STATEFUL REASONING VIA INSIGHT REPLAY

Anonymous authors
Paper under double-blind review

ABSTRACT

Chain-of-Thought (CoT) reasoning has become a foundation for eliciting multi-step reasoning in large language models, but recent studies show that its benefits do not scale monotonically with chain length: while longer CoT generally enables a model to tackle harder problems, on a given problem, accuracy typically increases with CoT length up to a point, after which it declines. We identify a major cause of this phenomenon: as the CoT grows, the model's attention to critical insights produced earlier in the trace gradually weakens, making those insights progressively less accessible when they are most needed. Therefore, we propose **InsightReplay**, a stateful reasoning approach in which the model periodically extracts critical insights from its reasoning trace and replays them near the active generation frontier, keeping them accessible as the reasoning scales. Extensive experiments on a $\mathbf{2} \times \mathbf{3} \times \mathbf{4}$ benchmark grid, covering model scales $\{8\text{B}, 30\text{B}\}$, model families $\{\text{Qwen3.5}, \text{DeepSeek-R1-Distill-Qwen}, \text{Gemma-4}\}$, and reasoning benchmarks $\{\text{AIME}, \text{HMMT}, \text{GPQA Diamond}, \text{LiveCodeBench v5}\}$, show that 3-round InsightReplay yields accuracy gains across **all 24 settings**, with a macro-averaged improvement of $+\mathbf{1.65}$ points over standard CoT, and a largest single-setting gain of $+\mathbf{9.2}$ points on R1-Distill-32B's LiveCodeBench v5 subset. Our results suggest that the effectiveness of test-time scaling depends not only on how much a model reasons, but also on whether critical intermediate insights remain accessible throughout long reasoning trajectories.

1 INTRODUCTION

Chain-of-Thought (CoT) reasoning is a key mechanism for eliciting reasoning in large language models. By explicitly generating intermediate steps, CoT decomposes difficult objectives into smaller sub-problems and has become a foundation for test-time scaling: increasing reasoning-token budgets often improves performance on difficult tasks [1, 2, 3, 4].

However, recent work shows that the benefits of additional reasoning tokens do not scale monotonically. On fixed-difficulty problems, accuracy typically follows an inverted-U relationship with CoT length (see Figure 1): performance first improves as the model reasons longer, reaches a peak, and then declines as the trajectory becomes excessively long [5, 6, 7]. Harder problems shift this peak rightward, requiring longer reasoning before degradation appears.

What drives this decline? Prior work has identified that not all reasoning tokens contribute equally: a small subset of sentences disproportionately determines the final answer, and certain attention heads consistently focus on them [8]. We complement this line of work by asking a different question: *how does the model's access to such critical insights evolve as the reasoning chain grows?*

In Section 2.2, we directly measure the pre-softmax attention from answer tokens to critical insights as a function of token distance, and find that it decays monotonically. As a result, later reasoning is conducted with progressively degraded access to the very intermediate conclusions that matter most. This observation suggests a broader view of test-time scaling: increasing reasoning depth alone is insufficient unless important reasoning state can also remain accessible over time.

This echoes a challenge: preserving long-range information across time. LSTM [9] addressed this by introducing an explicit memory cell that persists and is continually updated as the sequence unfolds. A natural question therefore arises for long reasoning chains: *can we equip the growing reasoning chain with an analogous mechanism that keeps critical insights accessible over time?*

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.

To answer the question, we introduce *stateful reasoning* for CoT, where the reasoning state at any point is the cumulative set of *insights* the model has generated so far, compressed abstractions of prior reasoning. We instantiate this with **InsightReplay**, which periodically generates insights and replays them near the generation frontier. Keeping insights close to the decoding position preserves the model's attention to them. This mitigates the attention decay that critical intermediate content suffers in long trajectories, raising the achievable peak accuracy and expanding the regime in which extended reasoning translates into higher accuracy.

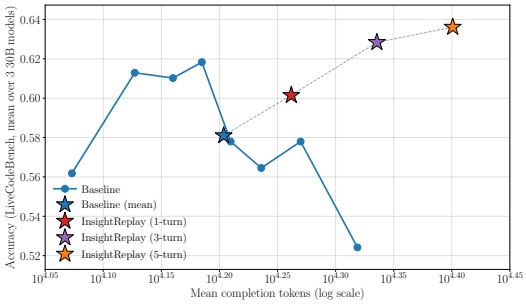


Figure 1: Accuracy vs. mean tokens for Baseline (standard CoT) and InsightReplay (1, 3, 5 replay rounds), averaged over three 30B-tier models on the LiveCodeBench v5 subset. Each star reports the overall mean accuracy.

Figure 1 previews this property. The Baseline curve exhibits the characteristic inverted-U pattern: accuracy first rises with CoT length, peaks at around 15K tokens, and then declines as the chain grows further. InsightReplay behaves differently. It operates precisely in the long CoT regime where additional reasoning already degrades the Baseline, yet each additional replay round (1, 3, 5) achieves progressively higher mean accuracy. By 3 replay rounds, InsightReplay's mean accuracy already exceeds the Baseline's peak; by 5 rounds, performance continues to improve further. In effect, InsightReplay turns the Baseline's degradation regime into a continued-growth regime, extending the range of token budgets over which additional reasoning remains productive.

The main contributions of this paper are as follows:

- We empirically study critical insights in CoT and identify two key properties: **(I)** critical insights and the surrounding reasoning trace are complementary; **(II)** attention to critical insights decays as CoT grows, making them progressively less accessible to the model.
- Building on these observations, we propose **InsightReplay**, a stateful reasoning approach that periodically extracts critical insights and replays them near the active generation frontier. We further provide a theoretical analysis showing that InsightReplay can shift the optimal reasoning length rightward and raise the achievable peak accuracy.
- We validate InsightReplay in a purely inference-time setting and show that replaying critical insights yields consistent improvements across a broad range of models and reasoning benchmarks, without any additional training.
- We further reinforce this reasoning pattern through post-training, demonstrating that InsightReplay improves both training stability and reasoning performance over vanilla CoT under same training settings.

2 Properties of Critical Insights in Long CoT

2.1 Critical Insights and the Reasoning Trace are Complementary

Related Work. Recent work provides evidence that information in CoT is not uniformly distributed: *compression methods* show that many reasoning tokens can be pruned with minimal accuracy loss [10, 11, 12, 13]; *summarize-and-restart methods* first generate a reasoning segment, then summarize it into a compact form, discard the original segment, and resume thinking from the summary [14, 15, 16]; and *importance-attribution methods* identify a small number of sentences or tokens that disproportionately influence the final answer [8, 17]. Appendix L.2 (Table 8) compares InsightReplay with context management, compression, self-reflection, and retrieval-based methods. What remains underexplored is the relationship between the original insights produced within the CoT and the surrounding reasoning trace that produced them. We seek to quantify whether the two are substitutable or complementary.

Experimental Setting. Using `Qwen3-8B` as the base model, we generate thinking traces for AIME problems [18]. We then use the same model to extract 5–7 key insights from each trace. We construct seven conditions with identical chat templates and answer formats, differing only in the content inside the thinking tag: **(A)** original full thinking; **(B)** repeated question + insights; **(C)** insights only; **(D)** full thinking + repeated question + insights; **(E)** full thinking + insights; **(F)** random tokens + repeated question + insights; **(G)** random tokens + insights. For each condition, we compute the answer probability $P(\text{ans}) := P_\theta(a_{\text{gt}} \mid x) = \prod_t P_\theta(a_{\text{gt}}^{(t)} \mid x, a_{\text{gt}}^{(<t)})$ via a single forward pass on `Qwen3-8B`, where x is the prompt prefix up to and including `\boxed{` and a_{gt} are the ground-truth answer tokens; full details are in Appendix A.

Table 1: Effect of thinking content on answer probability $P(\text{ans})$ on 60 AIME problems with `Qwen3-8B`. ✓ = present, – = absent, ∅ = random tokens length-matched to CoT reasoning trace.

Conditions	Content inside `<think>`			Tokens	$P(\text{ans})$	P/Tok (‰)
	CoT reasoning trace	Repeated Q	Insights			
A	✓	–	–	16,731	0.512	0.031
B	–	✓	✓	378	0.273	0.722
C	–	–	✓	236	0.387	**1.640**
D	✓	✓	✓	17,109	**0.557**	0.033
E	✓	–	✓	16,967	0.545	0.032
F	∅	✓	✓	17,109	0.131	0.008
G	∅	–	✓	16,967	0.127	0.007

Results & Conclusion. Table 1 reveals four observations: **(I) Insights carry concentrated signal.** Condition C retains 75.6% of the answer probability with only 1.4% of the tokens, indicating that the core reasoning content distills into a small set of critical insights. **(II) Insights provide additive value on top of the reasoning trace.** Replaying insights on top of the original reasoning trace further improves answer probability from 0.512 to 0.545–0.557, demonstrating that even when the reasoning trace is already present, re-exposing the model to its own insights yields measurable gains. **(III) Bad reasoning trace actively harms inference.** As an extreme case, F and G replace the entire CoT reasoning trace with length-matched random tokens, dropping answer probability to 0.127–0.131, which is below the insight-only conditions B (0.273) and C (0.387). **(IV) Repeating the question helps only when paired with a reasoning trace.** The repeated question helps with a real reasoning trace (D vs. E), but hurts substantially without any reasoning trace (B vs. C).

2.2 Critical Insights Become Less Accessible as Reasoning Chains Grow

Related Work. While prior work has identified important reasoning steps and the heads that attend to them [8], and while positional encodings such as RoPE [19] are known to bias attention toward nearby tokens, empirical attention patterns are ultimately shaped by learned query-key interactions rather than positional geometry alone [20]. Whether attention to critical insights actually decays with distance therefore requires direct measurement. We take critical insights as the unit of analysis and measure how attention from answer tokens to these insights varies with token distance.

Experimental Setting. On AIME with `Qwen3-8B` and `Gemma-4-31B-it`, the model first generates a CoT trace, after which `Qwen3.5-397B-A17B` identifies 5–7 critical insights, each localized to a token span c ending at position p_e. We use a stronger extractor here because attention measurement requires precise token boundaries that smaller models do not produce reliably. To isolate the effect of distance, we insert semantically neutral filler tokens at sentence boundaries between the insight and the answer. Each filler unit is `"\n "` (newline + space), chosen for its minimal semantic impact. Let p_a denote the answer position, so the baseline distance is $d = p_a - p_e$. We sweep an insertion ratio $\rho \in \{0, 0.1, 0.2, 0.3, 0.4\}$, inserting $N = \lfloor \rho d \rfloor$ filler units total, distributed uniformly at random across the sentence boundaries between insight and answer. The resulting total distance is $(1 + \rho)d$ regardless of placement. We use one sample at $\rho = 0$ and 10 random allocations at each non-zero ρ, yielding 41 variants per insight. Since the insight and all preceding tokens are untouched, any change in attention is primarily attributable to distance. For each variant, we

compute the per-layer **pre-softmax** attention score from the answer token to the insight span:

$$s^{(\ell)}(c, a) = \frac{1}{|H|} \sum_{h} \frac{1}{|c|} \sum_{t \in c} \frac{\mathbf{q}_{p_a}^{(\ell,h)} \cdot \mathbf{k}_t^{(\ell,h)}}{\sqrt{d_{\text{head}}}},$$

where c denotes the set of token positions covered by the insight span (with $|c|$ its size and t ranging over its elements), ℓ is the layer index, h indexes attention heads, $\mathbf{q}$ and $\mathbf{k}$ are post-RoPE query and key vectors, $|H|$ is the number of heads, and d_{head} is the per-head dimension. We use pre-softmax scores rather than post-softmax weights because softmax normalization over the full preceding context mechanically squashes weights as context length grows, confounding the effect of distance with that of sequence length. Full implementation details are in Appendix B.

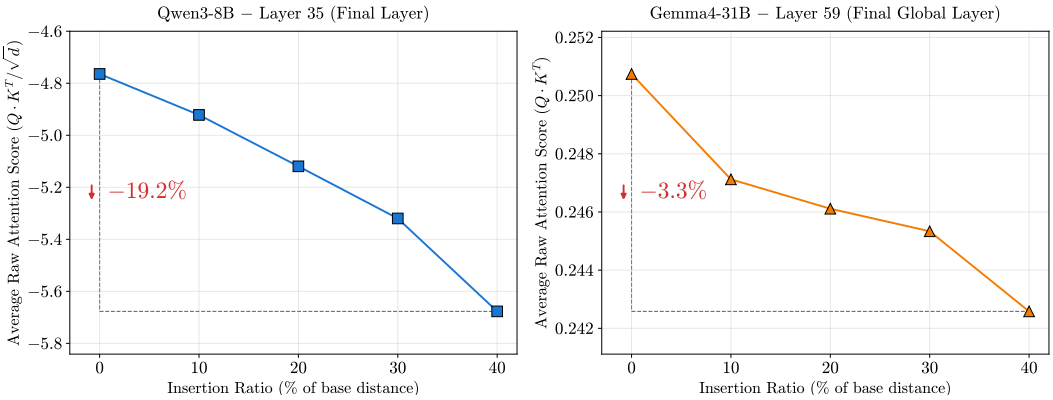


Figure 2: Average pre-softmax attention from the answer token to insights vs. insertion ratio at each model's final attention layer, under native scaling ($\mathbf{q} \cdot \mathbf{k}/\sqrt{d_{\text{head}}}$ for `Qwen3`; $\mathbf{q} \cdot \mathbf{k}$ for `Gemma-4`). Annotations show the relative decline from $\rho = 0$ to 0.4. For each model, the 95% problem-level paired-bootstrap CI for the mean score difference lies below zero (nonnegative bootstrap tail fraction < 0.001).

Results & Conclusion Figure 2 reports the relative decline in attention to critical insights from ρ=0 to ρ=0.4: 19.2% for `Qwen3-8B` (final layer) and 3.3% for `Gemma-4-31B-it` (final global-attention layer). We draw two conclusions: **(I)** Mean attention decreases monotonically across the tested ratios; the 95% paired-bootstrap CI for the $\rho = 0.4$ minus $\rho = 0$ difference lies below zero in both models. **(II)** The decay is not specific to a single architecture or RoPE configuration: it persists in Gemma-4-31B-it, which uses partial-rotary RoPE on only the first quarter of head dimensions and a different attention scaling, suggesting the phenomenon reflects a general property of trained attention rather than a feature of any specific positional encoding.

3 INSIGHTREPLAY

3.1 METHOD OVERVIEW

Figure 3 illustrates the core idea of InsightReplay. Reasoning unfolds as an interleaving of two types of content: *reasoning chunks* (R_t), in which the model performs step-by-step thinking, and *insights* (I_t), short distilled conclusions the model itself produces to summarize prior reasoning. As the trajectory progresses, the sequence of insights $\{I_t\}$ acts as a continuously updated, compressed abstraction of what has been concluded so far, kept near the active generation frontier so it remains accessible to subsequent reasoning. Appendix D.6 illustrates the procedure, while Appendix K reports an automated audit of 395 extracted insights for faithfulness and errors.

Method Description. The process begins from the question Q. We initialize $I_0 = Q$, treating the question itself as an insight that anchors all subsequent reasoning, consistent with Section 2.1. The model then enters a periodic loop. At each round t, conditioned on the full history $(Q, R_1, I_1, \ldots, R_{t-1}, I_{t-1})$, the model: (i) generates a reasoning chunk R_t, and (ii) generates a new insight I_t via a summarization prompt that asks the model to consolidate the conclusions reached so

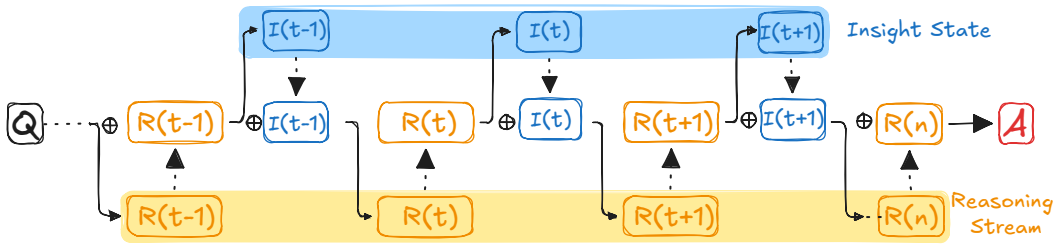


Figure 3: Illustration of InsightReplay. ▢: input question, ▢: reasoning chunk, ▢: insight, ▢: final answer, ⊕: appending a newly generated chunk to the conversation history, ⟶: forward inference, ⇢: grouping chunks into a logical state. The dashed arrows group chunks into two logical roles: insights collectively form the **Insight State**, and reasoning chunks form the **Reasoning Stream**. These groupings reflect logical roles only, and all chunks reside in the same conversation history.

far, with the thinking content $R_1, \ldots, R_t$ retained in context. Both R_t and I_t are appended to the history in order. Crucially, because I_t is generated in the presence of all prior insights $I_0, \ldots, I_{t-1}$, each new insight can reflect on and supersede earlier ones—incorporating new conclusions, correcting prior errors, or refining imprecise statements. The cumulative trace of insights thus forms an evolving abstraction of the reasoning so far, rather than a flat concatenation of independent summaries. After N rounds (we ablate $N \in \{1, 3\}$ in Section 4), the model produces the final answer A from the complete history. By construction, the most recent I_{t-1} always sits immediately before R_t, keeping the latest distilled conclusions close to the decoding position of subsequent reasoning.

3.2 Theoretical Analysis

We formalize the observations from Sections 2.1 and 2.2 and establish conditions for a rightward shift of the continuous optimum and a higher discrete peak accuracy.

3.2.1 Modeling

Following [5], for a task Q with difficulty T and model θ, the probability of an N-step CoT producing the correct final answer is:

$$P(a_{\text{final}}|Q,\theta,N) = \prod_{i=1}^{N} \underbrace{P(q_i|H_{i-1},Q,\theta,N)}_{\text{sub-question}} \underbrace{P(a_i|q_i,H_{i-1},Q,\theta,N)}_{\text{sub-answer}}, \tag{1}$$

where q_i is the i-th sub-question and $H_k := [q_1, a_1, \ldots, q_k, a_k]$ is the reasoning history up to step k.

They parameterize the sub-question accuracy as $(1 - \sigma(T))$, where $\sigma(T)$ denotes the probability that the model fails to generate the correct next sub-question for a task of total difficulty T, and the sub-answer accuracy as $(1 - E(N, M, T))$ with a linear error model $E(N, M, T) = T/(NM)$, where M represents the model's reasoning capability. Thus, the accuracy of N-step reasoning is:

$$A_0(N) = \alpha \left[(1 - \sigma(T))(1 - T/(NM))\right]^N, \tag{2}$$

$T, M > 0$, $0 < \sigma(T) < 1$, and $\alpha > 0$ is independent of N. Below we omit the factor α, which does not affect maximizers or comparisons. Crucially, their parameterization *drops the dependence on the history* H_{i-1}, implicitly assuming each step's accuracy is independent of its position.

However, our Section 2.2 shows: as i increases, the model's ability to access critical insights from earlier steps diminishes due to attention decay. We restore this missing dependence by introducing an *insight accessibility function* $\Phi(i)$ into the sub-question term. Specifically, we model the sub-question accuracy at step i as $(1 - \sigma(T)) \cdot \Phi(i)$, where $\Phi(i)$ captures how effectively the model can access prior insights when formulating the i-th sub-question. We apply $\Phi(i)$ to the sub-question term because generating the correct next sub-question requires synthesizing prior insights, whereas answering a sub-question, once correctly posed, depends primarily on the sub-question itself.

Assumption 1. *The continuous insight accessibility function* $\Phi : [0, \infty) \to (0, 1]$ *satisfies:*

1. $\Phi(0) = 1$ (zero distance ⇒ full accessibility);

2. Φ *is strictly monotonically decreasing (Section 2.2)* (farther $\Rightarrow$ less accessible);

3. $\lim_{i\to\infty} \Phi(i) = 0$ (infinite distance $\Rightarrow$ inaccessible).

Real-world CoT vs. InsightReplay. Let $p(N) := (1 - \sigma(T))(1 - E(N, M, T))$ and $\mathcal{N} := \{n \in \mathbb{Z}_{\geq 1} : n > T/M\}$. For $N \in \mathcal{N}$, the baseline uses accessibility $\Phi(i)$ at step i, whereas replay relocates insights to a fixed distance $d_0 \geq 0$ from the frontier, giving accessibility $\Phi(d_0)$ at every step. Distances are measured in units of reasoning steps. The normalized accuracies are

$$A_\Phi(N) = [p(N)]^N \prod_{i=1}^{N} \Phi(i), \qquad A_{\mathrm{IR}}(N) = [p(N) \cdot \Phi(d_0)]^N. \tag{3}$$

3.2.2 Main Results

Let $g(N) := \frac{d}{dN}[N \ln p(N)]$. For Lemma 2 and Theorem 1, we use the continuous relaxation $\widetilde{A}_\Phi(N) := \exp\{N \ln p(N) + \int_0^N \ln \Phi(x)\, dx\}$ and $\widetilde{A}_{\mathrm{IR}}(N) := [p(N)\Phi(d_0)]^N$ for real $N > T/M$. Appendix C details this sum-to-integral approximation.

Lemma 1 (Strict concavity). *$\ln A_0(N)$ is strictly concave for $N > T/M$, and consequently g is strictly decreasing on this domain.*

Lemma 2 (Uniqueness of continuous optima). *$\widetilde{A}_\Phi$ and $\widetilde{A}_{\mathrm{IR}}$ each admit a unique maximizer on $(T/M, \infty)$, characterized by*

$$g(\widetilde{N}^*_\Phi) = -\ln \Phi(\widetilde{N}^*_\Phi), \qquad g(\widetilde{N}^*_{\mathrm{IR}}) = -\ln \Phi(d_0). \tag{4}$$

Theorem 1 (Rightward shift in the continuous relaxation). *If $0 \leq d_0 < \widetilde{N}^*_\Phi$, then $\widetilde{N}^*_{\mathrm{IR}} > \widetilde{N}^*_\Phi$.*

Theorem 2 (Higher peak accuracy in the discrete model). *If $0 \leq d_0 < 1$, then $\max_{n\in\mathcal{N}} A_{\mathrm{IR}}(n) > \max_{n\in\mathcal{N}} A_\Phi(n)$.*

Proof intuition. For Theorem 1, $d_0 < \widetilde{N}^*_\Phi$ and Eq. (4) give $g(\widetilde{N}^*_{\mathrm{IR}}) < g(\widetilde{N}^*_\Phi)$; strict decrease of g implies the rightward shift. For Theorem 2, $d_0 < 1$ gives $\Phi(d_0) > \Phi(i)$ for every integer $i \geq 1$, so replay has higher accuracy at each admissible integer length and hence a higher maximum. Full proofs are in Appendix C.

4 Pure Inference Experiments

We evaluate InsightReplay across a $2 \times 3 \times 4$ experimental grid: two model scales, three model families, and four reasoning benchmarks. Full implementation details are provided in Appendix D.

4.1 Experimental Setup

Models. We evaluate 6 reasoning models, organized into a 2×3 grid of scales and families: **(1)** `Qwen3.5-9B` and `Qwen3.5-35B-A3B` [21], where the latter is a mixture-of-experts model with 3B active parameters; **(2)** `Gemma-4-E4B` and `Gemma-4-31B-it` [22], dense models that interleave local sliding-window attention with global full-context attention and apply partial-rotary RoPE on the first quarter of head dimensions in global layers; **(3)** `R1-Distill-Qwen-7B` and `R1-Distill-Qwen-32B` [3], distilled from DeepSeek-R1 onto Qwen2.5 backbones.

Benchmarks. We evaluate on four reasoning benchmarks: **AIME 2025/2026** [18] (60 math problems with integer answers), **HMMT** [23] (60 Harvard–MIT competition math problems), **GPQA Diamond** [24] (198 graduate-level science MCQs), and **LiveCodeBench v5** [25] (62 code problems uniformly sampled from the full benchmark; see Appendix D for the sampling procedure).

Variants. We compare four inference variants: **(1) Base**: standard thinking-mode CoT; **(2) VO** (*Verify-Only*): the model is prompted to continue thinking after its initial trace via a budget-forcing cue (`"Wait, let me double-check..."`), but performs no insight extraction or replay; **(3) IR1 / (4) IR3**: InsightReplay with 1 or 3 rounds of extract-then-replay. The contrast between IRk and VO isolates the contribution of the InsightReplay mechanism beyond extended thinking alone.

Sampling and metrics. For each problem, we draw 16 completions with temperature $\tau = 1$ and top-$p = 0.95$. The model generates until it produces an EOS token or reaches its native context length. We report accuracy as the primary metric, with average completion tokens as a cost measure.

Table 2: Main inference results on four reasoning benchmarks. **Base**: standard thinking-mode CoT inference. **VO** (Verify-Only): a control for test-time scaling, in which the model is instructed to keep thinking after its initial trace but performs no insight extraction or replay. **IR1/IR3**: InsightReplay with 1 or 3 rounds of extract-then-replay. Accuracy gains over Base are highlighted in blue when positive and red when negative; token columns report average completion tokens per sample.

Model	Data	Accuracy							Avg. completion tokens / sample						
		Base	VO	IR1	IR3	ΔVO	ΔIR1	ΔIR3	Base	VO	IR1	IR3	VO/B	IR1/B	IR3/B
Qwen3.5-9B	AIME	88.30	88.30	88.70	89.30	+0.00	+0.40	**+1.00**	37102	40835	41952	49404	1.10×	1.13×	1.33×
	GPQA	80.90	81.20	81.20	81.10	**+0.30**	**+0.30**	+0.20	15587	18104	18696	22584	1.16×	1.20×	1.45×
	LCB	52.50	56.60	58.20	61.20	+4.10	+5.70	**+8.70**	48097	52275	52526	58582	1.09×	1.09×	1.22×
	HMMT	83.50	83.90	84.20	84.50	+0.40	+0.70	**+1.00**	45477	49796	51088	58490	1.09×	1.12×	1.29×
Gemma4-E4B	AIME	45.40	46.00	46.00	46.10	+0.60	+0.60	**+0.70**	5874	7453	7414	10199	1.27×	1.26×	1.74×
	GPQA	56.40	56.60	56.70	57.00	+0.20	+0.30	**+0.60**	3342	4286	4266	6023	1.28×	1.28×	1.80×
	LCB	56.50	56.60	57.30	57.90	+0.10	+0.80	**+1.40**	6705	8191	8052	10357	1.22×	1.20×	1.54×
	HMMT	36.40	37.70	36.70	36.60	**+1.30**	+0.30	+0.20	7481	9203	9064	11906	1.23×	1.21×	1.59×
R1-Distill-7B	AIME	43.80	44.00	44.50	44.20	+0.20	**+0.70**	+0.40	12377	13432	14144	17879	1.09×	1.14×	1.44×
	GPQA	50.00	49.00	50.20	50.40	−1.00	+0.20	**+0.40**	6998	8059	7868	10246	1.15×	1.12×	1.46×
	LCB	7.30	8.90	8.50	8.50	**+1.60**	+1.20	+1.20	11631	12856	12438	14217	1.11×	1.07×	1.22×
	HMMT	28.30	29.10	29.00	29.00	**+0.80**	+0.70	+0.70	14417	15449	15818	19853	1.07×	1.10×	1.38×
Qwen3.5-35B	AIME	91.50	91.70	91.80	92.00	+0.20	+0.30	**+0.50**	30110	33584	34470	40966	1.12×	1.14×	1.36×
	GPQA	84.00	84.10	84.10	84.40	+0.10	+0.10	**+0.40**	19767	23228	23248	28101	1.18×	1.18×	1.42×
	LCB	67.60	68.50	70.60	71.70	+0.90	+3.00	**+4.10**	31538	35301	35886	41641	1.12×	1.14×	1.32×
	HMMT	88.50	88.50	88.40	88.60	+0.00	−0.10	**+0.10**	40124	44029	45161	51840	1.10×	1.13×	1.29×
Gemma4-31B	AIME	88.10	89.60	89.50	90.40	+1.50	+1.40	**+2.30**	7741	9620	9695	12882	1.24×	1.25×	1.66×
	GPQA	85.30	85.40	85.40	85.60	+0.10	+0.10	**+0.30**	5521	7014	6987	9320	1.27×	1.27×	1.69×
	LCB	80.80	81.00	81.50	81.90	+0.20	+0.70	**+1.10**	6553	8288	8224	10795	1.26×	1.26×	1.65×
	HMMT	88.10	88.70	88.10	88.90	+0.60	+0.00	**+0.80**	8499	10430	10541	13730	1.23×	1.24×	1.62×
R1-Distill-32B	AIME	55.60	55.90	56.40	57.50	+0.30	+0.80	**+1.90**	11065	11979	12318	15198	1.08×	1.11×	1.37×
	GPQA	62.20	62.00	62.30	62.80	−0.20	+0.10	**+0.60**	6598	7430	7277	8744	1.13×	1.10×	1.33×
	LCB	25.80	27.90	28.40	35.00	+2.10	+2.60	**+9.20**	9859	10990	10715	12526	1.11×	1.09×	1.27×
	HMMT	43.80	44.10	44.50	45.50	+0.30	+0.70	**+1.70**	12912	13867	14271	17278	1.07×	1.11×	1.34×
micro avg	–	64.82	65.18	65.47	66.04	+0.36	+0.65	**+1.21**	–	–	–	–	–	–	–
macro avg	–	62.11	62.72	63.01	63.75	+0.61	+0.90	**+1.65**	16891	18987	19255	23032	1.12×	1.14×	1.36×

4.2 Main Results

Table 2 summarizes the main results; Appendix J reports problem-level paired-bootstrap 95% confidence intervals and win/tie/loss counts for an independent six-setting IR1 rerun.

InsightReplay improves accuracy across all 24 settings. IR3 improves the reported accuracy over Base in all 24 settings, with macro- and micro-averaged gains of +1.65 and +1.21 points.

Gains are not merely an artifact of extended thinking. VO instructs the model to keep thinking after its initial trace but performs no insight extraction; despite using comparable token budgets, VO captures only +0.61 of the macro-averaged gain. The remaining +1.04, captured by IR3 over VO, is attributable to the InsightReplay mechanism itself. In other words, more than 60% of the total improvement comes from insight extraction and replay rather than from additional thinking time.

Token overhead is moderate. IR3 consumes on average 1.36× the tokens of Base, reflecting the insight extraction and replay rounds; IR1 incurs only 1.14× overhead while still delivering a macro-averaged improvement of +0.90. Appendix I reports throughput-amortized batch wall-clock measurements and the conditions for prefix-cache reuse.

Agentic Evaluation. We additionally evaluate InsightReplay in an agentic setting on SWE-Bench-Verified [26]. Due to space constraints, we defer the full protocol and results to Appendix E.

4.3 Empirical Validation of Peak Accuracy

We test the prediction of Theorem 2: InsightReplay should raise the achievable peak accuracy. We reuse the samples collected for the main results. For each problem, we sort the 16 Baseline completions by token count and bin them into 8 length levels (two samples per level). Binning is performed within each problem rather than globally to avoid conflating length with difficulty: a global short bin would be dominated by easy problems. The VO/IR1/IR3 samples inherit the bin assignment of their parent Baseline, so all variants are compared within each length level on samples sharing the same Baseline starting points. The Baseline curve in Figure 4 reports per-bin accuracy averaged over the four benchmarks (8 data points). For each of VO, IR1, and IR3, we compute per-bin accuracy under the same binning protocol and report the peak across the 8 bins, shown as a single star marker. See details in Appendix F. To probe whether the trend extrapolates beyond the ablated range, we additionally run IR5 on the LiveCodeBench v5 subset; the result is reported in Figure 1.

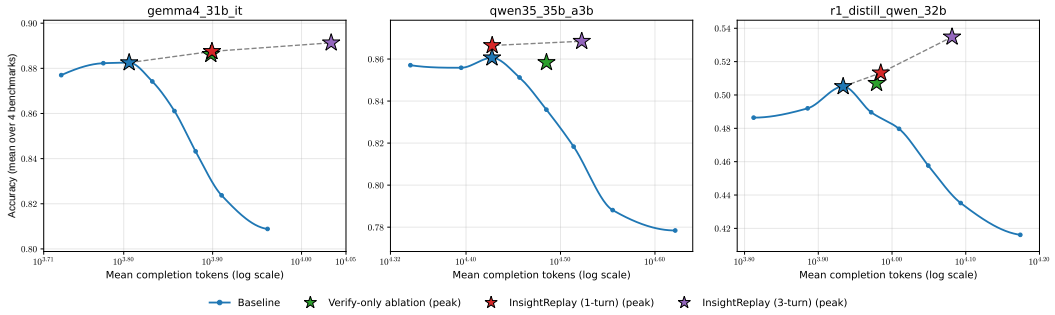


Figure 4: Accuracy vs. mean completion tokens for three $\sim$ 30B-scale models, averaged over the four benchmarks. The Baseline curve traces accuracy across 8 length levels of standard CoT; each star marker reports the peak accuracy of VO, IR1, or IR3 on the same set of problems. Dashed lines connect the Baseline peak to the IR1 and IR3 peaks, visualizing how peak accuracy evolves with the number of replay rounds. Results for $\sim$ 8B-scale models are shown in Appendix F due to space.

Figure 4 reports the curves for the three $\sim$ 30B-scale models. We highlight two findings.

The peak accuracy rises. The IR3 peak exceeds the Base peak on all models: $86.06\% \rightarrow 86.83\%$ on `Qwen3.5-35B-A3B`, $88.25\% \rightarrow 89.13\%$ on `Gemma-4-31B-it`, and $50.51\% \rightarrow 53.48\%$ on `R1-Distill-32B`. This matches Theorem 2 that the mitigation also lifts the achievable peak.

Budget forcing yields mixed results. VO raises the peak above Base on `Gemma-4-31B-it` and `R1-Distill-32B`, but falls below Base on `Qwen3.5-35B-A3B`. IR3 achieves a higher peak than VO on all three models.

5 Post-Training Experiments

To study whether InsightReplay can be reinforced through RL, we train `Qwen3-4B-Base` [27] with GRPO, comparing InsightReplay-enabled rollouts against standard CoT rollouts.

Setup. We use GRPO [28] as the policy-optimization algorithm and adopt the DAPO [29] soft overlong reward shaping to stabilize training. The actor is trained on DAPO-Math-15K [29] with a maximum response length of 30,720 tokens, a rollout group size of 8 per prompt, and a global batch size of 256 prompts. Validation is performed on AIME 2025 every 20 training steps with n=32 samples per problem, from which we report best@32, maj@32, and mean@32. Both runs use the same base model, data, and random seed. Full hyperparameters are provided in Appendix G.

Rollout Protocol. The two settings differ only in how rollouts are generated. In the **Baseline** setting, the policy generates a single thinking trace until it emits the EOS token or reaches the maximum response length. In the **InsightReplay** setting, once the policy emits the EOS token, we append a fixed continuation cue asking the model to extract and replay the insights (exact text in

Appendix G). Generation then continues under the same length budget, so the total response length is capped at the same maximum as the baseline. The cue is inserted with response_mask=0 which isolates the effect of the InsightReplay reasoning pattern from any auxiliary supervision signal.

Alternative Training Designs. We also experimented with a more elaborate scheme that explicitly encodes the multi-round InsightReplay structure during training via a 2D attention mask combined with per-round trace bundling. While methodologically clean, this design costs more compute than the protocol above. Nonetheless, we document this attempt in Appendix H for the interested reader.

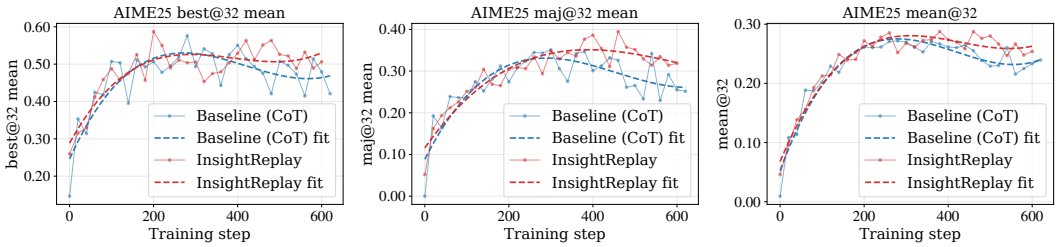


Figure 5: Validation accuracy on AIME 2025 throughout GRPO training, under InsightReplay rollouts vs. standard CoT rollouts. Across all three metrics (best@32, maj@32, mean@32). Dashed curves show cubic polynomial trend lines; the fitting procedure is described in Appendix G.

Results. Figure 5 reveals three observations comparing InsightReplay against the baseline.

(1) Training stability. The standard CoT baseline exhibits a pronounced late-training degradation: after step 300, all three metrics decline. The mean@32 drops from a peak of 27.9% at step 280 to 21.6% at step 560; best@32 similarly decays from 57.5% at step 280 to 42.0% at step 620. In contrast, InsightReplay maintains stable performance throughout training: from step 200 onward, mean@32 stays within the 25.0%–29.0% band, with no monotonic decline observed up to step 600.

(2) Peak accuracy. InsightReplay also achieves higher peak validation accuracy across all three metrics. On maj@32, InsightReplay peaks at 38.8% versus 34.1% for the baseline, an absolute improvement of +4.7 points; on mean@32, 28.8% versus 27.9%, a gain of +0.9 points; and on best@32, 59.1% versus 57.5%, a gain of +1.6 points. The peak gap is most pronounced on maj@32, which aggregates evidence across the 32 samples and is therefore most sensitive to whether the underlying reasoning is consistent, suggesting that InsightReplay improves the distribution of outputs.

(3) The advantage is not an initialization effect. For the first $\sim$ 200 steps, the two methods' learning curves overlap within sampling noise on all three metrics; divergence emerges only after step $\sim$ 250. This rules out alternative explanations such as the InsightReplay prompt placing the policy in a more favorable starting region or its reward shape biasing the optimization from the outset. The benefit must be produced during training rather than inherited from the setup.

Together, the three observations show that the InsightReplay reasoning pattern can be reinforced through RL training rather than only used at inference time. Beyond improving peak accuracy, it stabilizes long-CoT GRPO training in a regime where the standard CoT baseline suffers a pronounced late-training degradation, and does so without relying on any early-learning advantage.

6 Conclusion

We introduced InsightReplay, a stateful reasoning approach that periodically extracts critical insights from the CoT trace and replays them near the active generation frontier. By mitigating the attention decay that critical insights suffer in long trajectories, InsightReplay reshapes the inverted-U between CoT length and accuracy, shifting the peak rightward and lifting it upward, thereby expanding the regime in which extended reasoning translates into higher accuracy. We support this through both theoretical analysis and empirical validation: positive gains across all 24 cells of our scale $\times$ family $\times$ benchmark grid in the pure-inference setting, and consistent improvements in training stability and peak accuracy when used as a rollout protocol for post-training.

AI USE STATEMENT

We used generative AI tools to assist with manuscript editing and experimental planning. These tools helped improve the clarity and readability of the text and provided suggestions on experimental design and comparisons. The authors take responsibility for the final manuscript, experimental decisions, and reported results.

ETHICS STATEMENT

This work studies reasoning with existing language models and established benchmarks, without human-subject experiments or collection of new personal data. The main considerations are the reliability and potential misuse of model-generated outputs, including code, and the computational overhead of additional reasoning. Code evaluations use containerized environments, and we report token usage to characterize inference cost. Use and redistribution of the underlying models, datasets, and software should follow their respective licenses and terms.

REPRODUCIBILITY STATEMENT

The inference setup, prompts, and evaluation rules are described in Appendix D. Appendices A and B describe the diagnostic experiments; Appendix F describes length-based binning. The theoretical model is presented in Section 3.2, with proofs in Appendix C; post-training hyperparameters and the continuation cue are in Appendix G.

REFERENCES

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

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

[3] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[4] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[5] Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. When more is less: Understanding chain-of-thought length in llms. In *International Conference on Learning Representations*, volume 2026, pages 127108–127138, 2026.

[6] Wei-Lin Chen, Liqian Peng, Tian Tan, Chao Zhao, Blake JianHang Chen, Ziqian Lin, Alec Go, and Yu Meng. Think deep, not just long: Measuring llm reasoning effort via deep-thinking tokens. *arXiv preprint arXiv:2602.13517*, 2026.

[7] Aryo Pradipta Gema, Alexander Hägele, Runjin Chen, Andy Arditi, Jacob Goldman-Wetzler, Kit Fraser-Taliente, Henry Sleight, Linda Petrini, Julian Michael, Beatrice Alex, et al. Inverse scaling in test-time compute. *arXiv preprint arXiv:2507.14417*, 2025.

[8] Paul C Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. Thought anchors: Which llm reasoning steps matter? *arXiv preprint arXiv:2506.19143*, 2025.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[10] Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3351–3363, 2025.

[11] Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. C3ot: Generating shorter chain-of-thought without compromising effectiveness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24312–24320, 2025.

[12] Hang Yuan, Bin Yu, Haotian Li, Shijun Yang, Christina Dan Wang, Zhou Yu, Xueyin Xu, Weizhen Qi, and Kai Chen. Not all tokens are what you need in thinking. *arXiv preprint arXiv:2505.17827*, 2025.

[13] Zeju Li, Jianyuan Zhong, Ziyang Zheng, Xiangyu Wen, Zhijian Xu, Yingying Cheng, Fan Zhang, and Qiang Xu. Making slow thinking faster: Compressing llm chain-of-thought via step entropy. In *International Conference on Learning Representations*, volume 2026, pages 38331–38349, 2026.

[14] Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. Inftythink: Breaking the length limits of long-context reasoning in large language models. In *International Conference on Learning Representations*, volume 2026, pages 107391–107430, 2026.

[15] Chenxiao Yang, Nathan Srebro, David McAllester, and Zhiyuan Li. Pencil: Long thoughts with short memory. *arXiv preprint arXiv:2503.14337*, 2025.

[16] Lovish Madaan, Aniket Didolkar, Suchin Gururangan, John Quan, Ruan Silva, Ruslan Salakhutdinov, Manzil Zaheer, Sanjeev Arora, and Anirudh Goyal. Rethinking thinking tokens: Llms as improvement operators. *arXiv preprint arXiv:2510.01123*, 2025.

[17] Zicheng Lin, Tian Liang, Jiahao Xu, Qiuzhi Lin, Xing Wang, Ruilin Luo, Chufan Shi, Siheng Li, Yujiu Yang, and Zhaopeng Tu. Critical tokens matter: Token-level contrastive estimation enhances llm's reasoning capability. *arXiv preprint arXiv:2411.19943*, 2024.

[18] Mathematical Association of America. Maa invitational competitions. https://maa.org/maa-invitational-competitions/, 2025. Accessed: 2026-04-10.

[19] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[20] Federico Barbero, Alex Vitvitskyi, Christos Perivolaropoulos, Razvan Pascanu, and Petar Veličković. Round and round we go! what makes rotary positional encodings useful? *arXiv preprint arXiv:2410.06205*, 2024.

[21] Qwen Team. Qwen3.5. https://huggingface.co/collections/Qwen/qwen35, 2026. Accessed: 2026-05-03.

[22] Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

[23] Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *Advances in Neural Information Processing Systems*, 38, 2026.

[24] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

[25] Naman Jain, 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 *International Conference on Learning Representations*, volume 2025, pages 58791–58831, 2025.

[26] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157, 2024.

[27] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

[29] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

[30] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[31] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297, 2025.

[32] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[33] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[34] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[35] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[36] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20286–20332, 2025.

A EXPERIMENTAL DETAILS FOR PROPERTY I

Dataset. We use 60 problems from the American Invitational Mathematics Examination (AIME) spanning the 2024 and 2025 competitions.

Thinking trace generation. For each problem, we generate 16 independent chain-of-thought samples using `Qwen3-8B` in its native thinking mode, and report results averaged over these samples. We use the Qwen3 chat template and prompt the model with:

> *Solve the following math competition problem step by step. Show your detailed reasoning, then give your final answer as a single integer inside* `\boxed{}`.

Generation is performed via vLLM [30] with sampling (`temperature` $= 0.6$, `top_p` $= 0.95$), tensor parallelism across 2 GPUs, `max_model_len` $= 30{,}000$, `dtype=bfloat16`, and a maximum of 30,000 new tokens. The model's output follows the `<think>...</think>` convention; we parse the content inside the tags as the thinking trace and the content outside as the final response. For traces that reach the 30,000-token limit before producing a closing `</think>` tag, we treat the content up to the closing tag as the thinking trace if it appears, and otherwise treat the entire generated output as the thinking trace. All seven conditions in Section 2.1 are evaluated on the same set of $60 \times 16 = 960$ generated traces.

Insight extraction. For each thinking trace, we use `Qwen3-8B` to extract a list of 5–7 key insights. The extraction model receives the original question and the full thinking trace, and is instructed to:

> *Extract exactly* 5 *to* 7 *key insights from this thinking, in the order they were derived.*

The model produces a numbered list (e.g. *1. The area of* $\triangle ABC$ *is* $6\sqrt{14}$ *by Heron's formula.*), which we parse with a regex-based extractor that captures each enumerated entry. We cap the result at 7 insights per problem in the rare cases where the model produces more. This phase uses vLLM with greedy decoding, `max_tokens` $= 8192$, and the same tensor parallelism configuration as the trace generation phase. Across all 60 problems and 16 samples per problem, the mean number of extracted insights is 6.9, with all extractions falling within the requested 5–7 range.

Probability scoring. We load `Qwen3-8B` via HuggingFace Transformers in `bfloat16` precision on a single GPU. For each problem, we construct a prefix string following the Qwen3 chat template:

```
<|im_start|>user
{user_prompt}<|im_end|>
<|im_start|>assistant
<think>
{THINK_CONTENT}
</think>

\boxed{
```

where `THINK_CONTENT` varies across conditions (see below). We then append the ground-truth answer tokens $a_{\text{gt}} = (a_1, a_2, \ldots, a_K)$ (the tokenized integer answer) to the prefix and perform a single forward pass over the concatenated sequence. The score is the joint log-probability of the answer tokens conditioned on the prefix:

$$\log P_\theta(a_{\text{gt}} \mid x_{\text{prefix}}) = \sum_{k=1}^{K} \log P_\theta\big(a_k \mid x_{\text{prefix}}, a_{<k}\big), \tag{5}$$

where x_{prefix} denotes the prompt up to and including the opening token sequence `\boxed{`, and K is the number of tokens in the tokenized ground-truth answer.

Condition definitions. Let $\mathbf{q}$ denote the problem statement, $\mathbf{t}$ the full thinking trace generated in the previous paragraph, and $\mathbf{f} = (f_1, \ldots, f_K)$ the insights extracted by the extraction step (formatted as a numbered list). Let $\mathbf{r}$ denote a sequence of pseudo-random tokens whose token count matches that of $\mathbf{t}$ for the same problem. The seven conditions differ only in `THINK_CONTENT`:

- **(A) Full thinking (baseline)**: $\mathbf{t}$
- **(B) Repeated question + insights**: `Question:` $\mathbf{q}$, followed by $\mathbf{f}$
- **(C) Insights only**: $\mathbf{f}$
- **(D) Full thinking + repeated question + insights**: $\mathbf{t}$, then `Question:` $\mathbf{q}$, followed by $\mathbf{f}$
- **(E) Full thinking + insights**: $\mathbf{t}$, followed by $\mathbf{f}$
- **(F) Random tokens + repeated question + insights**: $\mathbf{r}$, then `Question:` $\mathbf{q}$, followed by $\mathbf{f}$
- **(G) Random tokens + insights**: $\mathbf{r}$, followed by $\mathbf{f}$

Adjacent components are concatenated with two newline characters (`\n\n`). To construct $\mathbf{r}$, we sample strings of the form `xq{0-9999}` with a fixed random seed of 42 for reproducibility, then truncate or pad at the token level to match $|\text{tokenize}(\mathbf{t})|$ exactly.

Software and hardware. All experiments use vLLM 0.19.0 for trace generation and insight extraction, and HuggingFace Transformers 5.6.0 for probability scoring. Models are served in `bfloat16` precision. Trace generation and insight extraction each run on 2× NVIDIA GPUs with tensor parallelism; probability scoring runs on a single GPU.

B Experimental Details for Property II

Models. `Qwen3-8B`: 36 layers, 32 query heads, 8 KV heads via GQA, head dim 128, full RoPE, with QK-norm. `Gemma-4-31B-it`: alternates sliding-window and global-attention layers with different head dimensions and per-layer `partial_rotary_factor`; we extract scores only from global layers (head_dim = 512) so the attention pattern is not truncated by the sliding window. The insight-extraction model is `Qwen3.5-397B-A17B`.

CoT generation. Each target model generates a complete CoT trace via vLLM with greedy decoding (`max_new_tokens=38,000`, `temperature=1.0`, `do_sample=False`, `bfloat16`). The user prompt is:

> *Solve the following math problem step by step. Show your complete reasoning, then give the final answer inside* `\boxed{}`.

For Qwen3-8B we wrap the trace in `<think>...</think>`; for Gemma-4 we use the native thought block. The ground-truth answer is appended as `\boxed{<answer>}` at the fixed position p_a (the first token after `\boxed{`).

Insight extraction and token-level localization. We prompt `Qwen3.5-397B-A17B` to list 5–7 critical insights, each containing a verbatim span copied from the trace. Each insight is then relocated in the full tokenized sequence via a case-sensitive substring match plus character-to-token offset mapping using the target model's tokenizer, yielding `(start_token_idx, end_token_idx)` for the insight span c. Insights whose verbatim span cannot be recovered, or whose sentence position leaves fewer than one downstream sentence before p_a, are discarded. This yields an average of ∼6 usable insights per problem.

Filler unit calibration. The filler unit `"\n "` is calibrated per tokenizer at runtime to determine its token cost τ. The number of filler units actually inserted at ratio ρ is $N = \lfloor \lfloor \rho \cdot d \rfloor / \tau \rfloor$, and the realized distance after insertion is $d' = d + N \cdot \tau$. Random allocation of fillers across sentence boundaries uses a seeded RNG (`seed=42` plus problem-id hash) for reproducibility.

Attention-score extraction. We run a layer-by-layer forward pass with `bfloat16`. At each layer ℓ we recompute the post-RoPE query vector $\mathbf{q}_{p_a}^{(\ell,h)}$ and post-RoPE key vectors $\mathbf{k}_t^{(\ell,h)}$ for $t \in c$, and store the per-layer score $s^{(\ell)}(c, a)$ together with the realized distance d'. For GQA models we map each query head to its shared KV head via integer division by the GQA group size.

Model-specific handling. `Qwen3-8B` applies standard full-dimension RoPE and QK-norm; we replicate the ordering q-proj $\to$ q-norm $\to$ RoPE, and likewise for keys. `Gemma-4-31B-it` uses head_dim $= 512$ on global layers; to avoid materializing the full attention matrix at this size we use PyTorch SDPA for `head_dim>256` and fall back to `flash_attention_2` elsewhere. Only the explicit $\mathbf{q} \cdot \mathbf{k}$ computation for the (answer, insight) pair enters the logging path.

Aggregation and reporting. Scores are aggregated per (ℓ, ρ) by first averaging across samples, then across insights within a problem, then across problems. For distance-based plots, we bin by realized distance d' with bin size 500 tokens. We additionally identify candidate "receiver" heads as the top-32 heads ranked by score kurtosis across variants.

Compute. Attention extraction uses 8-way data parallelism across NVIDIA H200 GPUs. Each worker loads one full model instance (`Qwen3-8B` on one GPU; `Gemma-4-31B-it` sharded across multiple GPUs via `device_map`).

C Proofs of Theoretical Results

C.1 Preparation

Throughout, let $c := T/M > 0$, $s := 1 - \sigma(T) \in (0, 1)$, and $p(N) = s(1 - c/N)$ for real $N > c$. Integer reasoning lengths lie in $\mathcal{N} = \{n \in \mathbb{Z}_{\geq 1} : n > c\}$; outside this domain the per-step error c/N is at least one. The replay distance $d_0 \geq 0$ is fixed independently of N and measured in units of reasoning steps. Assumption 1 requires Φ to be continuous, positive, and strictly decreasing, with $\Phi(0) = 1$ and $\Phi(x) \to 0$.

Normalization. Equation (2) includes the positive constant α. The two curves in Eq. (3) omit this common factor; multiplying both by α changes neither their maximizers nor their ordering. To keep the normalization explicit, define

$$L(N) := \ln \frac{A_0(N)}{\alpha} = N \ln p(N), \qquad g(N) := L'(N). \tag{6}$$

Thus $\ln A_0(N) = \ln \alpha + L(N)$, and the derivatives of L and $\ln A_0$ agree. In all comparisons below, A_Φ and A_{IR} retain the normalized definitions in Eq. (3).

C.1.1 Proof of Lemma 1

Proof. By Eq. (2),

$$\ln A_0(N) = \ln \alpha + N \ln s + N \ln\Big(1 - \frac{c}{N}\Big). \tag{7}$$

The first term is constant and the second is linear in N. For the remaining term,

$$h(N) := N \ln\Big(1 - \frac{c}{N}\Big), \quad h'(N) = \ln\Big(1 - \frac{c}{N}\Big) + \frac{c}{N - c}, \quad h''(N) = -\frac{c^2}{N(N-c)^2} < 0. \tag{8}$$

Consequently, $\ln A_0$ is strictly concave and

$$g'(N) = L''(N) = (\ln A_0)''(N) = -\frac{c^2}{N(N-c)^2} < 0, \qquad N > c. \tag{9}$$

□

C.1.2 Continuous Relaxation and Proof of Lemma 2

Discrete curves and their continuous surrogate. For integer $n \in \mathcal{N}$, set $S(n) := \sum_{i=1}^{n} \ln \Phi(i)$. The exact log-accuracies are

$$\ln A_\Phi(n) = L(n) + S(n), \qquad \ln A_{\mathrm{IR}}(n) = L(n) + n \ln \Phi(d_0). \tag{10}$$

For continuous optimization, replace $S(n)$ with the integral $I(N) := \int_0^N \ln \Phi(x)\, dx$ and define, for real $N > c$,

$$\widetilde{A}_\Phi(N) := \exp\{L(N) + I(N)\}, \qquad \widetilde{A}_{\mathrm{IR}}(N) := \exp\{L(N) + N \ln \Phi(d_0)\}. \tag{11}$$

The replay curve has a direct continuous extension; the baseline requires a sum-to-integral approximation. These are the functions analyzed in Lemma 2 and Theorem 1.

The lower integration limit 0 includes one unit interval for each of the n discrete steps. Since $\ln \Phi$ is decreasing and $\ln \Phi(0) = 0$, its right and left Riemann sums give

$$S(n) \le I(n) \le S(n) - \ln \Phi(n), \qquad 0 \le I(n) - S(n) \le -\ln \Phi(n). \tag{12}$$

Thus $\widetilde{A}_\Phi$ is a surrogate, not an exact interpolation of A_Φ at integer lengths. The bound does not assert a uniformly small approximation error or closeness of the two optimal lengths. Replacing $\int_0^N$ by $\int_1^N$ would change the relaxed baseline log-curve only by an N-independent constant and hence leave its maximizer unchanged; we use $\int_0^N$ consistently throughout.

Proof of Lemma 2. Continuity of Φ and the fundamental theorem of calculus give the *exact* derivatives of the relaxed curves:

$$(\ln \widetilde{A}_\Phi)'(N) = g(N) + \ln \Phi(N), \qquad (\ln \widetilde{A}_{\mathrm{IR}})'(N) = g(N) + \ln \Phi(d_0). \tag{13}$$

Here $g(N) = \ln s + \ln(1 - c/N) + c/(N - c)$, so $g(N) \to +\infty$ as $N \downarrow c$ and $g(N) \to \ln s < 0$ as $N \to \infty$. Since $\ln \Phi(c)$ and $\ln \Phi(d_0)$ are finite, both derivatives in Eq. (13) tend to $+\infty$ at the lower endpoint. At infinity, the baseline derivative tends to $-\infty$, while the replay derivative tends to $\ln s + \ln \Phi(d_0) < 0$.

Both derivatives are continuous and strictly decreasing: g is strictly decreasing by Lemma 1, $\ln \Phi(N)$ is strictly decreasing, and $\ln \Phi(d_0)$ is constant. Each derivative therefore crosses zero exactly once. Each curve is strictly increasing before this point and strictly decreasing after it, so each has a unique global maximizer on (c, ∞). The stationary conditions are precisely Eq. (4):

$$g(\widetilde{N}^*_\Phi) = -\ln \Phi(\widetilde{N}^*_\Phi), \qquad g(\widetilde{N}^*_{\mathrm{IR}}) = -\ln \Phi(d_0). \tag{14}$$

□

The assumption $0 < \sigma(T) < 1$ ensures $\ln s < 0$ and excludes, for example, $\sigma(T) = 0$ and $d_0 = 0$, where replay need not have a finite maximizer. Uniqueness here concerns real-valued lengths in the relaxed model; integer lengths can tie for the maximum.

C.2 Main Results

C.2.1 Proof of Theorem 1

Proof. The stated assumption $0 \le d_0 < \widetilde{N}^*_\Phi$ and strict decrease of Φ imply $\Phi(d_0) > \Phi(\widetilde{N}^*_\Phi)$. Using Eq. (4),

$$g(\widetilde{N}^*_{\mathrm{IR}}) = -\ln \Phi(d_0) < -\ln \Phi(\widetilde{N}^*_\Phi) = g(\widetilde{N}^*_\Phi). \tag{15}$$

Since g is strictly decreasing on (c, ∞), this gives $\widetilde{N}^*_{\mathrm{IR}} > \widetilde{N}^*_\Phi$. □

This is a statement about the continuous relaxation. Its condition is explicit: $d_0 < 1$ alone does not imply $d_0 < \widetilde{N}^*_\Phi$ when the relaxed optimum is below one. No claim that d_0 is automatically small relative to the optimum is needed for the proof.

C.2.2 Proof of Theorem 2

Proof. We work directly with the integer-length curves in Eq. (3); no integral approximation is used. For $n \in \mathcal{N}$, $p(n) > 0$, and $0 \le d_0 < 1$ implies $\Phi(d_0) > \Phi(i)$ for every integer $1 \le i \le n$. Therefore,

$$A_{\mathrm{IR}}(n) = [p(n)]^n [\Phi(d_0)]^n > [p(n)]^n \prod_{i=1}^{n} \Phi(i) = A_{\Phi}(n). \tag{16}$$

Both integer-length curves attain a maximum: they are positive on $\mathcal{N}$ and bounded above by $s^n \to 0$, so only finitely many lengths can exceed their value at the smallest admissible length. Choose any maximizers n^*_{Φ} and n^*_{IR}, without assuming uniqueness. Evaluating the pointwise inequality at n^*_{Φ} gives

$$\max_{n \in \mathcal{N}} A_{\mathrm{IR}}(n) = A_{\mathrm{IR}}(n^*_{\mathrm{IR}}) \ge A_{\mathrm{IR}}(n^*_{\Phi}) > A_{\Phi}(n^*_{\Phi}) = \max_{n \in \mathcal{N}} A_{\Phi}(n). \tag{17}$$

□

Integer-length ordering. Under $0 \le d_0 < 1$, the discrete ratio $r(n) := A_{\mathrm{IR}}(n)/A_{\Phi}(n)$ satisfies $r(n+1)/r(n) = \Phi(d_0)/\Phi(n+1) > 1$. If n^*_{Φ} is the largest baseline maximizer, then for any $n < n^*_{\Phi}$ in $\mathcal{N}$, $A_{\mathrm{IR}}(n) = r(n)A_{\Phi}(n) < r(n^*_{\Phi})A_{\Phi}(n^*_{\Phi}) = A_{\mathrm{IR}}(n^*_{\Phi})$. Hence every replay maximizer is at least n^*_{Φ}, although equality can occur. For example, with $c = 0.1$, $s = 0.5$, $\Phi(x) = e^{-x}$, and $d_0 = 0.5$, both discrete curves maximize at $n = 1$. Thus the strict continuous shift of Theorem 1 is not asserted as a strict shift between integer optima.

D Pure Inference Experiment Details

This appendix expands on the experimental setup of Section 4, covering inference infrastructure, decoding configuration, prompting templates, evaluation protocol, and reproducibility notes that were omitted from the main text for brevity.

D.1 Hardware and Inference Stack

All inference experiments are run on a single node with 8× NVIDIA H200 141GB GPUs interconnected via NVLink. Models are served through vLLM [30] v0.19.0 via its OpenAI-compatible HTTP endpoint, with one server instance per model and all 8 GPUs visible to that instance. The parallelism layout depends on model size: the 30B-tier models (Qwen3.5-35B-A3B, Gemma-4-31B-it, R1-Distill-Qwen-32B) use tensor parallelism TP $= 2$ with data parallelism DP $= 4$, while the 8B-tier models (Qwen3.5-9B, Gemma-4-E4B-it, R1-Distill-Qwen-7B) use TP $= 1$ with DP $= 8$. KV-cache memory utilization is capped at 0.92. The maximum context length is set per model family at the largest value the KV budget can hold under the chosen DP: 200,000 tokens for Qwen and Gemma, and the native 131,072 tokens for the R1-distill family. We pass `--trust-remote-code` so that custom model code – e.g. Gemma's interleaved local/global attention with sliding-window size 1024 – loads with its released configuration; no custom attention modification is applied on our side.

D.2 Decoding Configuration

We sample with temperature $\tau = 1.0$ and nucleus top-$p = 0.95$; all other vLLM sampling parameters (e.g. `top_k`, `repetition_penalty`, presence/frequency penalties) are left at their library defaults so the comparison is not biased by ad-hoc decoding tweaks. We pass `skip_special_tokens=False` so that thinking-block markers (⟨`think`⟩ / ⟨`/think`⟩ for Qwen and R1-distill, ⟨|`channel`⟩`thought` / ⟨`channel`|⟩ for Gemma) appear in the raw response; downstream extraction relies on these markers to split the thinking trace from the post-think tail. The $k = 16$ samples per problem are produced by issuing k independent requests with $n = 1$ rather than a single request with $n = k$: this guarantees that vLLM's continuous-batching scheduler treats each sample as an independent unit, and that an HTTP retry on a transient error does not invalidate sibling samples. Independence across samples is provided by vLLM's internal RNG; no per-sample seed is pinned.

D.3 PROMPTING TEMPLATES

User-turn header. The user-turn content is a dataset-specific header concatenated with the formatted question; the chat template then wraps it into a model-specific prompt. The header explicitly specifies the answer-format the grader keys on:

- **AIME**: "*Solve the math problem. Final answer must be a single non-negative integer in* ⟨Answer⟩...⟨/Answer⟩."
- **GPQA Diamond**: "*Select the best answer. Final answer must be a single letter (A, B, C, or D) in* ⟨Answer⟩...⟨/Answer⟩."
- **LiveCodeBench v5**: "*Solve the coding task. Put your final solution in a single Python code block delimited by* ```python ... ```."
- **HMMT (Feb & Nov 2025)**: "*Solve the math problem. Final answer must be the EXACT value (simplified, no decimal approximations) in* \boxed{...}. *If there are multiple solutions, list all of them inside the single* \boxed{...} *separated by commas.*" The boxed-LaTeX format is used (rather than the ⟨Answer⟩ tag of AIME/GPQA) because HMMT answers are arbitrary symbolic expressions (e.g. \frac{1}{2}, 8\sqrt{10}) that are graded by symbolic verification rather than string equality.

InsightReplay injection. InsightReplay is a single post-hoc edit of the baseline response: once the baseline emits the thinking-close marker, we (i) ask the model to enumerate the important conclusions of its own thinking (ii) splice a self-verification block back in *before* the thinking-close marker; and (iii) let the model continue from that point until it re-emits a thinking-close marker and writes a fresh final answer. The block is phrased in the model's first person so that, embedded inside the thinking block, it reads as a continuation of the internal monologue rather than an external instruction. Concretely the block contains:

- an opening line "*Wait, before I commit to a final answer, let me restate what's being asked and double-check the key conclusions I've been relying on*";
- a one-line restatement of the user's header and the question, so the model does not drift from the task in a long thinking stream;
- a numbered list of the extracted insights under "*Key conclusions so far:*

D.4 DATASETS AND PREPROCESSING

We use four benchmarks. **AIME 2025/2026** concatenates the official problem sets of both years, yielding 60 short-answer problems (30 per year) with non-negative integer answers; each record carries a <year>-<idx> identifier and the gold answer string. **GPQA Diamond** is the full 198-question Diamond split; the upstream release already embeds the four answer choices into each problem's question field in their original order (mixing "A. ...", "A) ...", and "a) ..." styles across problems), with the structured choices field left null, so we do not shuffle the options and we do not append choice lines on our side – the model sees the question verbatim as released. **LiveCodeBench v5** is a fixed 62-problem uniform subsample of the full 880-problem v5 release, drawn once to bound the compute cost of running 30B-class models with $k = 16$ samples and $\sim$20,000-token CoT lengths; the subsample is shipped with the data files, so all models see the identical 62-problem subset, with the official hidden test cases attached verbatim. **HMMT (Feb & Nov 2025)** concatenates the MathArena releases of HMMT February 2025 and HMMT November 2025, yielding 60 short-answer problems (30 per contest, sources MathArena/hmmt_feb_2025 and MathArena/hmmt_nov_2025); unlike AIME, HMMT gold answers are arbitrary LaTeX expressions (rationals, radicals, exact closed forms) rather than integers in $[0, 999]$, so we keep them as raw strings and defer simplification to the symbolic grader (Section D.5). We apply no few-shot exemplars, no rephrasing of the math/science text, and no rewriting of the code prompts.

D.5 EVALUATION PROTOCOL

The per-sample correct flag is computed by first stripping the model's thinking block (everything up to and including the last thinking-close marker) and then applying dataset-specific extraction:

- **AIME.** The post-think tail is searched for an integer answer using an ordered list of patterns: ⟨Answer⟩ *N* ⟨/Answer⟩, \boxed{N}, \Answer{N}, "Answer: N" (with markdown bold/italic tolerated around the word "*Answer*" and between the colon and the digit), and a final "open ⟨Answer⟩ without a closing tag" fallback. The first pattern that matches wins; ties within a pattern resolve to the last occurrence. Grading is exact integer comparison against the gold answer.
- **GPQA Diamond.** The same priority list adapted to single-letter answers A–D. We deliberately do *not* match "Option A/B/C/D" phrases inside the reasoning, which would otherwise silently override the model's actual final answer with the last-mentioned option. Grading is case-insensitive letter equality.
- **LiveCodeBench v5.** The last ```` ```python ... ``` ```` block in the post-think tail is taken as the candidate solution and executed in a Docker sandbox (python:3.10-slim, --network none, --cpus 1, --memory 2g) against the full hidden test suite. The runner walks all test cases serially within a single sandbox invocation under a 20-second wall-clock budget covering all tests for that candidate; a sample is correct iff every test passes. We grade 8 candidates in parallel via a process pool. Function-style tests call Solution().<func>(*args); stdin-style tests pipe the input to python solution.py and compare stdout (after stripping leading/trailing whitespace) against the expected output.
- **HMMT.** A LaTeX-aware grader is used because gold answers are arbitrary symbolic expressions. We first extract the candidate's final answer from the post-think tail with a brace-counting scan over the *last* \boxed{...} (brace counting handles nested {} that a flat regex would mis-balance), with end-of-text fallbacks to $$ ... $$, \[... \], inline $ ... $ (with a leading "*lhs* =" stripped), and finally a bare standalone final line. All fallbacks are restricted to the end of the response so math fragments mid-reasoning cannot win. Equivalence is checked with the math_verify library (HuggingFace MathArena) configured with float_rounding=3 ($\sim 5\times10^{-4}$ tolerance) plus two robustness fallbacks: (i) if the candidate contains \pm, we expand it into both signed branches and additionally retry the comma-joined two-solution form, since the prompt asks for multi-solution sets to be packed into a single \boxed{} separated by commas; (ii) if symbolic verification fails, we evaluate gold and candidate to 30-digit sympy floats and accept on relative tolerance 10^{-3}, catching cases where a model truncates a decimal just inside the symbolic-rounding cutoff. This setup judges \sqrt{640} equivalent to 8\sqrt{10} and 0.5 equivalent to \frac{1}{2}, while keeping 0.6 and \frac{1}{2} distinct.

D.6 Case Analysis

This subsection illustrates the InsightReplay inference pipeline (one extraction + one verification round) on a concrete example from our evaluation runs.

Problem.

> Let A be the set of positive integer divisors of 2025. Let B be a randomly selected subset of A. The probability that B is a nonempty set with the property that the least common multiple of its elements is 2025 is $\frac{m}{n}$, where m and n are relatively prime positive integers. Find $m + n$.

(2025 AIME II, Problem 7; correct answer is 237. We use GEMMA-4-E4B-IT for this example.)

Color key. Across the three passes below, the visual coding is consistent: **cyan** blocks are model *inputs* (the prompts the model is given), **blue** blocks are model *outputs* (the generations we read back), the **orange** sub-block inside Pass 3's input marks the portion that is *spliced in by the inference harness* rather than written by the model, and any *red italic* text is an *editorial annotation by us* — elisions of long verbatim content, computed parameter values, or explanatory notes — that is *not* part of the literal prompt or output.

Pass 1: Baseline Reasoning

Input.

`User:` Let A be the set of positive integer divisors of 2025. Let B be a randomly selected subset of A. The probability that B is a nonempty set with the property that the least common multiple of its elements is 2025 is $\frac{m}{n}$, where m and n are relatively prime positive integers. Find $m + n$. The final answer must be a single non-negative integer wrapped in `<Answer>...</Answer>`.

Output.

`<think>`
The set of divisors of $2025 = 3^4 \cdot 5^2$ has size $|A| = (4+1)(2+1) = 15$, so the total number of subsets is 2^{15}. By inclusion–exclusion on the conditions "no element with $v_3(d) = 4$" (P_3) and "no element with $v_5(d) = 2$" (P_5):

$$\begin{aligned} |P_3| = 2^{12}, \quad |P_5| = 2^{10}, \quad |P_3 \cap P_5| &= 2^8, \\ K &= 2^{15} - 2^{12} - 2^{10} + 2^8 \\ &= 2^8 \left(2^7 - 2^4 - 2^2 + 1\right) \\ &= 256 \left(128 - 16 - 4 + 1\right) \\ &= 256 \cdot 105. \end{aligned}$$

So $P = \frac{256 \cdot 105}{256 \cdot 128} = \frac{105}{128}$, giving $m = 105$, $n = 128$, and $m + n = 233$.
`</think>`

`<Answer>233</Answer>`

The model's reasoning is largely correct, but it commits a subtle arithmetic slip in the very last simplification: $128 - 16 - 4 + 1 = 109$, not 105. The slip is reported *verbatim* as the final answer, 233.

For this trace the baseline thinking body is 3326 tokens, which the adaptive cap formula $\min(11, 2 + \lfloor (t-1)/5000 \rfloor)$ maps to $K = 2$ insights; this K flows through into the extraction prompt below.

PASS 2: EXTRACTION

Input. The model is given a fresh prompt that wraps its own `<think>` body from Pass 1 and asks for a faithful, non-evaluative listing of the conclusions reached.

`User:` You just finished an internal reasoning pass on the problem below. List the most important concrete conclusions you reached during that pass — the facts, equations, intermediate results, or commitments that your final answer will rest on. These will be fed back to you as a checklist to verify before you commit.

`RULES:`

- Output AT MOST *K (computed from the thinking-body length; see Pass 1 commentary)* insights (fewer is fine). Pick the most load-bearing.
- Each insight: one short sentence, factual and specific.
- Do NOT re-derive, do NOT evaluate correctness, do NOT restate the final answer.
- Output format: plain bullet list, one insight per line, each line starting with '– '.

`Problem:` *(problem restated verbatim — identical to Pass 1's input)*

`Your prior reasoning:`
(Pass 1's `<think>` body verbatim — the divisor count, inclusion–exclusion derivation, and final arithmetic ending at $K = 256 \cdot 105$ and $m + n = 233$. Elided here for space; in the actual prompt this is several hundred tokens of literal text from Pass 1's output.)

`Key conclusions to verify (bullet list only):`

Output.

– The number of subsets $B \subseteq A$ such that $\text{lcm}(B) = 2025$ is $K = 2^{15} - 2^{12} - 2^{10} + 2^8 = 256 \cdot 105$.

– The resulting probability is $\frac{m}{n} = \frac{105}{128}$, where m and n are relatively prime positive integers.

The extractor faithfully surfaces the model's own conclusions — *including the arithmetic bug that* $256(128 - 16 - 4 + 1) = 256 \cdot 105$. This faithfulness is critical: the next pass needs to see the model's actual reasoning chain, not a sanitized version, otherwise the verification step has nothing concrete to check.

PASS 3: VERIFICATION AND COMMIT

Input. The model receives a re-thought version of its original `<think>` block: the original reasoning from Pass 1 is preserved verbatim *up to but not including* the closing `</think>` marker, and immediately before that marker the inference harness splices in a structured verification block (highlighted in orange below). The injection is phrased in the model's own voice so the continuation reads as a single uninterrupted thinking stream.

`User:` *(problem statement, identical to Pass 1's input)*

`Assistant: <think>`
(Pass 1's `<think>` body verbatim — divisor count, inclusion–exclusion setup, derivation of $K = 256 \cdot 105$, and conclusion $m+n = 233$ — preserved up to but not including the closing `</think>` marker. Elided here for space.)

[Spliced in by harness]

`Wait, before I commit to a final answer, let me restate what's being asked and double-check the key conclusions I've been relying on.`

`The user's request:` Solve the math problem. Final answer must be a single non-negative integer in `<Answer>...</Answer>`.

`Question:` Let A be the set of positive integer divisors of 2025
... (problem restated verbatim) ... Find $m + n$.

`Key conclusions so far:`

1. The number of subsets $B \subseteq A$ such that $\text{lcm}(B) = 2025$ is $K = 2^{15} - 2^{12} - 2^{10} + 2^8 = 256 \cdot 105$.
2. The resulting probability is $\frac{m}{n} = \frac{105}{128}$, where m and n are relatively prime positive integers.

`Before finalizing, my current working answer is 233.`

`Let me verify each of these conclusions and check whether they actually support this answer -- or whether I've missed something that would change it.`

(The model now continues from this point, still inside the same `<think>` block.)

Output.

Let me re-check each conclusion. The set-up is correct: $|A| = 15$, $|P_3| = 2^{12}$, $|P_5| = 2^{10}$, $|P_3 \cap P_5| = 2^8$. Direct evaluation:

$$K = 32768 - (4096 + 1024 - 256) = 32768 - 4864 = 27904.$$

Now I'll re-check the factored form $K = 2^8(2^7-2^4-2^2+1)$: $2^7-2^4-2^2+1 = 128-16-4+1$. Compute step by step: $128 - 16 = 112$, $112 - 4 = 108$, $108 + 1 = 109$.

The previous simplification was erroneous: $128 - 16 - 4 + 1 = 109$, not 105. So $K = 256 \cdot 109 = 27904$ (consistent with the direct evaluation), and $P = \frac{256 \cdot 109}{2^{15}} = \frac{109}{128}$. Then $m = 109$, $n = 128$, and $\gcd(m, n) = 1$ since 109 is prime and $128 = 2^7$. Therefore $m + n = 109 + 128 = 237$.
</think>

<Answer>237</Answer>

The verification pass catches and corrects the arithmetic slip from Pass 1, flipping the answer from the incorrect 233 to the correct 237. The original reasoning chain was almost entirely sound, and a single concrete checklist of the model's own intermediate claims was enough to surface and resolve the error without any external feedback.

E SWE-BENCH EVALUATION

To evaluate the effectiveness of InsightReplay on real-world software-engineering tasks, we conduct experiments on SWE-Bench-Verified [26], a benchmark consisting of GitHub issues paired with corresponding test cases that verify whether the issue has been resolved.

Dataset. We randomly select 100 instances from SWE-Bench-Verified. Each instance contains a natural-language bug report, a Docker image with the target repository checked out at the relevant commit under `/testbed`, and a set of test cases (both fail-to-pass and pass-to-pass) for evaluation.

Model and Inference Setup. We use **Qwen3.5-35B-A3B** as the base model, served via vLLM [30] with 4-way data parallelism across 4 GPUs. The vLLM server is configured with a maximum model length of 40,960 tokens, 0.9 GPU memory utilization, and the Hermes tool-call parser (`--tool-call-parser hermes`) to enable native function-calling support. For generation, we set temperature to 1.0, top-p to 0.95, and the maximum output tokens to 30,000.

Agent Architecture. The model operates in an agentic loop with access to six tools: `file_read`, `file_edit`, `file_write`, `grep`, `glob`, and `bash`, all scoped to the `/testbed` directory inside a Docker container. Each tool is registered as a native function schema in the chat-completion API, allowing the model to invoke tools through structured tool-call messages rather than text-based parsing. The agent iterates for up to 50 turns per sample (`max_turns`=50). Each turn consists of a model generation followed by tool execution. A prompt-guard mechanism monitors the estimated prompt token count and stops the agent early if it approaches the context-window limit of 40,960 tokens (with a 512-token margin). When the model determines the task is complete, it outputs the token `DONE` to terminate the loop. The final patch is extracted via `git diff` from the Docker container.

Sampling and Evaluation. For each of the 100 instances, we independently sample $k = 8$ trajectories, yielding 800 candidate patches in total. Each candidate patch is evaluated by applying it to a fresh Docker container with the original repository state, installing necessary dependencies, and running the instance-specific test suite. A candidate is considered successful if all fail-to-pass tests now pass and all pass-to-pass tests remain passing. We report the **candidate-level pass rate**, i.e., the fraction of all 800 candidate patches that resolve the issue.

Compared Methods. We compare three prompting strategies, all sharing the same agent architecture, tools, and inference hyperparameters:

- **CoT Zero-shot**: The system prompt instructs the model to think explicitly and concretely before each action and follow a five-step workflow: (1) understand the bug report, (2) reproduce the bug, (3) inspect relevant source files, (4) analyze the root cause and apply a fix, and (5) re-run tests to confirm resolution. No few-shot examples are provided.

Table 3: Candidate-level pass rates on SWE-Bench-Verified (100 selected instances, $k = 8$ samples per instance, `Qwen3.5-35B-A3B`).

Benchmark	CoT Zero-shot	CoT Few-shot	InsightReplay (Ours)
SWE-Bench-Verified	23.12% (185/800)	20.87% (167/800)	**29.75**% (238/800)

- **CoT Few-shot**: Extends the CoT zero-shot prompt with three detailed few-shot examples demonstrating best practices: understanding the issue before patching, preferring narrow inspection and minimal edits, and always validating with the original failure mode.
- **InsightReplay (Ours)**: Uses the same five-step workflow but introduces a structured `<Finding>...</Finding>` mechanism. After each high-level workflow step, the model is required to output exactly one `<Finding>` block summarizing what was learned in that step. Generation stops at the closing tag, and the accumulated insights are rolled up into a summary message that replaces the growing conversation history. This compresses the context while preserving key insights and enforces a disciplined step-by-step progression. The system prompt also includes a detailed few-shot example illustrating the full five-step InsightReplay workflow.

Results. Table 3 reports candidate-level pass rates. InsightReplay achieves 29.75% (238/800), outperforming both CoT Zero-shot (23.12%, 185/800) and CoT Few-shot (20.87%, 167/800). Relative to the stronger CoT baseline, this is an absolute gain of 6.63 percentage points and a relative improvement of approximately 28.7%.

F Length-Sweep Protocol Details

To jointly characterize the relationship between compute budget and accuracy for the length sweep in Section 4.3, we apply the following bin-based visualization protocol per (model, dataset, method) triple.

Per-problem binning. For each problem, we independently sample $k = 16$ baseline completions (temperature 1.0, top-p 0.95), producing 16 thinking trajectories of varying length. We sort these 16 samples by completion tokens (ascending) and bin them uniformly into 8 length levels (2 samples per level). Crucially, this binning is done *per problem*, not globally across all problems: since problems differ substantially in intrinsic difficulty, a global token-based binning would conflate length with difficulty (a "short" bin would be dominated by easy problems, while a "long" bin would be dominated by hard problems). Per-problem binning instead identifies, for each problem, its 8 relative length levels (very-short, short, medium, . . ., very-long), so that within each level we compare samples that come from the *same set of problems*—isolating the length effect from the difficulty effect. Figure 6 illustrates this procedure schematically.

Sample-level pairing for VO/IR. The corresponding VO / IR1 / IR3 samples *strictly inherit* the baseline's token ordering: each VO/IR/IR3 sample is paired to its source baseline sample (sorted by that baseline's token count). This pairing ensures that within any given length level, all four methods compare samples extended from *the same set of baseline starting points*, isolating the method's contribution from sampling variance.

Per-level statistics. Within each length level, we record two statistics:

- x = the mean of the 2 samples' *actual* completion tokens (baseline curve = baseline's own tokens; the other three methods = baseline tokens + injected insights + extended generation tokens);
- y = the level's accuracy (mean of $\{0, 1\}$ correctness).

Cross-benchmark aggregation. We first compute the 8 levels' (x, y) values per problem, then average over all problems to obtain the 8 data points for each (model, dataset, method) triple. For the cross-benchmark merged curves shown in Figures 4 and 7, we equally weight the 8 data points

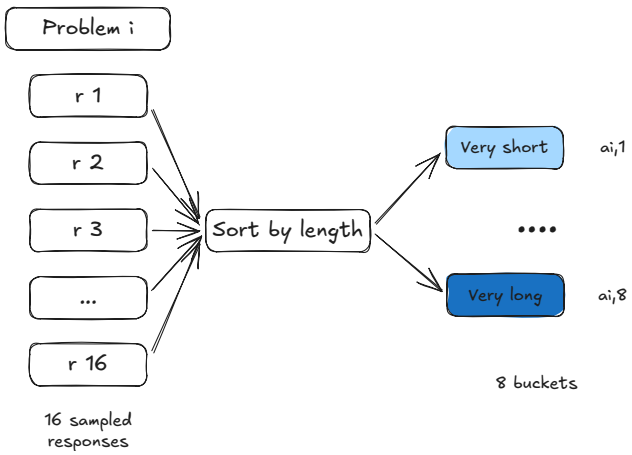


Figure 6: Per-problem binning procedure. For each problem i, we draw $k = 16$ independent baseline samples $(r_1, \ldots, r_{16})$, sort them by completion length, and uniformly partition them into 8 length buckets (2 samples per bucket), denoted $a_{i,1}$ (very short) through $a_{i,8}$ (very long). Aggregation across problems then averages within each bucket index, producing the 8 data points per (model, method) curve in Figure 4. Critically, the binning is done *per problem*: each $a_{i,k}$ refers to problem i's own k-th length level relative to its own sampling distribution, not to a global token threshold. This controls for variation in problem difficulty.

across AIME / GPQA / LiveCodeBench / HMMT (each benchmark contributing $1/4$), to avoid the largest dataset (GPQA, $n = 3168$) dominating the merged curve.

Results on $\sim$ 8B-scale models. Figure 7 reports the length-sweep curves for the three $\sim$ 8B-scale models (`Qwen3.5-9B`, `Gemma-4-E4B`, and `R1-Distill-Qwen-7B`), applying the same protocol as in Section 4.3. The same two patterns observed at the 30B scale—rightward shift of the optimal CoT length and upward shift of the peak accuracy—hold here as well, indicating that the inverted-U reshaping predicted by Theorems 1 and 2 is robust across model scales.

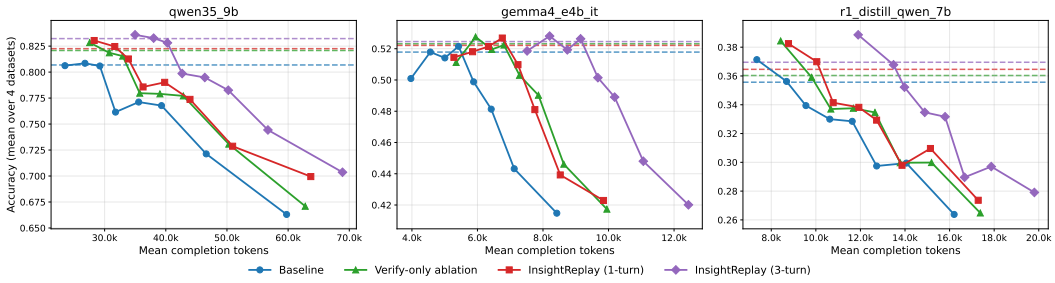


Figure 7: Length-sweep results for $\sim$ 8B-scale models (`Qwen3.5-9B`, `Gemma-4-E4B`, and `R1-Distill-Qwen-7B`), averaged with equal weight over the four benchmarks. Dashed lines mark each variant's peak (mean of its top-3 length levels). The same rightward and upward shift of the IR3 curve relative to Base observed at the 30B scale (Figure 4) holds at the 8B scale, confirming that the inverted-U reshaping is consistent across model sizes.

G Post-Training Details

All experiments are implemented on top of the open-source **verl** RL training framework [31], using its FSDP2 actor backend and vLLM rollout engine.

This appendix lists the full set of hyperparameters used for the InsightReplay RL study reported in the main text. Both the baseline (standard CoT rollout) and the InsightReplay run

share *identical* settings: same base model, same dataset, same optimizer, same KL/clip configuration, same rollout sampler, same length budget, and the same compute topology. The two runs differ in exactly one configuration line—the INSIGHTREPLAY run additionally sets `actor_rollout_ref.rollout.agent.default_agent_loop=insight_replay_agent`, which activates the rollout-time continuation-cue injection described in Section 5; all remaining values below are shared.

Compute and parallelism.

- Hardware: 16 nodes $\times$ 8 GPUs (NVIDIA H800) $=$ 128 GPUs total.
- Training framework: `verl` with FSDP2; rollout engine: vLLM.
- Ulysses sequence-parallel size: 2; FSDP shard size: 8; rollout tensor-parallel size: 1 (one vLLM instance per GPU, 128 instances total).
- Dynamic batching enabled for both actor update and inference; per-GPU token cap $= 1\times(L_{\text{prompt}} + L_{\text{resp}})$ for the actor and $2\times(L_{\text{prompt}} + L_{\text{resp}})$ for inference / log-prob.
- No parameter, optimizer, or policy offloading; reshard after forward; entropy checkpointing enabled.
- vLLM rollout uses chunked prefill with `max_num_batched_tokens`=32,768, `gpu_memory_utilization`=0.7, prefix caching disabled, and `free_cache_engine=True`.

Model and data.

- Base model: `Qwen3-4B-Base` (no SFT warm-up).
- Training set: `DAPO-Math-15K` (RL split).
- Validation set: AIME 2025.
- Maximum prompt length: 2,048 tokens.
- Maximum response length: 30,720 tokens (identical for both runs; under INSIGHTREPLAY the cue and the second-phase continuation share this same budget).
- Left truncation; data shuffling enabled.

Optimization.

- Algorithm: GRPO (`adv_estimator=grpo`).
- Optimizer: AdamW; learning rate 1×10^{-6}; warmup steps $=$ 10; weight decay $=$ 0.1; gradient clipping $=$ 1.0.
- Loss aggregation: `token-mean`.
- Entropy coefficient: 0; entropy computed from logits with chunking.
- KL regularization *fully disabled*: no KL term in the reward (`use_kl_in_reward=False`, $\beta_{\text{KL}} = 0$), no KL loss against the reference policy (`use_kl_loss=False`, coefficient $= 0$).
- PPO-style clipping with DAPO's clip-higher asymmetry: $\varepsilon_{\text{low}} = 0.2$, $\varepsilon_{\text{high}} = 0.28$.
- No critic warm-up.

Rollout protocol.

- Group size per prompt: $n = 8$ trajectories.
- Global batch: 256 prompts per training step ($\Rightarrow 256 \times 8 = 2{,}048$ trajectories per step).
- PPO mini-batch: 128 prompts.
- Training sampler: temperature $=$ 1.0, top-p $=$ 1.0, top-k $=-1$ (no top-k truncation); `ignore_eos=False`.
- Validation sampler: temperature $=$ 1.0, top-p $=$ 0.7, $n = 32$ samples per problem (used for best@32, maj@32, mean@32).

Reward shaping. We use the DAPO reward manager with the soft overlong penalty enabled to stabilize training:

- `overlong_buffer.enable=True`.
- Buffer length: 4,096 tokens (i.e., the soft penalty engages once a response is within 4,096 tokens of the 30,720-token cap).
- Penalty factor: 1.0.
- `max_resp_len` passed to the reward equals the rollout `max_response_length` (30,720).

Schedule and checkpointing.

- Total training epochs: 10 (both runs use the same seed, the same data shuffling, and the same number of update steps).
- Validation frequency: every 20 training steps; validation is also run before training starts.
- Checkpoint frequency: every 50 training steps; checkpoints save {model, optimizer, extras}.
- `balance_batch=False`; `resume_mode=auto`.
- Logging: TensorBoard.

Rollout continuation cue. For the INSIGHTREPLAY rollout, once the policy emits the end-of-sequence token, we append the following fixed continuation cue before allowing generation to continue under the same response-length budget:

> *Wait, before I commit to a final answer, let me restate what's being asked and cross-verify by a completely different method than what I used above.*
>
> *The user's request:* `{question}`
>
> *Key conclusions so far:*

This cue is inserted with response_mask=0, so the injected tokens affect the model only as conditioning context and never contribute directly to the policy loss.

InsightReplay-specific configuration. The only configuration difference between the two runs is the rollout agent loop:

- Baseline: `default_agent_loop` unset (single-pass CoT rollout).
- INSIGHTREPLAY: `actor_rollout_ref.rollout.agent.default_agent_loop=insight_replay_agent`. Whenever phase 1 terminates by emitting EOS, the rollout worker splices in the fixed continuation cue (Section 5) and resumes generation; the total response is still capped at 30,720 tokens, so the per-trajectory length budget is identical to the baseline.

A consolidated summary of all hyperparameters appears in Table 4.

Training-curve smoothing for visualization. In Figure 5, for each metric we additionally overlay a smooth trend line obtained by fitting a degree-3 polynomial $\hat{f}(t) = \sum_{k=0}^{3} a_k t^k$ to the per-step validation values (t_i, y_i) via ordinary least squares:

$$(a_0, a_1, a_2, a_3) = \arg\min_{a} \sum_i \big(y_i - \hat{f}(t_i)\big)^2.$$

We then evaluate $\hat{f}$ on 200 equally spaced points spanning the observed training-step range and render the resulting curve as a dashed line. This fit is used purely as a visual smoother: it suppresses per-evaluation noise so that the relative trend between the two runs is easier to read at a glance. We do not draw any quantitative conclusion from the polynomial coefficients themselves.

Table 4: Full RL training hyperparameters. Values are shared by the baseline and INSIGHTREPLAY runs unless otherwise noted.

Hyperparameter	Value
Model & data	
Base model	Qwen3-4B-Base (no SFT)
Training set	DAPO-Math-15K
Validation set	AIME 2025
Max prompt length	2,048
Max response length	30,720
Algorithm	
RL algorithm	GRPO
Reward manager	DAPO (soft overlong)
KL in reward / KL loss	disabled / disabled
Clip ratio (low, high)	$(0.2,\ 0.28)$
Entropy coefficient	0
Loss aggregation	token-mean
Overlong buffer length	4,096
Overlong penalty factor	1.0
Optimization	
Optimizer	AdamW
Learning rate	1×10^{-6}
LR warmup steps	10
Weight decay	0.1
Gradient clipping	1.0
Batching	
Global batch (prompts)	256
Rollouts per prompt	8
PPO mini-batch (prompts)	128
Dynamic batching	enabled
Rollout (training)	
Sampler	temperature 1.0, top-p 1.0, top-k off
Engine	vLLM, chunked prefill, no prefix cache
GPU memory utilization	0.7
Rollout (validation)	
Sampler	temperature 1.0, top-p 0.7
Samples per problem	$n = 32$
Schedule	
Total epochs	10
Validation frequency	every 20 steps
Checkpoint frequency	every 50 steps
Compute	
Hardware	128× NVIDIA H800 (16 nodes)
Framework	verl + FSDP2; vLLM rollout
Ulysses SP size	2
FSDP shard size	8
Rollout TP size	1
Run-specific (only difference)	
`default_agent_loop`	unset (baseline) / InsightReplay

H ALTERNATIVE TRAINING METHOD

This appendix documents an earlier attempt to train the multi-round InsightReplay structure directly, rather than inducing it at rollout time with a fixed continuation cue. The pipeline has two stages: we first construct supervised multi-round traces, and then optimize the policy with a training procedure that explicitly mirrors the round-wise visibility pattern used at inference time.

H.1 Method Overview

Base model and goal. We start from `Qwen3.5-9B-Base` [21]. The goal of this alternative design is to make the multi-round reasoning structure explicit during training: each round exposes only the question, the currently visible reasoning context, and the replayed intermediate insights that should be accessible at that round.

Stage 1: Supervised multi-round trace construction. To obtain SFT data in the target format, we sample 2,000 math problems from `DAPO-Math-17K` [29]. For each problem, a strong teacher model, `Qwen3.5-397B-A17B`, generates a full solution, which we then decompose into a sequence of reasoning steps and compact intermediate insights. We convert these into multi-round targets where all non-final rounds contain a reasoning segment together with an intermediate insight, and the last round produces the final answer. This procedure yields 1,892 valid problems and 8,102 SFT training entries. A concrete example appears in Appendix H.3.

Stage 2: Direct multi-round RL training. Our RL training procedure consists of two steps, illustrated in Figure 8.

In **Step 1**, we start from the standard causal mask and add a round-level visibility mask over the concatenated multi-round response. This makes the intended computation explicit: at each round, the model can attend only to the original question, the relevant reasoning tokens, and the replayed insights that should be visible in that round, while all other tokens are masked out on top of the causal constraint. However, this alone creates a RoPE mismatch: masked tokens become invisible under attention, but they still occupy positions during training, so the relative distance between two visible tokens no longer matches the rollout setting in which those masked tokens are physically absent.

In **Step 2**, we regroup each training case by round. For every round, we delete all masked-out tokens, keep only the tokens visible in that round, and then bundle the resulting per-round traces from the same case for joint optimization. This yields contiguous `position_ids` that match rollout exactly. For the advantage, standard GRPO uses a sample-wise normalized quantity within a prompt group, e.g.,

$$A_i^{\mathrm{GRPO}} = \frac{R_i - \mathrm{mean}(\{R_j\})}{\mathrm{std}(\{R_j\})}.$$

In our alternative design, we instead assign the same case-level reward to every round from the same case and use $A_r = R - b$ for all rounds r, reusing the same shared advantage across the bundled round-level traces.

The direct multi-round recipe yields mixed results: it shows late improvements in $\mathrm{mean@32}$ and $\mathrm{best@32}$ relative to its own baseline, but does not provide a consistent advantage across all evaluation metrics. Given its additional round-wise processing and implementation complexity, we did not adopt it as the main training recipe.

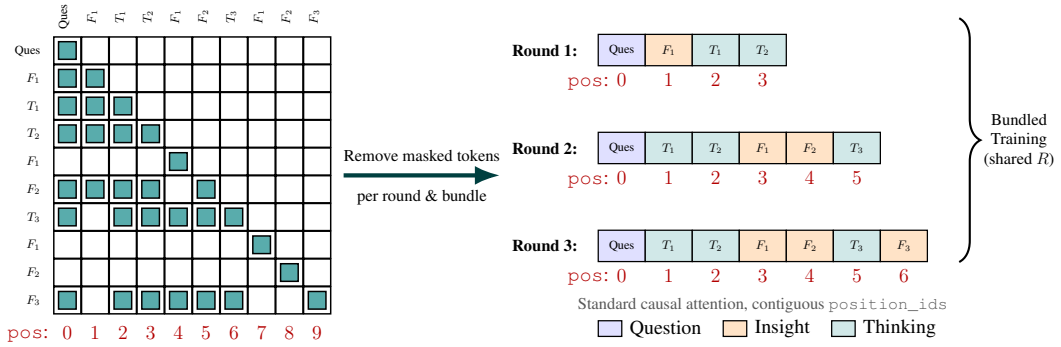
H.2 SFT Data Construction Details

Source Data. We sample 2,000 math problems from DAPO-Math-17K [29], a large-scale mathematical reasoning dataset.

Solution Generation. For each sampled problem, we use `Qwen3.5-397B-A17B` to generate a complete solution in non-think mode, with temperature $\tau = 0.7$ and top-$p = 0.95$.

Post-hoc insight Construction. Each generated solution is then decomposed into a multi-round insight format as follows:

1. The full solution is split into logical reasoning steps (up to 8 steps).
2. Each reasoning step is summarized into a concise insight (1–2 sentences, up to 256 tokens) via a separate model call.



Step 1: 2D Attention Mask

Step 2: Per-Round Traces

Figure 8: Two-step training construction for the alternative multi-round method. Step 1 applies a round-specific visibility mask over the concatenated trajectory; Step 2 removes masked tokens round by round and bundles the resulting traces so that training-time attention and position indices match rollout.

3. Multi-turn SFT entries are constructed so that each round contains a reasoning step in `<think>...</think>` followed by an intermediate conclusion in `<finding>...</finding>` tags, with the final round producing the answer.

This process yields 1,892 valid cases and 8,102 SFT training entries. The average number of insight rounds per problem is 3.3, with a maximum of 9 rounds.

H.3 INSIGHT FORMAT: DETAILED EXAMPLE

This subsection illustrates the multi-round insight format with a concrete example from our RL training rollout. The model solves the problem in 3 rounds: two insight rounds followed by one answer round. At each round, the model receives all accumulated context (previous reasoning and insights) and generates the next step.

Problem.

Define a sequence $\{a_n\}_{n\geq 1}$ as follows:

- For $n \leq 2$: $a_n = n - 1$
- For all $n \geq 2$: a_n is the remainder of $a_{n-1} + a_{n-2}$ divided by 3

Compute $\sum_{i=2018}^{2025} a_i$.

ROUND 1: INSIGHT ROUND

Input. The model receives the original problem as the user prompt.

Model Output.

<think>
First, compute the initial terms of the sequence:

$$a_1 = 0, \quad a_2 = 1, \quad a_3 = (1+0) \bmod 3 = 1,$$
$$a_4 = (1+1) \bmod 3 = 2, \quad a_5 = (2+1) \bmod 3 = 0,$$
$$a_6 = (0+2) \bmod 3 = 2, \quad a_7 = (2+0) \bmod 3 = 2,$$
$$a_8 = (2+2) \bmod 3 = 1, \quad a_9 = (1+2) \bmod 3 = 0,$$
$$a_{10} = (0+1) \bmod 3 = 1.$$

</think>

<finding>The sequence $\{a_n\}$ exhibits periodic behavior starting from $n = 3$, with values $1, 2, 0, 2, 2, 1$ repeating cyclically. The period is 6.</finding>

The model computes the first 10 terms and identifies periodicity. Note that the insight contains an *incorrect* period estimate (6 instead of 8)—the model will self-correct in the next round.

ROUND 2: INSIGHT ROUND

Input. The model receives the original problem concatenated with the full Round 1 reasoning (<think> content) and the Round 1 insight as accumulated context.

Model Output.

<think>
Examining the first 10 terms: $a_1 = 0, a_2 = 1, a_3 = 1, a_4 = 2, a_5 = 0, a_6 = 2, a_7 = 2, a_8 = 1, a_9 = 0, a_{10} = 1$.

Comparing (a_1, a_2) and (a_9, a_{10}): both are $(0, 1)$. Since this is a linear recurrence determined by consecutive pairs, the sequence repeats from this point. The period length is $T = 9 - 1 = 8$, and the terms within one period are: $0, 1, 1, 2, 0, 2, 2, 1$.

The sum over one period: $S_{\text{cycle}} = 0 + 1 + 1 + 2 + 0 + 2 + 2 + 1 = 9$.

We need $\sum_{i=2018}^{2025} a_i$. First, determine a_{2018}'s position in the period: $2018 \bmod 8$. The mapping $n \bmod 8$: $1 \to a_1(0),\ 2 \to a_2(1),\ \ldots,\ 0 \to a_8(1)$. </think>

<finding>The sequence is periodic with period 8, repeating $0, 1, 1, 2, 0, 2, 2, 1$. The sum of one period is 9.</finding>

Building on the first round's computation, the model re-examines the sequence, corrects the period to 8, computes the period sum, and sets up the index mapping for the final calculation.

ROUND 3: ANSWER ROUND

Input. The model receives the original problem, the full reasoning from Rounds 1 and 2, and all accumulated insights.

Model Output.

<think>
Compute $2018 \pmod 8$: $2018 = 8 \times 252 + 2$, remainder is 2.

So a_{2018} corresponds to the 2nd term in the period.

The terms $a_{2018}, \ldots, a_{2025}$ map to positions $\{2, 3, 4, 5, 6, 7, 8, 1\}$, which is exactly a permutation of $\{1, 2, 3, 4, 5, 6, 7, 8\}$.

Therefore, their sum equals $S_{\text{cycle}} = 9$.

Answer: 9 </think>

The sequence $\{a_n\}$ is periodic with period $T = 8$ and per-period sum $S = 9$. Since $2018 \equiv 2 \pmod 8$, the 8 terms a_{2018} through a_{2025} span exactly one complete period, so their sum equals 9.

Answer: $\boxed{9}$

H.4 SFT Hyperparameters

We fine-tune for one epoch with the following setup:

- Learning rate: 1×10^{-5}
- Maximum sequence length: $40{,}000$ tokens
- Global batch size: 128
- Micro batch size per GPU: 4
- Hardware: 8 GPUs with FSDP
- Gradient checkpointing enabled
- No padding mode with right truncation

H.5 RL Setup

We apply GRPO [28] in two settings: an insight-enabled rollout and an insight-disabled baseline. The RL training set is DAPO-Math-15K, a $15{,}917$-problem subset of DAPO-Math-17K. During RL training, we sample 8 responses per prompt. Validation is performed on AIME 2025 [18] every 20 training steps using 32 responses per problem, from which we compute $\mathrm{best@32}$, $\mathrm{maj@32}$, and $\mathrm{mean@32}$.

Reward Function. We use the DAPO reward manager with the same soft overlong penalty as in the main post-training experiment. The model's answer is extracted from the last `Answer:` line, normalized, and compared against the ground truth, yielding a correctness reward of $+1.0$ for correct answers and -1.0 for incorrect ones. In addition, we enable the soft overlong penalty to discourage trajectories that approach the maximum response budget: `overlong_buffer.enable=True`, the penalty factor is 1.0, and `max_resp_len` passed to the reward matches the rollout `max_response_length`.

RL Hyperparameters. Both experiments use the following shared configuration on 8 NVIDIA H200 GPUs:

- Algorithm: GRPO with token-level loss aggregation
- Sequence parallelism (Ulysses SP): 8
- Rollout tensor parallelism: 2
- Global training batch size: 64 prompts
- Rollout n: 8 responses per prompt (512 total samples per step)
- PPO mini-batch size: 32
- Learning rate: 1×10^{-6} with 10 warmup steps
- KL loss coefficient: 0.001
- Maximum prompt length: $27{,}000$ tokens
- Maximum response length: $25{,}000$ tokens
- Total training: 5 epoch
- Validation frequency: every 20 steps

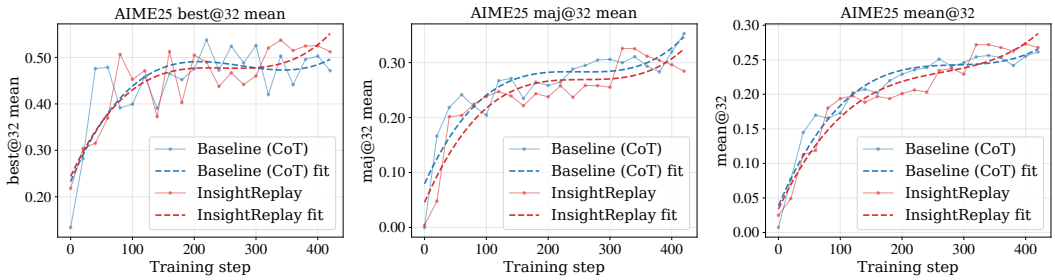


Figure 9: Validation accuracy on AIME 2025 for the alternative training method over a unified 420-step budget. We report best@32, maj@32, and mean@32 using 32 responses per problem, evaluated every 20 training steps. Solid lines show recorded validation values; dashed lines show fitted trends.

Insight-Mode Configuration. In the insight-enabled experiment, rollouts use multi-round generation with up to 5 insight rounds per problem. At each round, the model produces a reasoning step and an intermediate conclusion; all previous reasoning steps and insights remain in context for the next round. The rollout continues until either the maximum number of turns is reached or the model fails to emit a well-formed `<finding>...</finding>` tag.

Baseline Configuration. In the insight-disabled experiment, the model generates a single continuous response per problem without any multi-round structure.

H.6 Controlled Variables

To ensure a fair comparison, both experiments:

- use identical RL hyperparameters;
- are evaluated on the same AIME 2025 benchmark with the same 32-sample validation protocol; and
- differ only in whether the multi-round insight rollout is enabled.

H.7 Experiment Results

Figure 9 summarizes the validation performance of the alternative training method over a unified training budget of 420 steps. As in the main post-training comparison, we track best@32, maj@32, and mean@32 on AIME 2025 throughout training.

The outcome is mixed. By the final checkpoint within the 420-step budget, the multi-round InsightReplay variant exceeds its baseline on mean@32 and best@32, but remains below it on maj@32. Intermediate crossings indicate that the advantage is neither uniform across metrics nor consistent throughout training.

We also emphasize that these experiments were stopped after exactly 420 steps rather than trained further. In addition, this direct multi-round training mode is substantially more expensive operationally. Each InsightReplay rollout must merge multiple rounds into a single training example, and the prompt grows longer at later rounds because more reasoning and insight tokens are carried forward. As a result, token cost increases over the course of a single rollout. For this reason, despite the late crossover on mean@32 and best@32, we ultimately did not adopt this training strategy in the main paper.

I Replay Placement and Serving Cost

I.1 Placement and Prefix-Cache Compatibility

The retained prefix. The core inference protocol in Appendix D continues an existing reasoning trace. After the initial response, the model extracts conclusions from that trace in a separate call. The inference harness retains the reasoning prefix up to the thinking-close marker, appends a replay block, and resumes generation to produce a new thinking-close marker and final answer. Thus, replay is appended to the *retained reasoning prefix*; the previous closing marker and post-thinking answer are replaced by the continuation. Earlier reasoning tokens are preserved, and extracted conclusions occur after the reasoning from which they were derived. Appendix D.6 gives a worked example of this operation.

Cache reuse and its conditions. For a causal transformer, the cached keys and values of an unchanged prefix remain valid when new tokens are appended. Inserting tokens into the middle of that prefix generally requires recomputing the dependent suffix, because its positions or preceding context have changed. Frontier appending therefore permits reuse of the retained prefix without such suffix recomputation. Actual reuse requires identical token IDs, positions, and model configuration, as well as resident cache blocks and server support for prefix caching. The insight-extraction request is a separate model call and can use a different prompt wrapper; its prefill cost is not eliminated merely by making the continuation append-only. Additional work includes extraction, prefill of the replay block, and decoding of the continuation and new answer.

Relation to the distance parameter. The fixed d_0 in Section 3.2 is an idealization of keeping relevant conclusions close to the next reasoning step, rather than an instruction to edit arbitrary earlier positions in the trace. Even at a replay boundary, different tokens in the replay block have different distances from the frontier, and these distances grow during subsequent decoding. The theoretical model explains a potential benefit of restoring recency under its stated assumptions; it does not identify an empirically optimal insertion distance or replay interval. Mid-trace relocation can also change semantic order, so a placement experiment would need to control for both coherence and distance. No insertion-position sweep is reported here.

Scope. This retained-prefix account applies to the core inference experiments. The SWE-Bench agent in Appendix E uses a different history-management variant that replaces conversation history with a summary. Cache and context-retention claims for the core protocol should not be transferred to that variant without separate measurement.

I.2 Batch Wall-Clock Measurements

Measurement protocol. We measure the Base stage and the additional one-round replay stage from timestamped stage boundaries in the experiment driver's run log. Stage time includes request construction, vLLM inference (prefill and decode), retry rounds, and automated grading; one-time server startup is excluded. The additional replay stage includes insight extraction and continuation. Dividing each stage's elapsed time by its number of evaluated trajectories gives seconds per case. There are 16 trajectories per problem: 960 on AIME, 3,168 on GPQA Diamond, and 992 on the LiveCodeBench (LCB) subset. Base and incremental times sum to the end-to-end IR1 time.

Requests are submitted under vLLM's continuous batching with up to 1,000 in-flight requests for the smaller model tier and 128 for the larger tier. These figures therefore measure *throughput-amortized batch time*, not the latency experienced by an individual request. The setup uses 8× NVIDIA H200 GPUs, vLLM [30] v0.19.0, memory utilization 0.92, and prefix caching enabled. Parallelism is TP= 1, DP= 8 for the smaller tier and TP= 2, DP= 4 for the larger tier, as in Appendix D.

Results and interpretation. Across these 18 model–benchmark settings, weighted time increases from 4.30 to 4.98 seconds per case, an overhead of approximately 16%. Cell-level overhead ranges from 9% to 34%. The larger relative overheads on Gemma-4 occur where Base batch times are shorter, consistent with fixed request and extraction costs contributing more to short runs. The 16% overall increase is similar in magnitude to the 14% completion-token increase for IR1 in Table 2. However, that token summary covers 24 cells and four benchmarks, whereas the timing summary

Table 5: Batch wall-clock cost of one-round InsightReplay. All time columns are seconds per case, amortized over concurrently processed trajectories. Incremental time is the additional replay stage after Base. The overall row is weighted by the number of trajectories in each cell. Times and percentages are rounded independently.

Model	Dataset	Base	Incremental	End-to-end	Overhead
Qwen3.5-9B	AIME	3.27	0.46	3.73	14%
	GPQA	1.92	0.41	2.33	21%
	LCB	5.42	0.50	5.92	9%
Gemma-4-E4B-it	AIME	0.31	0.09	0.40	29%
	GPQA	0.17	0.06	0.23	34%
	LCB	0.76	0.17	0.93	23%
R1-Distill-7B	AIME	1.19	0.17	1.36	14%
	GPQA	0.31	0.05	0.36	15%
	LCB	0.53	0.05	0.58	9%
Qwen3.5-35B-A3B	AIME	10.26	1.49	11.75	15%
	GPQA	16.41	2.73	19.14	17%
	LCB	23.70	2.55	26.25	11%
Gemma-4-31B-it	AIME	2.56	0.69	3.25	27%
	GPQA	1.91	0.58	2.49	30%
	LCB	2.41	0.62	3.03	26%
R1-Distill-32B	AIME	6.97	0.93	7.90	13%
	GPQA	2.35	0.31	2.66	13%
	LCB	2.89	0.31	3.20	11%
Overall		**4.30**	**0.68**	**4.98**	**16%**

weights 18 cells across three benchmarks by trajectory count; the two aggregates are not a matched comparison. These timings characterize the measured batch workload, including grading, rather than isolated GPU execution time or production request latency. They do not measure cache-hit rates, peak memory, or IR3 wall-clock overhead.

J Paired Replication and Uncertainty Estimates

J.1 Independent Replication and Resampling Unit

We perform an independent rerun of the three larger-tier models on AIME and the fixed LiveCodeBench subset, using 16 completions per problem and the sampling protocol in Appendix D. Table 6 reports Base and IR1 results from this rerun; it is separate from Table 2, whose point estimates come from the original runs. Within the rerun, each IR1 continuation starts from its corresponding Base sample's reasoning trace.

For a cell with n problems, let $y_{ij}^{(v)} \in \{0, 1\}$ indicate correctness of completion j on problem i under variant v. We first average within each problem and compute the paired difference,

$$\bar{y}_i^{(v)} = \frac{1}{16} \sum_{j=1}^{16} y_{ij}^{(v)}, \qquad d_i = \bar{y}_i^{(\mathrm{IR1})} - \bar{y}_i^{(\mathrm{Base})}, \qquad \widehat{\Delta} = \frac{100}{n} \sum_{i=1}^{n} d_i.$$

Here $\widehat{\Delta}$ is in percentage points (pp). We draw 10,000 bootstrap samples of n problem IDs with replacement and recompute the mean paired difference for each sample. Resampling whole problems keeps the variants paired and the completions clustered within each problem; it avoids treating the $16n$ completions as independent problem-level observations. We report the 2.5th and 97.5th percentiles of the bootstrap distribution as a 95% percentile confidence interval.

We also report the bootstrap tail fraction

$$\widehat{q}_{\le 0} = \frac{1}{10{,}000} \sum_{b=1}^{10{,}000} \mathbf{1}\left\{ \widehat{\Delta}^{*(b)} \le 0 \right\}.$$

This describes the fraction of resampled mean differences that are non-positive. Because the bootstrap distribution is not centered under a null hypothesis, we treat this as a descriptive bootstrap statistic, rather than a calibrated hypothesis-test p-value. The intervals and tail fractions are cell-wise, without multiple-comparison adjustment.

Table 6: Independent paired rerun with 16 completions per problem. Base and IR1 are accuracies (%); Δ and confidence intervals are in pp. $\widehat{q}_{\leq 0}$ is the bootstrap tail fraction defined above. W/T/L counts problem-level wins, ties, and losses. The last two columns count trajectory-level wrong-to-correct and correct-to-wrong transitions. Differences are computed before rounding the displayed accuracies.

Model	Dataset	Base	IR1	Δ	95% CI	$\widehat{q}_{\leq 0}$	W/T/L	W→C	C→W
Qwen3.5-35B-A3B	AIME	92.6	92.6	+0.0	$[-0.3, +0.3]$	0.650	1/58/1	1	1
	LCB	67.3	70.6	+3.2	$[+1.8, +4.7]$	< 0.001	21/40/1	36	4
Gemma-4-31B-it	AIME	86.7	87.8	+1.1	$[+0.5, +1.9]$	< 0.001	10/50/0	12	1
	LCB	81.0	81.1	+0.1	$[-0.2, +0.5]$	0.401	2/59/1	3	2
R1-Distill-32B	AIME	56.2	56.9	+0.6	$[+0.0, +1.5]$	0.042	5/54/1	7	1
	LCB	26.8	31.8	+4.9	$[+2.8, +7.5]$	< 0.001	21/41/0	52	3

J.2 Problem-Level and Trajectory-Level Comparisons

A problem is a win if IR1 has more correct completions than Base among its 16 samples, a loss if it has fewer, and a tie otherwise. The trajectory transition counts compare each continuation with the initial answer from the same reasoning prefix. These two summaries describe different aspects of the effect: wins and losses show how gains are distributed over problems, whereas transitions show how many individual initial outcomes are repaired or degraded. Their net counts also satisfy

$$\widehat{\Delta} = \frac{100}{16n}\big(N_{\mathrm{W}\to\mathrm{C}} - N_{\mathrm{C}\to\mathrm{W}}\big).$$

The largest gains in this rerun occur on LCB for Qwen3.5-35B-A3B (+3.2 pp, CI $[+1.8, +4.7]$) and R1-Distill-32B (+4.9 pp, CI $[+2.8, +7.5]$). These cells have 21 problem-level wins each, against one and zero losses, respectively. Gemma-4-31B-it on AIME has ten wins and no losses, with a +1.1 pp gain and CI $[+0.5, +1.9]$. In contrast, the intervals for Qwen3.5-35B-A3B on AIME and Gemma-4-31B-it on LCB contain zero. The lower endpoint for R1-Distill-32B on AIME rounds to 0.0 pp, so its near-zero boundary should not be interpreted from the rounded interval alone.

On the two largest-gain LCB cells, wrong-to-correct transitions outnumber correct-to-wrong transitions by 36 to 4 and 52 to 3. This establishes a favorable balance of outcome changes for the complete IR1 intervention on those trajectories. Sharing a starting trace does not, by itself, separate the effect of replayed content from the verification cue or additional computation.

Scope of inference. This analysis covers six IR1 cells, not all 24 original settings or IR3. The LCB evaluation uses the same 62 problems selected once uniformly from the 880-problem release for all models and variants; the bootstrap does not replace evaluation on the full release. The intervals summarize variation across observed problems with their sampled completions. They do not separately quantify variation from repeating the entire generation process, and the small number of problems remains a limit even with multiple completions per problem.

K Automated Audit of Insight Faithfulness and Quality

K.1 Sample and Annotation Categories

We audit 100 randomly sampled InsightReplay cases, stratified across six models and three benchmarks (AIME, GPQA Diamond, and LiveCodeBench), using sampling seed 42. The cases contain 395 extracted insights. An external LLM judge (Claude Fable 5) evaluates each insight against its source trace; arithmetic claims are additionally checked computationally. These computational

checks supplement the judge for numeric claims and do not constitute verification of every mathematical, scientific, or programming claim in the trace.

We separate *faithfulness to the source* from *correctness of the source claim*. Each insight receives one of four mutually exclusive labels:

- **Supported**: accurately reflects the source trace and is judged correct.
- **Distorted**: draws on a source claim but misrepresents it, for example by changing a value, dropping a condition, or removing a hedge.
- **Unsupported**: makes a claim not supported by the source trace.
- **Inherited error**: faithfully reproduces a source claim that is itself judged incorrect.

Separately, an insight is flagged as **misleading** if accepting it as fact would likely steer the continuation toward a wrong answer. This flag can overlap any of the four labels; its percentage is not an additional part of their partition.

Table 7: Automated insight audit. Values other than n are percentages of insights within each slice. The four verdict columns are mutually exclusive; misleading is an overlapping flag. Trajectory slices use the correctness of the initial Base answer. Percentages are rounded independently.

Slice	n	Supported	Distorted	Unsupported	Inherited error	Misleading
All	395	90.1	4.1	0.0	5.8	9.9
AIME	147	89.1	5.4	0.0	5.4	12.9
GPQA	113	84.1	3.5	0.0	12.4	14.2
LCB	135	96.3	3.0	0.0	0.7	3.0
Correct Base trajectories	263	97.7	1.9	0.0	0.4	0.8
Incorrect Base trajectories	132	75.0	8.3	0.0	16.7	28.0

K.2 Findings and Implications

Faithfulness does not ensure correctness. Across all insights, 90.1% are labeled supported and 5.8% inherited errors. Thus, most insights reflect their source, but some faithfully repeat incorrect reasoning. Distortion accounts for another 4.1%; extraction can introduce errors even when it does not fabricate a new claim. No unsupported claims or instruction leakage were observed in this audited sample. These zero counts do not establish zero occurrence rates outside the sample.

Association with baseline outcomes. Misleading insights are more frequent in trajectories with incorrect Base answers (28.0%) than in those with correct Base answers (0.8%). Inherited errors show the same pattern (16.7% versus 0.4%). Across benchmarks, LCB has the largest supported fraction (96.3%), whereas GPQA has the largest inherited-error fraction (12.4%). These are descriptive associations: they do not isolate insight quality as the cause of a benchmark-level gain or establish what would happen if an insight were corrected, removed, or corrupted.

Why an erroneous insight can still be useful. The core protocol does not verify or correct an extracted insight before replay. Bringing a conclusion into the local context may make it easier to use or to re-examine. In Appendix D.6, extraction preserves the mistaken evaluation $128-16-4+1=105$; the continuation corrects it to 109. This illustrates a possible correction path, not a measured correction rate. Conversely, a misleading insight can also reinforce an error. Retaining the reasoning trace provides an opportunity to recheck a conclusion against its source, but does not guarantee that the model will do so.

Audit limits. These estimates depend on automated judgments and include several insights from the same case. They should not be interpreted as human-validated labels or as 395 independent trajectory observations. The audit does not provide a controlled perturbation study of insight quality, or a round-by-round estimate of correction versus propagation over IR3. Such measurements would be needed to quantify how extraction errors affect later replay rounds.

L Supplementary Mechanistic Discussion and Positioning

L.1 Distance Sensitivity and a Random-Span Control

The accessibility hypothesis concerns the use of information that is still present in context. Replaying a compact conclusion close to the next decoding position can reduce the need to retrieve that conclusion from its original distant occurrence. It does not require equally strong attention to both occurrences, although the retained trace remains available for details and verification. Whether a particular conclusion is used depends on its content and the current query, in addition to its distance.

As a supplementary control to Section 2.2, we measure scores to randomly selected token spans under the same filler-insertion setting. On Gemma-4-31B-it's final global-attention layer, these spans have lower average pre-softmax scores and a non-monotonic response to insertion ratio, whereas the critical-insight scores decrease monotonically, with a 3.3% decline from $\rho = 0$ to $\rho = 0.4$. Here $\rho = 0$ means no *additional* filler; the original insight-to-answer distance can already be large. The supplementary random-span comparison is reported qualitatively; we do not report a confidence interval or significance test for the difference between the two types of spans.

This control suggests that the observed trend need not hold uniformly for arbitrary token spans. It does not establish that all critical content obeys the same decay law, nor does it directly measure attention to replayed copies during free generation. Pre-softmax scores are diagnostics of query–key interactions, rather than normalized attention mass or causal attribution scores.

What the existing content controls establish. Table 1 distinguishes preserving the trace from replacing it with a short summary. Full trace plus insights yields answer probability 0.545–0.557, compared with 0.512 for the full trace alone and 0.387 for insights alone. Conditions F/G replace the *original reasoning trace* with length-matched random tokens while retaining the insights; their probabilities are 0.131 and 0.127. These controls support the usefulness of the original trace and its complementarity with insights. They are not an experiment that replaces only the replayed insight block with random content while holding the trace fixed. Together with the trajectory transitions in Appendix J, they motivate further controlled tests of which replayed conclusions affect the final answer.

L.2 Relationship to Context Management and Refinement

InsightReplay shares summary generation, information selection, and revision with several existing approaches. The useful distinctions are the source of the replayed content, whether the original trace is retained, and the point at which the continuation is resumed. Table 8 summarizes these distinctions for the core inference protocol, without treating the method families as mutually exclusive.

Budget forcing and native replay behaviors. The VO cue is related to the budget-forcing strategy in s1 [36]: it prompts the model to continue thinking without an explicit extracted-insight block. It is a control inspired by that idea, rather than a reproduction of s1's complete training and inference recipe. Any spontaneous restatement or self-checking already present in the model contributes to Base performance. Comparisons with VO assess the additional structured intervention, but do not measure how often native replay occurs or attribute every accuracy difference to a single component of the intervention.

The limited LSTM analogy. The connection to LSTMs [9] is functional: maintaining an explicitly updated state alongside a growing sequence. The architectural mechanisms differ. A recurrent model compresses its history into a fixed-dimensional state, whereas a transformer can retain earlier tokens in its context and still fail to use them effectively. For InsightReplay, distance-dependent query–key interactions and context utilization are the more direct motivation. The evidence in Sections 2.1–2.2 concerns this use of retained information, rather than loss of information from a recurrent hidden state.

Table 8: Operational positioning of the core retained-trace protocol. These comparisons describe mechanisms, not matched empirical rankings.

Related family and examples	Relationship to InsightReplay
Context summarization / restart: InftyThink, PENCIL [14, 15]	Summaries can replace earlier reasoning to limit retained context. Core InsightReplay appends conclusions while preserving the source trace.
CoT compression: TokenSkip, C3oT [10, 11]	Compression shortens reasoning; replay adds a compact representation of selected conclusions. The operations can be combined, but that combination is not evaluated here.
Scratchpad revision: improvement operators [16]	Rewriting a working state and keeping an auxiliary state alongside a trace are related designs. Core InsightReplay uses the latter.
Self-reflection: Reflexion, Self-Refine [32, 33]	These methods use feedback to guide revision. InsightReplay also uses a verification cue, but explicitly supplies conclusions extracted from the retained reasoning as material for the continuation.
Retrieval and replay: RAG [34]	Retrieval can supply evidence from an external store. Core InsightReplay re-exposes conclusions already contained in the model's own current trajectory.
Agent memory management: MemGPT [35]	Memory management moves information between a bounded context and external storage. Core InsightReplay addresses access within retained context; the SWE-Bench variant also performs history compression.

L.3 Budget Comparisons and Scope of the Mechanism

Completion costs versus fixed budgets. In Table 2, VO and IR1 use similar average completion-token totals ($1.12\times$ and $1.14\times$ Base), with macro accuracy gains of $+0.61$ and $+0.90$ pp. This is a closer cost comparison than IR3 versus VO: IR3 uses $1.36\times$ Base tokens, so its larger $+1.65$ pp gain also involves more computation. Similar average completion counts are not identical per-problem context limits or total serving costs. Likewise, the length curves in Appendix F bin sampled trajectories by realized length; they are descriptive and are not a randomized experiment with a common enforced token budget.

The post-training comparison in Appendix G does enforce the same maximum response length of 30,720 tokens. The injected cue and continuation share that cap, and cue tokens have $\mathrm{response_mask} = 0$, excluding them from the policy loss. Equal maximum length controls the permitted rollout budget, but does not imply equal realized token use or training wall-clock cost.

Where replay may help. The theoretical model separates reasoning competence, represented by $p(N)$, from accessibility, represented by Φ. This motivates a distinction between failing to derive a useful conclusion and failing to use one already derived. Replay adds no external solution, but may help the model recover or reconsider its own conclusions. The categories are not directly observed from final-answer accuracy alone: even an incorrect trajectory can contain useful intermediate results, and an erroneous conclusion can sometimes be corrected on continuation. Consequently, benchmark difficulty or baseline accuracy alone does not identify the cases that will benefit. The audit in Appendix K documents both useful content and error risks, without establishing a causal decomposition of the gains.

Complementary efficiency methods. Retaining a trace and adding replay content increases token use and can increase memory and latency. Compression, selective continuation, and multi-sample aggregation offer other trade-offs; their effectiveness depends on the task and budget. Replay could be combined with such methods, but the present experiments do not establish dominance over them or a universal minimum cost for alternative test-time scaling.