
  for all new boundaries b_i in increasing order **do**
    $m_i \leftarrow \textsc{HesitationMass}(x, y_{\le b_i})$ ▷ from the recorded sampling distribution
    $\mathcal{H}_i \leftarrow \{m_j : j \in \mathcal{Q},\ b_j < b_i,\ j \text{ active and clean}\}$
    define θ_i iff $|\mathcal{H}_i| \geq H_{\min}$ and $\sigma(\mathcal{H}_i) > 0$
    mark i active; mark it dirty iff $s_i \neq \perp$ and $s_i < C$; append i to $\mathcal{Q}$
    enabled $\leftarrow (n < K) \wedge (b_i > p) \wedge (s_i = \perp \vee s_i \geq C + R)$
    if enabled and θ_i defined and $m_i > \theta_i$ **then**
      $\tau \leftarrow i$; form $\mathcal{W}_\tau$ by Eq. 10
      if $\mathcal{W}_\tau = \emptyset$ **then**
        raise an error and terminate this run
      end if
      choose $b_\star$ by Eq. 3
      $y \leftarrow y_{\le b_\star}$; mark the deleted boundaries inactive
      $p \leftarrow b_\star;\ n \leftarrow n + 1;\ s \leftarrow 0;$ rolled $\leftarrow$ true
      break
    end if
  end for
  if rolled **then**
    continue ▷ an end-of-sequence token among the deleted tokens is ignored
  else if e **then**
    break
  end if
end while
return y

K Complete Trigger and Rollback Controls

Table 8 gives the full results behind Section 5.4; the main-text table retains ReTrail (R0), w/ entropy trigger (E1), and w/ random trigger (E2) to compare trigger rules with the rollback rule held fixed.

Table 8: **Complete trigger and rollback controls on Qwen-7B.** First-seed responses on 30 AIME 2026 problems and a fixed 80-problem LiveCodeBench slice. All variants share the same bidirectional waveform and cooldown.

		AIME 2026		LiveCodeBench	
ID	Method	Acc. ↑	Len. ↓	Acc. ↑	Len. ↓
R0	ReTrail	**56.67**	17.36	**41.25**	12.11
E1	w/ entropy trigger	50.00	19.09	37.50	13.58
E2	w/ random trigger	50.00	16.53	37.50	11.58
E3	w/ 512-token rollback w/ entropy trigger	43.33	13.99	36.25	14.43
E4	+ entropy-peak rollback w/ entropy trigger	46.67	17.20	35.00	14.44
E5	w/o rollback w/ random trigger	43.33	17.95	36.25	12.48
E6	+ 512-token rollback w/ random trigger	36.67	15.13	36.25	11.94
E7	w/o rollback	36.67	14.36	35.00	10.58

	AIME 2026			LiveCodeBench		
ID	Total (k)	Deleted (k)	Interventions	Total (k)	Deleted (k)	Interventions
R0	18.21	0.85	2.40	13.16	1.04	1.93
E1	19.77	0.68	2.73	14.19	0.61	2.80
E2	17.26	0.73	2.37	12.39	0.82	2.55
E3	15.12	1.13	2.33	15.23	0.80	1.85
E4	18.84	1.65	2.80	16.10	1.66	2.76
E5	17.95	0.00	2.77	12.48	0.00	2.69
E6	16.38	1.25	2.43	13.25	1.32	2.61
E7	14.36	0.00	2.50	10.58	0.00	2.49

IDs identify the same variants in both panels. Names specify changes from ReTrail; all other components are retained. Acc.: exact match (AIME) or pass@1 (LiveCodeBench), in percent; bold marks the best accuracy. Len.: mean retained output tokens per problem in thousands, as in Table 1. The lower panel reports mean total generated tokens (Total), including deleted tokens, mean deleted tokens (Deleted), and mean actual interventions per problem, including problems with no intervention. Token counts are in thousands; totals and their components are rounded independently. Interventions count bias applications, including those without rollback. The 512-token rollback variants use the nearest eligible boundary. The w/o rollback variants apply the bias at the current detection boundary. The LiveCodeBench slice contains 51 AtCoder and 29 LeetCode problems from March–April 2025 (latest contest: April 6).

The component ablation establishes the benefit of rollback with guided regeneration. Table 8 now tests the signal used to choose its timing and location on Qwen-7B. R0, E1, and E2 compare trigger rules with mass-based rollback held fixed. R0 versus E3 and E2 versus E6 compare rollback rules under hesitation-based and random triggering, respectively. All variants use the same bidirectional waveform and reasoning-step boundaries. They also share the clean-history and eligibility rules, together with the retained-output token limit within each benchmark. The common settings are $H_{\min} = 3$, $W = 2048$, $A = 5$, $C = 304$, $R = 512$, and $K = 3$, as in Appendix D.

Frozen trigger settings. Let H_i be the entropy of the full next-token distribution at boundary b_i, before logit bias, temperature scaling, and top-p truncation. Let $\mu_{H,<i}$ and $\sigma_{H,<i}$ be its mean and population standard deviation over the earlier active, clean boundaries. The entropy trigger and the

random trigger use

$$H_i > \mu_{H,<i} + 2.05\,\sigma_{H,<i}, \qquad p_{\text{random}} = \frac{59}{3254} \approx 0.01813. \tag{12}$$

The entropy threshold requires at least $H_{\min}$ history values and $\sigma_{H,<i} > 0$. The random trigger draws an independent Bernoulli decision with probability p_{random} at each eligible boundary. Calibration uses 30 reasoning trajectories from AIME 2025, selected with sampling seed 42. The entropy coefficient is chosen by matching the candidate alarm count of the hesitation trigger with $\lambda = 1.5$, without using answer correctness. The empirical coefficient 2.04927369 is rounded to 2.05; the random probability uses the corresponding candidate alarm rate. Both settings are fixed across AIME 2026 and LiveCodeBench, with no benchmark-specific tuning. Candidate alarm matching precedes the actual interventions governed by rollback and cooldown.

Rollback choices. Hesitation peaks follow Equation 3. Entropy peaks maximize the recorded entropy over the same candidate boundaries in Equation 10, with ties resolved in favor of the latest boundary. The fixed-distance controls (E3 and E6) select the eligible boundary nearest to $b_\tau - 512$. Variants without rollback apply the waveform at the detection boundary. Each variant generates one response per problem on the same evaluation questions, including problems with no intervention.

Hesitation-based rollback improves accuracy. R0 and E3 retain the hesitation trigger and differ in the rollback rule. Selecting the point by hesitation mass improves accuracy on both tasks. On AIME 2026, accuracy rises from 43.33% to 56.67%, while total generation increases from 15.12k to 18.21k tokens. On LiveCodeBench, accuracy rises from 36.25% to 41.25%, while generation falls from 15.23k to 13.16k tokens. These comparisons support hesitation-based selection of the rollback point over fixed-distance rollback, with additional generation on AIME 2026 and a reduction on LiveCodeBench.

The location rule also helps under random triggering. E2 and E6 in Table 8 use the same random trigger rule and regeneration bias. Replacing fixed-distance rollback with mass-based selection raises AIME 2026 accuracy from 36.67% to 50.00% and LiveCodeBench accuracy from 36.25% to 37.50%. Thus, mass-based location selection improves accuracy under both hesitation-based and random triggering in these comparisons. Section 5.5 tests the restart choice at the same trigger and generation history.

L Fixed-Trigger Restart-Location Protocol

Testing where revision should begin. We fix the first executable hesitation event on each first-seed Original Qwen-7B trajectory and compare where to resume. All alternatives share the decision to intervene and the subsequent reflection bias. They differ in the prefix retained for regeneration, so final-answer accuracy evaluates the usefulness of the restart choice at this shared trigger.

Alternative restart locations. We compare the ReTrail mass sub-spike with the highest-entropy candidate, a random candidate in the same 512-token rollback-distance bin, and the current trigger boundary. The first three use the same eligible boundaries within the preceding 2048 tokens. Entropy supplies an alternative ranking, while distance-bin matching reduces differences in rollback depth. These alternatives are labeled w/ entropy-peak rollback, w/ distance-matched random rollback, and w/o rollback, respectively, in Table 4. The w/o rollback variant applies the bias where the trigger occurs. The hesitation threshold coefficient is 1.5, and entropy is computed after temperature scaling and top-p truncation. All 30 AIME 2026 and 80 LiveCodeBench problems have an eligible event under these rules.

Repeated continuations and generation cost. Each location receives three continuations with matched sampling seeds and a single application of the same bidirectional bias ($A = 5$, $C = 304$). Evaluating the four location rules uses twelve continuations per problem. All continuations enter the accuracy average, with each problem weighted equally.

The original context limit of 131,072 tokens is preserved: a prompt of length P and a retained generated prefix of length b allow at most $131{,}072 - P - b$ new tokens. This allowance determines

the continuation limit at each location. Total generation counts the tokens generated through event detection once per continuation and adds the newly generated tokens. It therefore includes the suffix deleted by the restart.

Table definitions and additional counts. The following definitions and counts accompany Table 4. Acc.: mean final-answer accuracy in percent (AIME exact match; LiveCodeBench pass@1), with each problem weighted equally. Total: mean tokens generated through event detection plus newly generated continuation tokens, including deleted tokens; the shared prefix is counted once per continuation. Counts measure generated output tokens. The current-boundary group removes only the tokens used to recognize the boundary (five on average). The random location equals the ReTrail location on 13.33% of AIME and 18.75% of LiveCodeBench problems. Length-truncated answers remain in the results: 1/1/3/0 on AIME and 1/1/1/1 on LiveCodeBench, in row order.

Hesitation-based selection improves continuation accuracy. The ReTrail sub-spike achieves the highest mean accuracy among the four restart locations on both benchmarks (Table 4). On AIME 2026, it gains 7.78 percentage points over the entropy peak, 10.00 points over the distance-bin random baseline, and 3.33 points over the current trigger boundary. The corresponding gains on LiveCodeBench are 2.09, 2.09, and 2.92 points. With the trigger and regeneration bias held fixed, these results directly support using hesitation mass to select where revision begins.

Accuracy and token cost at the shared trigger. On AIME 2026, the selected location combines the highest accuracy with the lowest total generation at 14.00k tokens. On LiveCodeBench, it uses 13.14k tokens, compared with 12.81k for the distance-bin random location and 12.61k for the current boundary. The latter comparison raises accuracy from 35.00% to 37.92%. The fixed-trigger results agree with the complete-policy controls on the benefit of hesitation-based location selection, with its generation cost measured in both comparisons.

M SIGNALS ALONG ORIGINAL REASONING TRAJECTORIES

Trajectories and recorded reflection tokens. Figures 4–6 show the same 40 Original trajectories with a shared row order and token coordinates. Selection uses completed reasoning length without consulting answer correctness, so these short trajectories provide a qualitative view of the signals. The recorded hesitation mass sums eight token IDs for *Wait*, *But*, *Alternatively*, and *However*, each with and without a leading space. Each trajectory ends immediately before the closing reasoning token; the final answer is excluded.

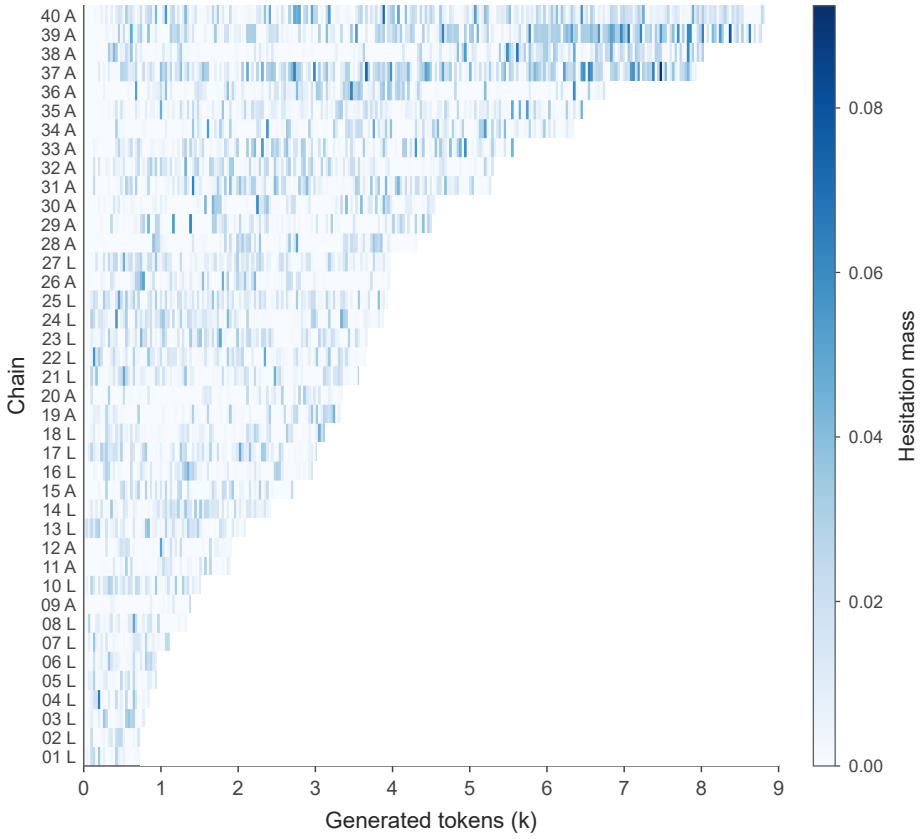

Figure 4: **Hesitation mass during Original decoding.** Rows show reasoning trajectories from DeepSeek-R1-Distill-Qwen-7B on 20 AIME 2026 questions (A) and 20 LiveCodeBench questions (L). We retain the shortest completed run among three seeds per question, then select the 20 shortest runs per task and sort the rows by length. Each cell averages the raw probability mass of the eight recorded reflection tokens over 32 generated tokens. The linear color scale is shared across rows, and white marks positions beyond each trajectory's reasoning end.

Comparing hesitation mass and entropy. Hesitation mass measures probability assigned to the recorded reflection tokens, whereas entropy measures uncertainty over the full vocabulary. Figure 5 uses the same 32-token groups as Figure 4, allowing comparison at matching positions. Each heatmap has its own linear color scale shared across all rows, with no normalization within a trajectory.

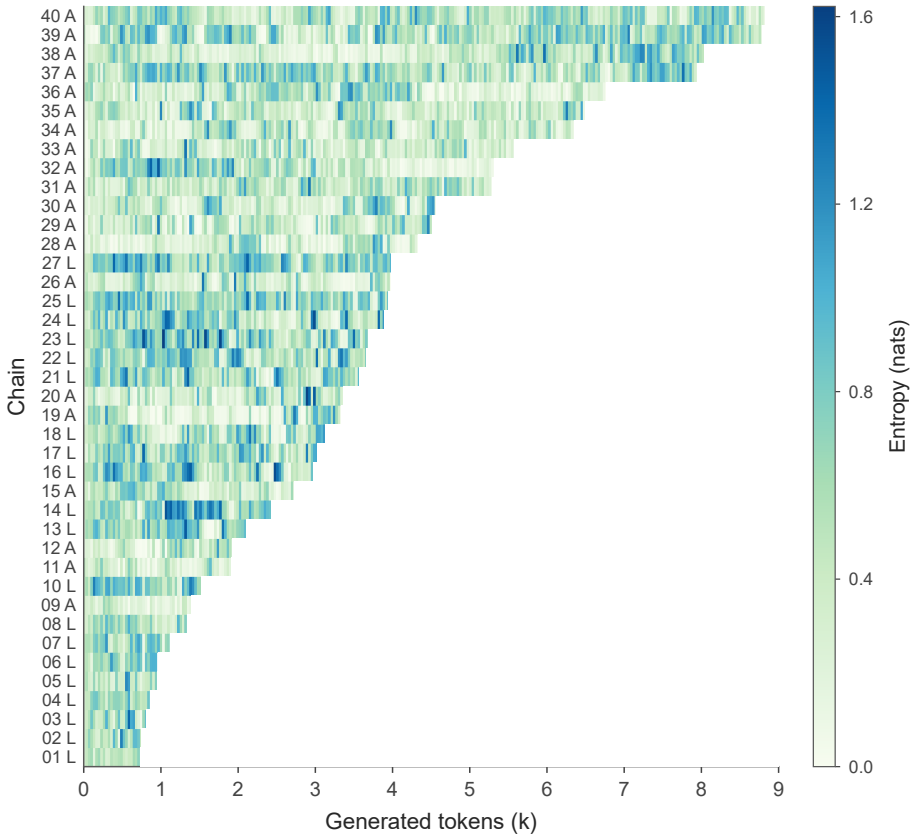

Figure 5: **Next-token entropy during Original decoding.** Full-vocabulary entropy in nats is computed from the raw model distribution before temperature scaling and top-p truncation. Each cell averages 32 generated tokens. Trajectories, row order, and token coordinates match the hesitation-mass heatmap; A denotes AIME 2026 and L denotes LiveCodeBench. The linear color scale is shared across rows.

Retrospective detection and earlier-position selection. Figure 6 marks detected rises as *alarms* and their selected earlier positions as *subspikes*. Detection starts after at least three prior tokens and compares each raw token-level mass with the prior mean plus 1.5 population standard deviations, requiring a positive standard deviation. Subspikes follow Equation 3 with tokens in place of boundaries. Detection then pauses for $C + R = 816$ tokens while history keeps accumulating. The annotation operates at every token without deleting history or applying a probability bias; Section 4 instead operates at reasoning-step boundaries and updates the trajectory through rollback. The marked pairs therefore illustrate positions selected on unchanged Original trajectories, rather than executed ReTrail interventions. Because the early history is close to zero, even a small rise exceeds the threshold, and every first alarm occurs within the first 110 tokens. The other 148 alarms lie a median of 334.5 tokens after their subspikes (Figure 6b). Of these, 117 occur within 100 tokens after the preceding pause ends, so the pause largely sets the spacing between alarms (Figure 6c).

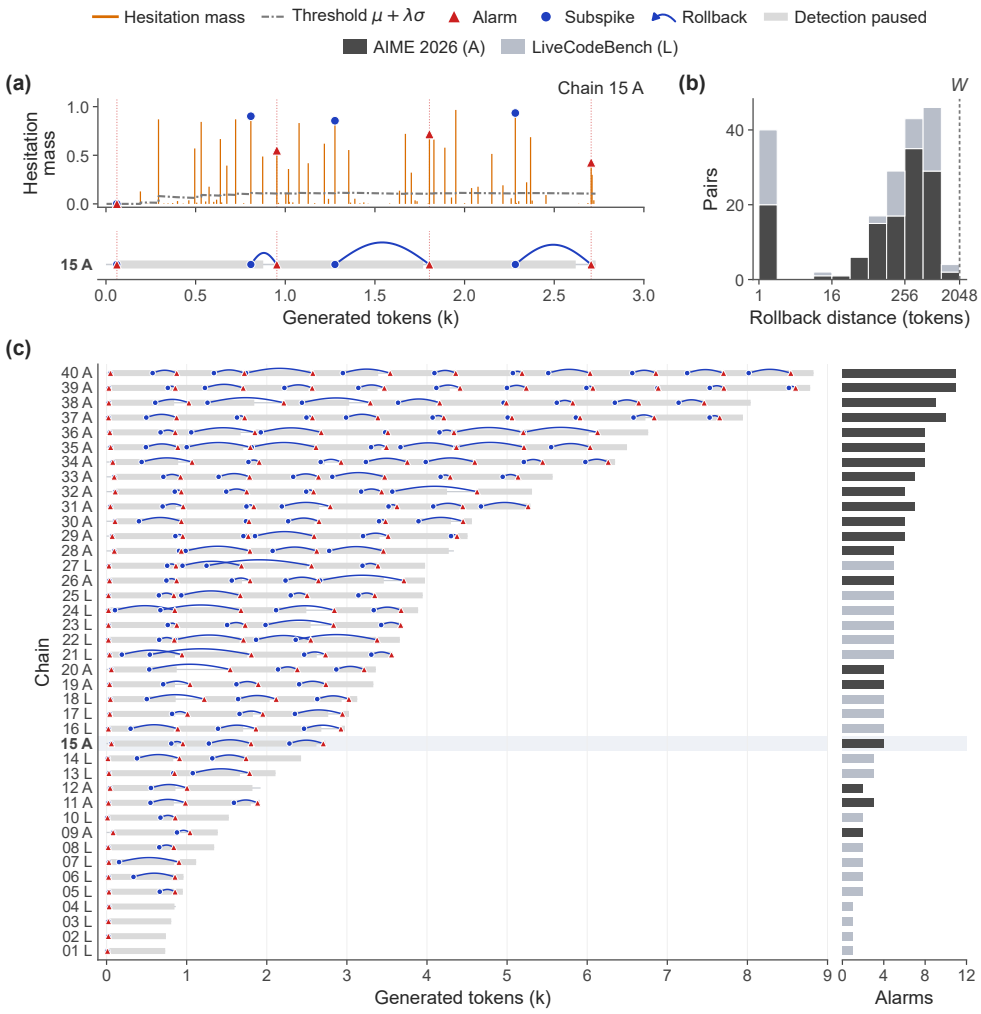

Figure 6: **Alarm and subspike positions on Original trajectories.** (a) Chain 15 (AIME 2026) at token resolution, with raw hesitation mass in orange and the threshold dash-dotted. The strip below enlarges row 15 of panel (c), where blue arcs join each alarm to its subspike and grey bars mark the 816 paused tokens after each alarm. (b) Alarm-to-subspike distance on a base-2 axis, with the look-back W dashed; the bar at 1 holds each trajectory's first alarm. (c) All 40 trajectories in the row order of Figures 4 and 5, with chain 15 highlighted and alarm counts at right. A denotes AIME 2026 and L denotes LiveCodeBench. All pairs use $\lambda = 1.5$ and tie tolerance 0.05, with no cap on the number of alarms.

M.1 JOINT RANKS OF HESITATION MASS AND ENTROPY

The following comparisons use the same 40 Original trajectories as the preceding heatmaps. For each question, we retain the shortest completed reasoning run among three seeds, then select the 20 shortest question representatives per task, without using answer correctness. Hesitation mass uses the recorded reflection-token set described above; entropy uses the full vocabulary. Both signals are read before temperature scaling and top-p truncation.

Figure 7 compares within-trajectory ranks so that each question contributes equally despite differences in reasoning length and signal scale. Median per-question Spearman correlations are 0.77 on AIME 2026 and 0.62 on LiveCodeBench. The signals tend to rise together, but their highest-value positions need not coincide.

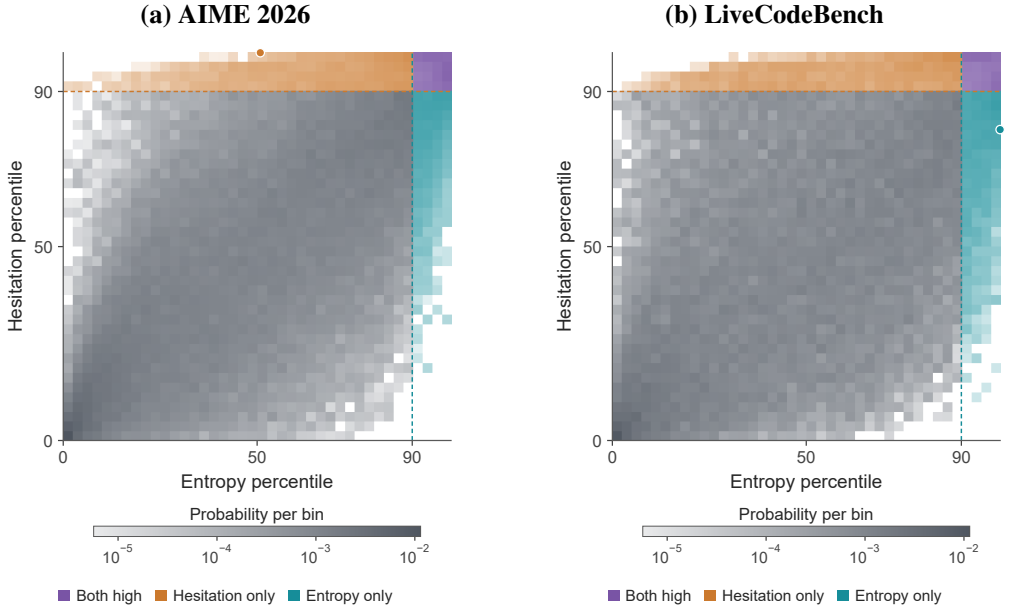

Figure 7: **Joint signal ranks on the selected Original trajectories.** Hesitation mass and next-token entropy are ranked within each trajectory, with average ranks for ties. Histograms give each of the 20 questions per task equal weight and share a logarithmic scale for probability per bin. Dashed lines mark the 90th percentiles. Purple denotes positions high in both signals; orange and cyan denote positions high only in hesitation mass and entropy, respectively. The orange and cyan dots identify the AIME and LiveCodeBench examples in Table 9.

M.2 OVERLAP OF HIGH-VALUE POSITIONS

For each trajectory, we select the top $q\%$ of positions separately under each signal and measure how many positions the two selections share. Each selection contains exactly $k = \lceil nq/100 \rceil$ positions; ties are broken by choosing earlier token positions. At $q = 10$, the median overlap is 46.3% on AIME 2026 and 32.8% on LiveCodeBench (Figure 8). Table 9 shows the saved text at two positions where the signals disagree; Figure 9 shows their local curves.

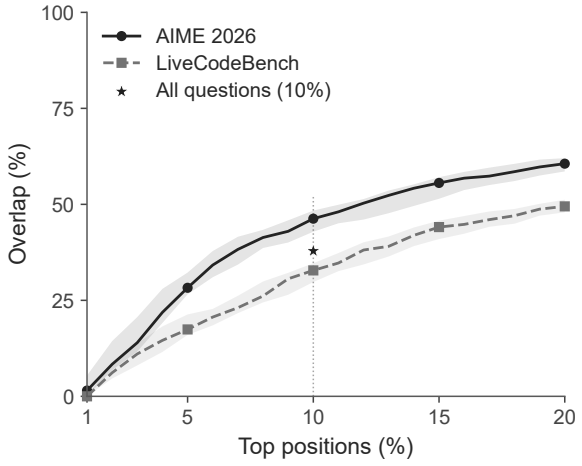

Figure 8: **Overlap between high-value signal positions.** Each signal selects its top $q\%$ of positions within a trajectory. Overlap is the intersection size divided by $k = \lceil nq/100 \rceil$, where n is the reasoning length. Curves show medians over 20 selected questions per task. Shading gives pointwise 95% percentile intervals from 10,000 bootstrap resamples of questions. At the top 10%, median overlap is 46.3% on AIME 2026 and 32.8% on LiveCodeBench; the star marks the median of 37.9% across all 40 questions.

Table 9: **Original text at the two highlighted signal positions.** Excerpts reproduce the saved reasoning text within 32 tokens of each selected position. Bold identifies the next generated token at that position; ellipses mark the excerpt boundaries. The AIME example has hesitation mass in the top 10% and entropy below the 90th percentile; the LiveCodeBench example has the reverse pattern.

Example	Saved reasoning excerpt
AIME 2026 Token 6822	… 9 is 243. So, perhaps the number is 243. Alternatively, maybe 81. But I need to verify. **Wait**, let me think. If I have two conditions, each reducing the count by a factor of 3, then the total number would be 3^7…
LiveCodeBench Token 1364	… can be. So the total number of x is max(0, min_upper - max_lower + 1). But wait, how about when the constraints are **conflicting** for i=0? Because the first element is x, but the first constraint is for i=0: x must be between u_0 and v_…

M.3 LOCAL SIGNALS AROUND THE TEXT EXAMPLES

The AIME example assigns nearly all probability mass to the recorded reflection tokens before generating *Wait*, while full-vocabulary entropy is 0.305 nats. At the LiveCodeBench example, entropy reaches the trajectory maximum of 3.505 nats before *conflicting*, while hesitation mass is only 7.13×10^{-8}. These positions illustrate how a high reflection-token probability and high uncertainty over the vocabulary can occur separately in the saved Original reasoning.

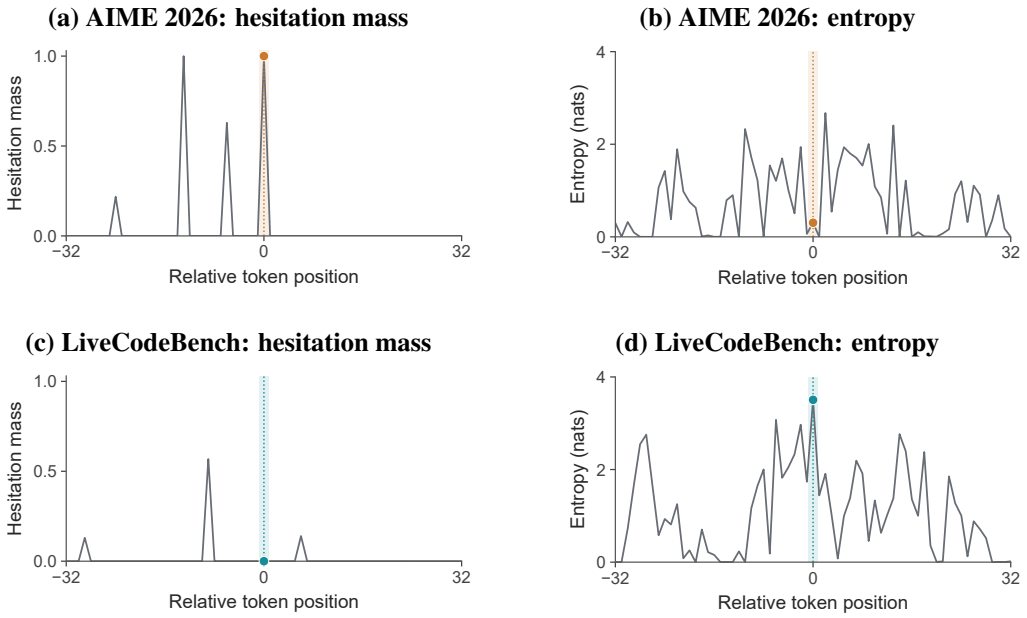

Figure 9: **Local signals at the two text examples.** Curves show raw token-level values over 65 positions, from 32 tokens before to 32 tokens after each highlighted position. Zero is token 6822 for AIME 2026, immediately before *Wait*, and token 1364 for LiveCodeBench, immediately before *conflicting*. Hesitation-mass panels share one vertical scale, and entropy panels share another. Colored dots match Figure 7; the corresponding text appears in Table 9. All curves come from the unchanged Original reasoning records.

N CASE STUDY TABLES

Tables 10–16 give the reasoning excerpts for the cases discussed in Appendix I.1.

Q1: Let N be the number of positive divisors of 17017^{17} that leave remainder 5 when divided by 12. Find N modulo 1000.

Before:	Now, let's factorize 221. Hmm, 221 divided by 13 is 17, because 13×17 is 221. So, putting it all together, 17017 = $7 \times 11 \times 13 \times 17$. So, all of these are prime numbers, right? 7, 11, 13, 17—yes, primes. ~~Therefore, 17017^{17} is equal to $7^{17} \times 11^{17} \times 13^{17} \times 17^{17}$. So, when we talk about the divisors of this number, each divisor will be of the form $7^a \times 11^b \times 13^c \times 17^d$, where $0 \leq a, b, c, d \leq 17$. So, each exponent can range from 0 to 17.~~
After:	**Wait** Wait, hold on. 17017 is equal to $7 \times 11 \times 13 \times 17$? Let me check that multiplication again. 7×11 is 77, 77×13 is 1001, 1001×17 is 17017. Yeah, that's correct. So, 17017 factors into four distinct primes: 7, 11, 13, and 17. Therefore, 17017^{17} is equal to $7^{17} \times 11^{17} \times 13^{17} \times 17^{17}$. So, any positive integer divisor of this number will be of the form $7^a \times 11^b \times 13^c \times 17^d$, where $0 \leq a, b, c, d \leq 17$. So, that's the general form of the divisors. ...
Final:	We need the product $7^a \times 11^b \times 13^c \times 17^d \equiv 5 \mod 12$. The residues of each prime power modulo 12 are: - $7^a \mod 12$ is 7 if a is odd, 1 if a is even. - $11^b \mod 12$ is 11 if b is odd, 1 if b is even. - $13^c \mod 12$ is always 1. - $17^d \mod 12$ is 5 if d is odd, 1 if d is even. We need $x \times y \times z \equiv 5 \mod 12$ where x, y, z are the residues of $7^a, 11^b, 17^d$ respectively. We find that the valid combinations are: 1. $x = 1$, $y = 1$, $z = 5$ (i.e., a even, b even, d odd) 2. $x = 7$, $y = 11$, $z = 1$ (i.e., a odd, b odd, d even) For each case, the number of choices for a, b, d is: - Case 1: 9 choices for a, 9 choices for b, 9 choices for d (total 729) - Case 2: 9 choices for a, 9 choices for b, 9 choices for d (total 729) Thus, the total number of valid (a, b, d) tuples is $729 + 729 = 1458$. For each tuple, c can be any value from 0 to 17, giving 18 choices. Therefore, the total number N is $1458 \times 18 = 26244$. Finally, we find $N \mod 1000$: $26244 \mod 1000 = 244$ Thus, the remainder when N is divided by 1000 is $\underline{\mathbf{244}}$. **Result: Correct.**

Table 10: Divisor counting with DeepSeek-R1-Distill-Qwen-7B (AIME 2026, aime26-7, sample s1). After detecting a rise in hesitation mass, ReTrail rolls back to the underlined prefix and regenerates from there. During regeneration, ReTrail first raises and then lowers the logits of reflection tokens. Strikethrough marks the discarded continuation. Blue highlights the first reflection token generated after rollback; red highlights other generated reflection tokens. The display pairs consecutive passages at the first rollback with the final response. The final response counts both parity cases and gives 244.

Q2: Start at position n in a line. Pay cost[j] to swap with a person ahead; swap with a person behind for free. Find the minimum cost to each position.

Before:	Okay, I need to solve this problem where I have to find the minimum total cost to reach each position in a line based on the given cost array. Let's think through this step by step. So, the problem says that I'm initially at position n, which is the end of the line with n+1 people (numbered 0 to n). I want to move to each position from 0 to n, and for each position i, I need to calculate the minimum cost to get there. ~~The way swapping works is a bit tricky. If someone is in front of me, I have to pay their cost to swap with them. But if someone is behind me, they can swap with me for free. So, the strategy must involve figuring out the optimal path to each position with the least cost. Let me look at the examples to get a better idea. In Example 1, the cost array is [5,3,4,1,3,2]. The output is [5,3,3,1,1,1]. So, for i=0, you pay 5. For i=1, you pay 3. For i=2, you can go from 1 to 2 for free because 2 is behind you. Then for i=3, you pay 1. And for i=4 and 5, you can swap with 3 for 1, then for 4 and 5, it's free.~~
After:	**Wait** Hmm, the cost array has size n, and the line has n+1 people. So for each position i in 0 to n, I need to find the minimum cost to reach there. The movement rules are: if someone is in front of me, I have to pay their cost[i] to swap. If they are behind, I can swap for free. So when I'm trying to get to a new position, I can either swap with someone in front (paying their cost) or swap with someone behind for free. But how do I model this? It feels like a dynamic programming problem, where I keep track of the minimum cost to reach each position. Let's think about the initial state. I start at position n, and the cost to get there is 0 because I'm already there. But wait, the output array is of size n, which suggests that the positions are 0 to n-1? Or maybe the problem statement has a typo. Wait, looking back: the output is size n, where each i is the position. So for example, in example 1, cost has length 6, output is length 6. So the positions are 0 to 5. ...
Final:	The problem can be efficiently solved using Dijkstra's algorithm, which is typically used for finding the shortest path in a graph with non-negative weights. Here, each position in the line is treated as a node in a graph, and each swap operation is treated as an edge with a specific weight. 1. **Graph Representation**: Each position (node) can be connected to other positions based on the cost of swaps. Specifically, from each position i, we can: - Move to any position j where $j < i$ with a cost equal to cost[j] (since swaps to the front are costly). - Move to any position j where $j > i$ with a cost of 0 (since swaps to the back are free). 2. **Dijkstra's Algorithm**: We use Dijkstra's algorithm to find the shortest path from the starting position n (end of the line) to each other position. This algorithm is suitable because all edge weights are non-negative. **Result: Passed.**

Table 11: Minimum-cost swapping with DeepSeek-R1-Distill-Qwen-7B (LiveCodeBench, problem 3832, sample s1). After detecting a rise in hesitation mass, ReTrail rolls back to the underlined prefix and regenerates from there. During regeneration, ReTrail first raises and then lowers the logits of reflection tokens. Strikethrough marks the discarded continuation. Blue highlights the first reflection token generated after rollback; red highlights other generated reflection tokens. The display pairs consecutive passages at the first rollback with the final response. The final program implements the directed swap costs and passes the saved tests.

Q3: For an undirected graph with N $\leq$ 10, find the minimum XOR of edge labels over all simple paths from vertex 1 to vertex N. A simple path visits no vertex twice.

Before:	Okay, I have to solve this programming problem. Let me read the problem statement carefully. So, the problem is about finding the minimum XOR of edge labels along all simple paths from vertex 1 to vertex N in a given undirected graph. A simple path is one that doesn't visit the same vertex more than once. Hmm. The constraints are small: N is up to 10, which is manageable. M can be up to N*(N-1)/2, which is like 45 when N is 10. So the graph can be dense but not too big. ~~The key here is that since N is small, we can potentially explore all possible simple paths from 1 to N, compute the XOR for each, and find the minimum.~~
After:	**Wait** So the approach I need to take is to consider all possible simple paths from 1 to N and compute the XOR of their edges, then find the minimum. But how to efficiently find all these paths without duplicates? Let me think about the possible approaches. Since N is small (up to 10), a brute-force approach might be feasible. We can perform a depth-first search (DFS) starting from node 1, exploring all possible paths until we reach node N, keeping track of the current XOR value. But wait, for each path, we need to compute the XOR of the edges along the path. So, for each path, when we reach N, we can compare its XOR with the current minimum and update accordingly. But how to avoid revisiting nodes in the same path to prevent cycles? Because the path is simple, so each node can be visited at most once. So, the plan is: 1. Use DFS to explore all possible paths from 1 to N. 2. For each path, compute the XOR of the edge labels. 3. Keep track of the minimum XOR found. But to implement this, I need to represent the graph in a way that allows me to traverse it without revisiting nodes. Let's think about the data structures. I can represent the graph as an adjacency list. Each node will have a list of connected nodes along with the edge weight. So for each node u, I'll have a list of tuples (v, w), where v is a neighbor and w is the weight of the edge. Then, during DFS, for each node, I'll keep track of the visited nodes in the current path. So, for each step, when moving from current node u to v, I check if v is not in the visited set. If it's not, I add it to the visited set and proceed. Wait, but the path is from 1 to N, so the visited set starts with 1, and each step adds a new node. ...
Final:	1. **Graph Representation**: Represent the graph using an adjacency list where each node points to its neighbors along with the edge weights. 2. **Depth-First Search (DFS)**: Use DFS to explore all possible paths from vertex 1 to vertex N. Track the current XOR value as we traverse each path. 3. **Visited Nodes**: Use a bitmask to keep track of visited nodes to ensure that each path is simple (no repeated nodes). 4. **Recursive Function**: Define a recursive function that performs the DFS. For each node, it explores all its neighbors, updates the visited bitmask, and computes the current XOR value. If a path reaches vertex N, it updates the minimum XOR value found. **Result: Passed.**

Table 12: Minimum-XOR paths with DeepSeek-R1-Distill-Qwen-7B (LiveCodeBench, abc396_d, sample s2). After detecting a rise in hesitation mass, ReTrail rolls back to the underlined prefix and regenerates from there. During regeneration, ReTrail first raises and then lowers the logits of reflection tokens. Strikethrough marks the discarded continuation. Blue highlights the first reflection token generated after rollback; red highlights other generated reflection tokens. The display pairs consecutive passages at the first rollback with the final response. The final program tracks visited vertices and passes the saved tests.

Q4: Find the sum of the 10th terms of all arithmetic sequences of integers that have first term equal to 4 and include both 24 and 34 as terms.

Before:	… So, both 20 and 30 are multiples of d. Therefore, d must be a common divisor of 20 and 30. ~~Let me find the common divisors of 20 and 30. First, find the greatest common divisor (GCD) of 20 and 30. The factors of 20 are 1, 2, 4, 5, 10, 20. The factors of 30 are 1, 2, 3, 5, 6, 10, 15, 30. The common factors are 1, 2, 5, 10. So, the GCD is 10. Therefore, the possible common differences d are the divisors of 10, which are 1, 2, 5, 10, and also their negatives: -1, -2, -5, -10. So, in total, d can be ±1, ±2, ±5, ±10. Wait, hold on.~~
After:	**Wait**, so d is a common divisor of 20 and 30. But since the sequences are of integers, d must be an integer. But also, the problem says "arithmetic sequences of integers", so yes, d must be an integer. But hold on, d could be positive or negative, but since the terms are increasing from 4 to 24 and 34, which are both larger than 4, so d must be positive. So, we can just consider positive divisors. …
Final:	For each value of d, we calculate the 10th term: - For $d = 1$: $4 + 9 \cdot 1 = 13$ - For $d = 2$: $4 + 9 \cdot 2 = 22$ - For $d = 5$: $4 + 9 \cdot 5 = 49$ - For $d = 10$: $4 + 9 \cdot 10 = 94$ Summing these 10th terms: $13 + 22 + 49 + 94 = 178$ Thus, the sum of the 10th terms is $\boxed{178}$. **Result: Correct.**
Original:	… Thus, d must be a common divisor of 20 and 30. The common divisors of 20 and 30 are ±1, ±2, ±5, ±10. Therefore, the possible values of d are 1, 2, 5, 10, -1, -2, -5, -10. … Adding the results: $178 + (-146) = 32$ … $\boxed{32}$ **Result: Incorrect. Reference answer: 178.**

Table 13: Arithmetic sequences with DeepSeek-R1-Distill-Qwen-7B (AIME 2026, aime26-15, sample s0). Original and ReTrail share the first 497 generated tokens. The excerpts show the first of two rollbacks by ReTrail. The underlined text ends the retained prefix; strikethrough marks the discarded continuation. Blue highlights the first reflection token generated after rollback; red highlights other generated reflection tokens. Regeneration excludes negative common differences and the completed trajectory returns 178. Original retains the negative cases and answers 32. Ellipses mark omitted passages.

Q5: Given an integer array properties with n rows, connect distinct nodes i and j when their rows share at least k distinct integers. Return the number of connected components.

Before:	… First, I need to understand what the graph looks like. Each index i in the properties array corresponds to a node. So if there are n elements in properties, there are n nodes. Now, there's an edge between node i and node j if the intersection of properties[i] and properties[j] is at least k. Wait, but the condition is for i and j where i $\neq$ j. ~~Wait, the function intersect(a, b) counts the number of distinct integers common to both a and b. So for each pair of nodes i and j, I need to calculate how many common elements they have, and if that's $\geq$k, I add an edge between them. Once the graph is built, I have to find the number of connected components. A connected component is a subgraph where every node is reachable from every other node in the same component. So the steps I need to take are: 1. For every pair of nodes (i, j) where i $<$ j, calculate the intersection size. 2. If the intersection is $\geq$k, add an edge between i and j. 3. Once the graph is built, perform a traversal (like BFS or DFS) to count connected components.~~ …
After:	**Wait**, the function intersect(a, b) counts the number of distinct integers common to both a and b. So for each pair of nodes i and j, I need to check if the size of their intersection is $\geq$k. If yes, add an edge between them. But wait, the graph is undirected, so edges are bidirectional. But for the purposes of connected components, it doesn't matter as long as the adjacency is correctly represented. So the steps I need to take are: 1. For each pair of nodes (i, j) where i $<$ j, check if their properties' intersection is $\geq$k. If yes, mark them as connected. 2. Once the graph is built, find the number of connected components. So how do I represent this graph? Well, one way is to use a Union-Find (Disjoint Set Union) data structure. Because Union-Find is efficient for finding connected components, especially when the graph is built dynamically. Let me think about how to model this. For each node, I can find all the nodes it's connected to, and then union them. Then, each connected component will be a set in the Union-Find structure. …
Final:	To efficiently determine the connected components, we use the Union-Find (Disjoint Set Union) data structure. This helps in dynamically managing and merging sets of nodes as edges are added. - Convert each row of the properties array into a set for efficient intersection operations. - Use the Union-Find structure to union nodes based on the intersection condition. - After processing all pairs, count the number of unique roots in the Union-Find structure, which gives the number of connected components. **Result: Passed.**

Table 14: Connected components with DeepSeek-R1-Distill-Qwen-7B (LiveCodeBench, problem 3809, sample s0). The excerpts show the first of three rollbacks by ReTrail. The underlined text ends the retained prefix; strikethrough marks the discarded continuation. Blue highlights the first reflection token generated after rollback; red highlights other generated reflection tokens. Regeneration changes the plan from graph traversal to Union-Find. Both are valid approaches to counting connected components. The final program uses Union-Find and passes the saved tests. Ellipses mark omitted passages.

Q6: Find the number of functions π mapping $A = \{1, 2, 3, 4, 5, 6\}$ onto itself such that $\pi^6(a) = a$ for every $a \in A$.

Before:	… So, adding up: 120 + 40 = 160; 160 + 120 = 280; 280 +15=295; 295 +1=296. So, 296 is the total number of such permutations. ~~But wait, let me think again. The problem is about functions π mapping A onto A, which are permutations, such that π^6 is the identity. So, yes, it's exactly the permutations whose orders divide 6. So, the count is 296.~~ …
After:	**Wait**, but I have a doubt. Wait, in the cycle structure (3,2,1), is the fixed point considered as a 1-cycle? But in the cycle structure formula, when we calculate the number of permutations with a specific cycle type, we include all cycles, including fixed points. But in the problem statement, it's a function π mapping A onto A, which is a permutation, so fixed points are allowed. So, I think the calculation is correct. Alternatively, another way to compute this is to note that the number of elements of order dividing 6 in S6 is equal to the number of permutations whose order divides 6. Which is exactly what we computed: 296. So, yeah, 296 should be the correct answer. …
Final:	The possible cycle structures of such permutations are: 1. A single 6-cycle. 2. Two disjoint 3-cycles. 3. A 3-cycle, a 2-cycle, and a fixed point. 4. Three disjoint 2-cycles. 5. All fixed points (the identity permutation). … Summing these, we get the total number of permutations: $120 + 40 + 120 + 15 + 1 = 296$ Thus, the number of such functions π is $\boxed{296}$. **Result: Incorrect. Reference answer: 396.**

Table 15: Permutation counting with DeepSeek-R1-Distill-Qwen-7B (AIME 2026, aime26-6, sample s0). The excerpts show the third and final rollback by ReTrail. The underlined text ends the retained prefix; strikethrough marks the discarded continuation. Blue highlights the first reflection token generated after rollback; red highlights other generated reflection tokens. The model rechecks whether fixed points are allowed but retains an incomplete list of cycle structures. The missing types $(3, 1, 1, 1)$, $(2, 2, 1, 1)$, and $(2, 1, 1, 1, 1)$ contribute 100 permutations. The final answer remains 296 instead of 396. Ellipses mark omitted passages.

Q7: A hemisphere with radius 200 sits on top of a horizontal circular disk with radius 200, and the hemisphere and disk have the same center. Let $\mathcal{T}$ be the region of points P in the disk such that a sphere of radius 42 can be placed on top of the disk at P and lie completely inside the hemisphere. The area of $\mathcal{T}$ divided by the area of the disk is $\frac{p}{q}$, where p and q are relatively prime positive integers. Find $p+q$.

Before:	… So, the region T is the set of points (x, y, 0) on the disk (so $x^2+y^2 \leq 200^2$) such that $x^2+y^2 \leq 23200$. ~~Wait, 23200 is less than 200^2, which is 40000. So, the region T is a circle with radius sqrt(23200). Let me compute sqrt(23200):~~ …
After:	**Wait**, but 23200 is greater than 200^2 (which is 40000). Wait, no, 23200 is 232*100, which is 23200, and 200^2 is 40000, so 23200 is less than 40000. So, $x^2+y^2 \leq 23200$ is a smaller circle inside the disk. …
Final:	… The ratio of the areas is $\frac{23200}{40000} = \frac{29}{50}$. Thus, the fraction is $\frac{29}{50}$, and $p+q = 29+50 = 79$. $\boxed{79}$ **Result: Correct.**

Table 16: Hemisphere geometry with DeepSeek-R1-Distill-Qwen-7B (AIME 2026, aime26-2, sample s0). The excerpts show the third and final rollback by ReTrail. The underlined text ends the retained prefix; strikethrough marks the discarded continuation. Blue highlights the first reflection token generated after rollback; red highlights the subsequent reflection token. The deleted passage correctly compares 23200 with 40000. Regeneration reverses this comparison, immediately corrects it, and eventually returns the correct answer 79. Ellipses mark omitted passages.