

Reflection Is Not Monolithic: Experience-Driven Control of Recheck in LLM Reasoning

Quanyu Long¹ Jiang Kai Jie¹ Jianda Chen¹ Qisheng Hu¹ Xu Guo¹
Leilei Gan² Wenya Wang¹

¹Nanyang Technological University, Singapore

²Zhejiang University

quanyu001@e.ntu.edu.sg

Abstract

Large Reasoning Models (LRMs) achieve strong performance by generating long reasoning traces with explicit reflection. However, reflection is not functionally uniform: some reflections revise the reasoning strategy, while others verify previously derived intermediate results. In this work, we examine whether local self-verification can be controlled selectively without suppressing useful reasoning. Through a large-scale analysis of mathematical reasoning traces, we distinguish strategy-level rethink from local self-verification recheck, and further separate rechecks into corrective and confirmatory cases. We find that recheck occurs frequently across models and benchmarks, but most observed rechecks are confirmatory, rarely uncovering errors or changing the reasoning trajectory. Motivated by this finding, we propose Experience-Driven Suppression (EDS), a conservative test-time controller that suppresses a recheck only when historical outcomes of similar episodes suggest low utility. Unlike early-exit or trigger-suppression methods that stop reasoning globally or suppress reflection cues broadly, EDS targets local recheck while preserving later rethink. Experiments show that EDS yields moderate reasoning-length reductions with limited accuracy changes, supporting behavior-specific rather than monolithic reflection control.

1 Introduction

Large language models (LLMs) have recently shown strong performance on complex reasoning tasks. Large Reasoning Models (LRMs), such as OpenAI o1 (OpenAI, 2024) and DeepSeek-R1 (DeepSeek-AI et al., 2025), further extend these capabilities through reinforcement learning (RL), producing long reasoning traces with increasingly explicit reflection and self-correction (Yang et al., 2025b). A useful perspective is that RL does not introduce reflection from scratch, but amplifies it:

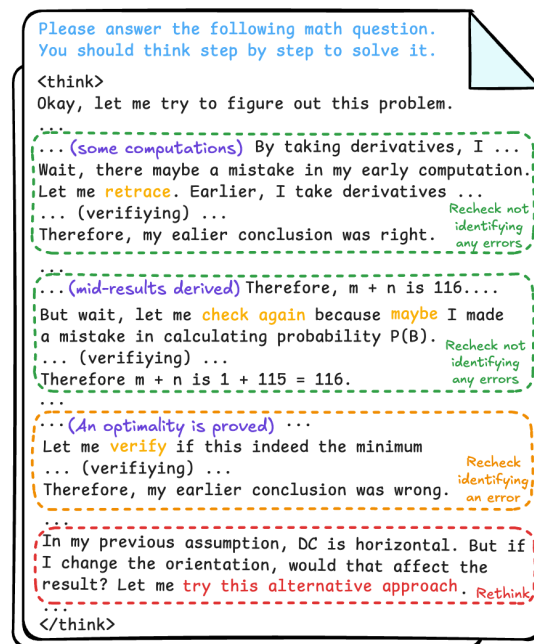


Figure 1: Reflective behaviors commonly observed in mathematical reasoning. We distinguish two functional types: **rethink**, where the model revises its strategy and explores an alternative line of reasoning; and **recheck**, where the model verifies already-derived intermediate results through re-computation, arithmetic checking, or optimality validation. Recheck can be further divided into **confirmatory** cases, which merely reaffirm earlier results, and **corrective** cases, which identify an error and change the subsequent reasoning. Our empirical analysis shows that recheck is frequent, yet overwhelmingly confirmatory.

pretrained LLMs may already exhibit occasional reflective behavior, though often shallow and ineffective (Liu et al., 2025b), whereas RL substantially increases both its frequency and apparent utility by optimizing task-level rewards. Reflection therefore appears to be an important mechanism behind the reliability of LRMs. At the same time, it also introduces inefficiency, as models continue deliberating and repeatedly verifying intermediate results even when the benefit is small, a phenomenon often described as overthinking (Sui et al., 2025).

While overthinking has been increasingly studied (Zhang et al., 2025b; Fang et al., 2025; Jiang et al., 2025; Zhang et al., 2026), existing work often treats reflection as a coarse behavior. We argue that reflection is functionally heterogeneous. As shown in Figure 1, reflective behaviors can be broadly separated into two types: **rethink**, where the model revises its reasoning strategy or reformulates the problem, and **recheck**, where the model verifies already-derived intermediate results through recomputation, arithmetic checking, constraint validation, or optimality testing. This distinction matters because suppressing rethink may remove useful strategy revision, whereas some rechecks may only repeat local verification.

Through large-scale analysis of reasoning traces, we find that recheck occurs **frequently** across models and datasets (Section 3.2), but most observed rechecks are **confirmatory**: they rarely identify errors or change the reasoning trajectory (Section 3.3). This finding is related to recent work showing that many reflective steps are redundant, self-affirming, or decorative (Kang et al., 2025; Liu et al., 2025a; Zhang et al., 2025a; Zhao et al., 2026). However, our contribution is not the broad observation that reflection can be redundant. Instead, we provide a more fine-grained functional account of which kind of reflection is redundant. Rather than studying post-answer reflection, broad self-affirmation, hidden-state correctness signals, or aha-style decorative thinking, we focus on *in-process self-verification* over previously derived intermediate results. This granularity allows us to distinguish strategy-level *rethink* from local *recheck*, and to measure whether a verification step actually changes the reasoning trajectory.

In this work, we examine whether local self-verification can be controlled selectively without suppressing useful reasoning. This question is different from asking whether a model should stop reasoning altogether: our goal is to decide whether a specific local verification step is worth continuing while still allowing later strategy revision. Since rechecks are not uniformly redundant, simply eliminating verification would be risky. Instead, effective reasoning should be selective: verification should be preserved when it is likely to catch errors, but reduced when similar checks have rarely changed the outcome in the past. Rather than treating self-verification as an unconditional safeguard, we view it as a behavior that can be regulated by experience.

Motivated by this perspective, we propose Experience-Driven Suppression (EDS), a conservative test-time controller for selective self-verification. EDS detects when the model initiates a recheck, retrieves similar verification episodes from an offline experience pool, and suppresses the current recheck only when historical outcomes suggest low utility. Unlike early-exit or trigger-suppression methods that stop reasoning globally or suppress reflection cues broadly, EDS intervenes only on local recheck while preserving the opportunity for later rethink. Empirically, EDS yields moderate reasoning-length reductions with small, model-dependent accuracy changes, supporting the value of behavior-specific control over monolithic reflection suppression.

2 Related Work

Efficient Reasoning and Reflection Control Recent work improves reasoning efficiency by adapting how much computation a model uses. Training-time methods teach models to switch between short and long reasoning, select reasoning formats, or optimize for shorter traces through length-aware rewards and process-level control (Zhang et al., 2025b; Fang et al., 2025; Jiang et al., 2025; Zhang et al., 2025c, 2026; Wu et al., 2025; Wang et al., 2025). Inference-time methods instead reduce computation without updating model parameters, using adaptive budgets, convergence criteria, post-answer truncation, early exit, or confidence-guided suppression (Shen et al., 2025; Liu and Wang, 2025; Mao et al., 2025; Kang et al., 2025; Sun et al., 2025; Yang et al., 2026; Huang et al., 2026). System-oriented models also expose practical reasoning toggles for high-throughput deployment (Bercovich et al., 2025).

Most closely related to our work are methods that directly suppress reflective reasoning. Liu et al. (2025a) suppress self-affirmation reflections by controlling reflection-leading tokens, while Huang et al. (2026) suppress reflection triggers when the model is sufficiently certain. Early-exit methods such as DEER (Yang et al., 2026) and stopping methods such as REFRAIN (Sun et al., 2025) further reduce reasoning by terminating the trajectory once additional reasoning appears unnecessary. These methods are effective for token reduction, but they operate through trajectory-level stopping, confidence-guided suppression, or generic reflection-trigger control. They do not explicitly

distinguish whether the upcoming reflection is a useful strategy revision or a redundant local verification. In contrast, our goal is not maximal compression, but behavior-specific control: we distinguish *rethink* from *recheck* and conservatively suppress only rechecks whose similar historical episodes suggest low utility.

Analyzing Reflective Reasoning Several recent studies question whether reflective steps in reasoning traces are genuinely useful. Kang et al. (2025) show that reflection after a candidate final answer often fails to change the answer. Liu et al. (2025a) identify self-affirmation reflections as a major source of inefficiency. Zhang et al. (2025a) use chunk-level annotation and hidden-state probes to study whether reasoning models internally encode correctness or self-verification status. Zhao et al. (2026) examine whether apparent “aha moments” are genuinely corrective or merely decorative. These works show that reflection is not uniformly useful.

Our analysis is complementary but more fine-grained. Rather than treating reflection, self-affirmation, or aha-style behavior as broad categories, we separate strategy-level *rethink* from local self-verification *recheck*. We further distinguish corrective rechecks from confirmatory rechecks, allowing us to measure whether a verification step changes the reasoning trajectory. Thus, our contribution is not simply to show that reflection can be redundant, but to identify a specific redundant subtype and examine whether it can be selectively controlled without suppressing useful reasoning.

3 Empirical Analysis

Analyses are conducted on rollouts generated by four LLMs, Qwen3-8B (Yang et al., 2025a), QwQ-32B (Team, 2025), gpt-oss-20b (OpenAI, 2025), and DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI et al., 2025). We evaluate models on four math benchmarks: AIME24 (MAA, 2024), AIME25 (AIME, 2025), AMC23 (AMC12, 2023) and MATH500 (Hendrycks et al., 2021).

3.1 How frequently do LLMs exhibit reflective behavior?

Recent reasoning-oriented LLMs generate traces that often include reflective behavior. Such reflection is commonly viewed as a mechanism for improving reasoning reliability, especially in models trained to deliberate extensively. To understand

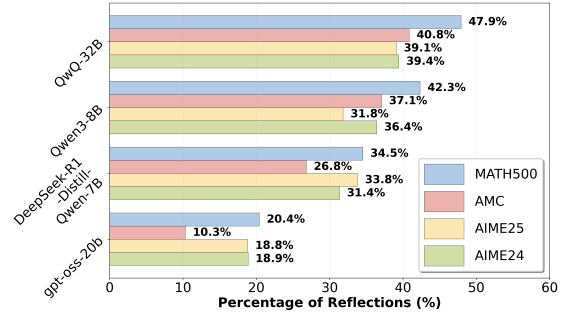


Figure 2: Percentage of steps classified as reflections.

its practical role, we begin by measuring how often reflective behavior actually appears in model-generated reasoning traces.

We analyze reasoning rollouts at the granularity of individual steps. Each rollout is segmented into a sequence of steps using the formatting cue “\n\n”. Each step is then annotated as reflective or non-reflective using GPT-5 (OpenAI, 2026). A step is labeled as reflective only if it explicitly refers to the model’s own reasoning process and engages in monitoring, evaluating, or revising that reasoning. Concretely, reflective steps include explicit self-referential or metacognitive language indicating verification, critique, reconsideration, or strategy switching (e.g., “I should check”, “this approach might be wrong”, “let me reconsider”). In contrast, steps that primarily advance the solution, such as carrying out calculations, applying formulas, or executing problem-specific logic, are not considered reflective. The exact prompts are provided in Appendix F. To assess the reliability of this annotation pipeline, we additionally conduct human evaluation and report the agreement results in Appendix C. Based on these annotations, we compute the proportion of reflective steps in each rollout; the results are shown in Figure 2.

Findings. Across all reasoning-oriented models and benchmarks, reflective steps account for a substantial portion of model-generated rollouts, often approaching or exceeding one third of all steps. Reflection is therefore not an occasional behavior, but a systematic component of LLM reasoning.

A natural intuition is that reflection should increase with problem difficulty: harder problems would seem to require more reconsideration, exploration, and strategic adjustment. However, our results reveal a different pattern. For all models, MATH500 exhibits the highest proportion of reflective steps, despite being generally less difficult than

AIME. This suggests that reflection frequency does not reliably track problem difficulty or the need for deeper reasoning. Instead, reflection appears to be triggered broadly, even in settings where extensive deliberation is unlikely to provide substantial benefit. This motivates a closer examination of *what kinds* of reflection models actually perform.

3.2 What Forms of Reflection do LLMs repeatedly exhibit?

Not all reflective steps serve the same function. Some reflections revise the path and help the model escape an unproductive line of reasoning, while others merely verify previously derived results. Distinguishing between these functional roles is important for understanding whether reflection primarily advances reasoning or instead acts as local self-monitoring.

Rethink vs. Recheck. We introduce a functional taxonomy that separates reflective steps into two categories: **rethink** and **recheck**. **Rethink** operates at the level of global reasoning strategy. Rather than evaluating a specific intermediate result, it revises assumptions, introduces alternative solution paths, abandons earlier plans, or reorganizes the overall reasoning process. Such reflections are exploratory in nature and are often necessary when the initial approach is flawed or incomplete. **Recheck** (i.e., **self-verification**), in contrast, operates at the level of local correctness. It focuses on verifying previously stated intermediate results, computations, or logical steps, with the explicit or implicit goal of assessing whether an earlier conclusion is valid. Rechecks may involve recomputation, constraint checking, case analysis, or validation of optimality or minimality. Crucially, rechecks do not introduce a new solution strategy; they assess an existing one. To operationalize this taxonomy, for each step identified as reflective in Section 3.1, we prompt GPT-5 with surrounding context, including preceding and subsequent steps, enabling it to assess the role that the local reflection plays in the overall reasoning trajectory. Each reflective step is then categorized as either a rethink or a recheck, with ambiguous cases labeled as unable to classify.

Findings. Figure 3 shows the distribution of reflection types. Across all datasets, recheck constitutes a substantial fraction of reflective behavior, accounting for roughly 40-58% of all reflections. Notably, recheck makes up a larger share on comparatively simpler datasets such as AMC

and MATH500. This pattern clarifies the finding from the previous subsection. There, we show that reflection is not more prevalent on harder problems. Here, we further see that on easier problems, a considerable portion of reflection is not strategy revision, but local self-verification. In other words, the excess reflection on simpler datasets is driven less by strategy-level rethink and more by repeated rechecking of intermediate results. This naturally raises the next question: do these rechecks actually uncover errors?

3.3 Do Rechecks Truly Identify Errors?

If rechecks frequently detect and correct mistakes, then their prevalence would support self-verification as an important component of reliable multi-step reasoning. If, however, most rechecks merely reaffirm already-correct intermediate results, then a substantial portion of reflective behavior may have low observed corrective utility. We therefore examine whether rechecks meaningfully contribute to error correction in practice.

Corrective vs. Confirmatory Rechecks. We further categorize rechecks by their functional outcome. A **corrective recheck** identifies an error in a previously stated intermediate result and leads to a correction. In contrast, a **confirmatory recheck** verifies an earlier step but does not uncover any error and does not change the intermediate result. The resulting distribution is shown in Figure 3.

Findings. We find that the overwhelming majority of rechecks are confirmatory rather than corrective. Across datasets, approximately 80–95% of rechecks merely confirm prior reasoning and do not alter intermediate results. This pattern holds consistently across both easier and harder benchmarks. These results indicate that although models frequently engage in self-verification, such behavior only rarely leads to actual error correction.

3.4 Why Does Confirmatory Rechecking Become Dominant?

To further probe how confirmatory rechecking evolves during reasoning-oriented training, we analyze reflective behaviors across several intermediate GRPO checkpoints. This analysis is not intended to establish a causal mechanism, but to provide checkpoint-level evidence on whether recheck behavior changes as RL training progresses. Under outcome-based RL, models are rewarded for final correctness, while intermediate verification

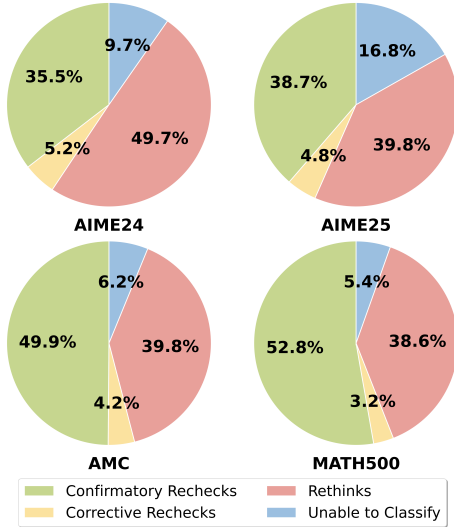


Figure 3: Reflective behavior is frequent across benchmarks, and a substantial portion of it consists of confirmatory recheck rather than rethink. Results are based on GPT-5 annotations with detailed prompts described in Appendix F. Although these labels are produced automatically for scalability, the annotation criteria are functionally explicit, and their quality is further checked through **human evaluation with high agreement rate** (Appendix C). Additional analysis results are provided in Appendix B.

steps are not directly supervised. In such a setting, rechecking may be reinforced as a generally useful behavior when it occasionally helps recover from errors, even if many individual rechecks only confirm already-correct intermediate results.

Type	Step 220	Step 320	Step 410
Corrective Recheck	100	132	112
Confirmatory Recheck	595	732	768
Rethink	787	907	1065
Unable to Classify	59	98	76

Table 1: Distribution of reflective behavior types across GRPO checkpoints.

Table 1 shows a pattern consistent with this view. Across checkpoints, confirmatory rechecks are consistently much more frequent than corrective rechecks, and their count increases from Step 220 to Step 410. Corrective rechecks, in contrast, remain comparatively rare and do not show a similarly clear increase. This suggests that training may amplify the overall tendency to verify, without necessarily making verification more selectively corrective. We treat this result as suggestive rather than conclusive: the checkpoints do not isolate the effect of RL from other training dynamics, but they sup-

port our broader observation that self-verification is frequently invoked beyond the cases where it changes the reasoning trajectory.

4 Experience-Guided Self-Verification

The preceding analysis shows that reflection is heterogeneous: *rethink* may revise the reasoning strategy, while *recheck* often performs local self-verification over previously derived results. Since rechecks are not uniformly redundant, our goal is not to remove verification altogether, but to examine whether low-utility rechecks can be selectively controlled without suppressing useful reasoning.

Motivated by this goal, we introduce **Experience-Driven Suppression (EDS)**, a simple test-time controller for selective self-verification. Unlike early-exit or trigger-suppression methods that stop reasoning globally or suppress reflection cues broadly (Liu et al., 2025a; Huang et al., 2026; Sun et al., 2025), EDS intervenes only on local recheck behavior. It uses past verification outcomes as an external signal: if similar rechecks have rarely corrected errors before, the current recheck is likely to have low observed corrective utility. EDS is therefore intended as a conservative instantiation of behavior-specific control, rather than a method for maximal token compression.

4.1 Experience Memory Construction

We first construct an experience pool that records historical outcomes of self-verification episodes, as illustrated in the left panel of Figure 4. Given reasoning rollouts, we identify sentences that initiate *recheck* behavior, such as recomputation, constraint checking, candidate testing, or optimality validation. We intentionally exclude *rethink* behaviors, since our goal is to control local verification rather than strategy revision. For each detected recheck, we extract a local context window containing the reasoning state and subsequent verification steps, and annotate whether the recheck is **corrective**, i.e., it identifies an error and leads to a correction of an intermediate result, or **confirmatory**, i.e., it verifies the prior result without changing the reasoning trajectory.

Each annotated episode is stored as an experience unit. Formally, the experience pool is $\mathcal{C} = e_1, e_2, \dots, e_n$, where each unit is $e_i = (c_i, y_i)$. Here, c_i denotes the local self-verification context and $y_i \in \{0, 1\}$ indicates whether the recheck is corrective ($y_i = 0$) or confirmatory ($y_i = 1$). Thus, $\mathcal{C}$

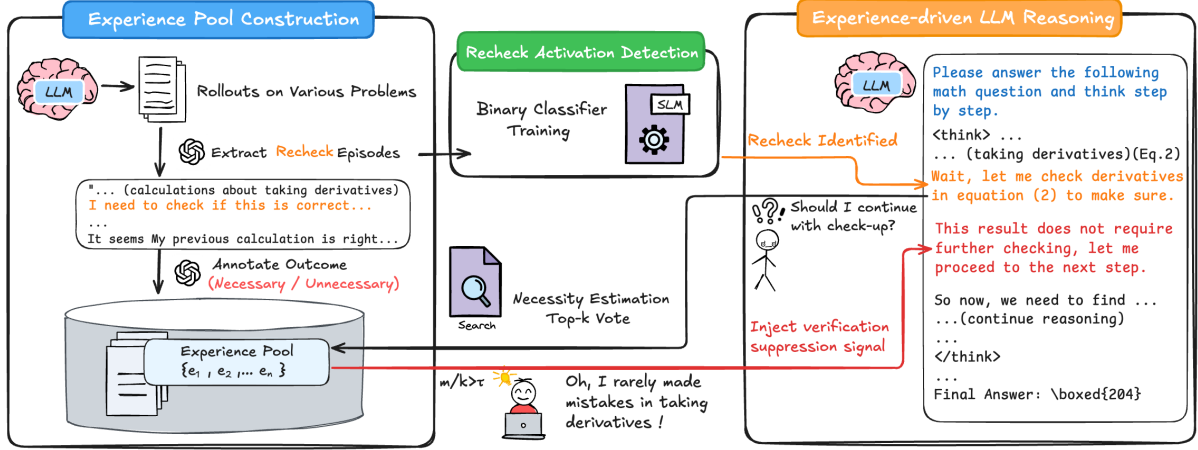


Figure 4: Overview of Experience-Driven Suppression (EDS). Recheck episodes are collected and annotated offline to form an experience pool of historical verification outcomes. At test time, when the model initiates a recheck, EDS retrieves similar past episodes and suppresses the current verification only when prior outcomes suggest low utility.

serves as an external memory of historical verification outcomes organized around local reasoning patterns. Details of data construction, annotation, and filtering are provided in Appendix A.

4.2 Experience-Enhanced Inference

At inference time, EDS controls recheck behavior through three steps: detecting a recheck, estimating its utility from past episodes, and suppressing it only when the evidence suggests low utility.

Recheck Detection. As the model generates a reasoning trace, a lightweight identifier monitors each newly generated sentence and predicts whether it initiates a recheck of a previously derived intermediate result. We use an external identifier rather than asking the backbone model to emit explicit markers, since we find such self-marking unreliable inside the `<think>` trace. This design keeps EDS external to the reasoning model and avoids changing the model’s generation format. Implementation details of the identifier are provided in Appendix A.

Experience Retrieval and Utility Estimation. When a recheck is detected, EDS forms a query from the local context preceding the recheck and retrieves the top- k most similar experience units from $\mathcal{C}$. We use BM25 (Robertson and Zaragoza, 2009) because lexical overlap is often effective for matching symbolic expressions and local mathematical operations. At the same time, BM25 is only a simple instantiation of experience retrieval rather than a claim of retrieval optimality.

Let the retrieved units be $\{e_{i_1}, \dots, e_{i_k}\}$, where

each $e_{i_j} = (c_{i_j}, y_{i_j})$. Instead of feeding retrieved texts back to the LLM, which would introduce extra token cost, EDS aggregates only their outcome labels:

$$\mathcal{P}_{\text{conf}} = \frac{1}{k} \sum_{j=1}^k \mathbb{I}[y_{i_j} = 1], \quad (1)$$

where $\mathbb{I}[\cdot]$ is the indicator function. $\mathcal{P}_{\text{conf}}$ measures the proportion of retrieved episodes whose rechecks were confirmatory. If $\mathcal{P}_{\text{conf}} > \tau$, historical experience suggests that the current recheck has low observed corrective utility, and EDS applies the suppression intervention. The threshold τ controls intervention aggressiveness: larger values require stronger historical evidence before suppression, while smaller values allow more aggressive length reduction. In our experiments, we use top-30 retrieval and set $\tau = 0.8$.

Selective Suppression. When a recheck is estimated to be low utility, EDS injects a short forward-directing signal immediately after the recheck sentence, e.g., “*This result does not require further checking; let me proceed to the next step.*”¹. The signal does not provide new mathematical content; it only discourages continuing the current verification and redirects generation to the next reasoning step. After the intervention, generation proceeds

¹We observe that the specific wording of the verification suppression signal affects its effectiveness. Language models exhibit discourse-level attractors for resolving epistemic uncertainty (e.g., phrases such as “wait” or “maybe”), which can cause verification behavior to persist even after an initial interruption. In practice, suppression signals that explicitly assert epistemic closure (e.g., “is correct”) or provide a forward redirection (e.g., “proceed”) are more effective at closing verification-related discourse.

normally, so the model can still continue reasoning and perform later *rethink* when needed. To avoid repeatedly interrupting the same verification span, we apply a short cooldown: after each suppression, recheck detection is disabled for the next three generated steps.

5 Experiments

5.1 Experiment Setups

We evaluate the effectiveness of Experience-Driven Suppression (EDS) on five mathematical reasoning benchmarks: AIME24 (MAA, 2024), AIME25 (AIME, 2025), AMC23 (AMC12, 2023), MATH500 (Hendrycks et al., 2021), and Olympiad Bench (He et al., 2024). Experiments are conducted on three models: Qwen3-8B (Yang et al., 2025a), QwQ-32B (Team, 2025), and DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI et al., 2025).

We compare EDS with several inference-time reasoning-control baselines. **Base** denotes standard model inference without intervention. **FTM** (Kang et al., 2025) represents answer-based truncation, which stops reasoning after the first candidate final answer is produced. **DEER** (Yang et al., 2026), **LTCGS** (Liu et al., 2025a), and **CGRS** (Huang et al., 2026) represent confidence-controlled reasoning reduction methods. These methods monitor reasoning transition points or reflection-trigger tokens, such as “wait”, and use model-internal confidence or certainty signals to decide whether to truncate subsequent reasoning or suppress reflection-leading tokens.

These baselines are effective but operate through relatively coarse control. FTM and DEER intervene at the trajectory level by stopping the remaining reasoning once their exit criteria are met. LTCGS and CGRS are more local, but still act on generic reflection triggers rather than on the functional role of the upcoming reflection. Consequently, they may reduce more tokens, but can also suppress useful strategy-level *rethink* together with confirmatory or corrective local *recheck*. In contrast, EDS adopts a more conservative behavior-level intervention.

To isolate the role of recheck-aware control, we further consider two recheck-specific settings. **Full-suppress**: an aggressive variant that injects a suppression signal whenever a recheck activation is identified. **EDS**: our method, which acts as a simple nonparametric controller by selectively suppressing detected rechecks only when retrieved his-

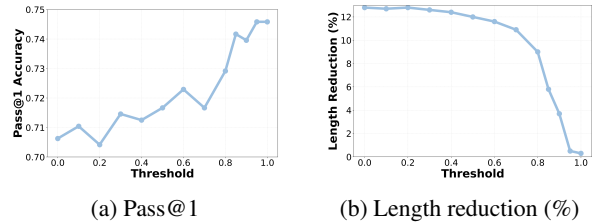


Figure 5: Accuracy–efficiency trade-off.

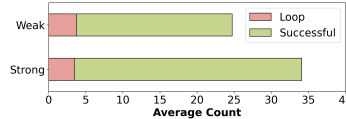


Figure 6: Average number of recheck triggers after weak and strong suppression signals, separated by successful suppression and loop cases.

torical experience suggests low observed corrective utility. Setups of EDS can be found in Appendix A.

5.2 Results

Table 2 reports the main comparison on Qwen3-8B, with additional model results in Appendix D. Rather than pursuing maximal token compression, **our goal is to examine whether local self-verification can be controlled selectively without suppressing useful reasoning**.

The results reveal a trade-off between intervention aggressiveness and accuracy preservation. Truncation-based methods such as FTM and DEER, as well as trigger-suppression methods such as LTCGS and CGRS, can substantially reduce reasoning length. However, because they operate at the trajectory level or on generic reflection triggers, they do not distinguish local recheck from potentially useful rethink, and several settings incur larger accuracy degradation. For instance, DEER reduces length by over 30% but incurs more than 3 points of accuracy loss, while more aggressive CGRS settings also show larger degradation.

A similar pattern appears within recheck-aware control. Full-suppress intervenes only on detected rechecks, but suppressing all rechecks still lowers average accuracy by 1.39 points, indicating that some rechecks should be preserved. EDS is more conservative: it suppresses a recheck only when retrieved historical episodes suggest low observed corrective utility. As a result, EDS reduces average reasoning length by 8.6% with a +0.48-point average accuracy change. These results support the central motivation of EDS: behavior-specific control can reduce part of repeated local verification

Method	Dataset	Accuracy (%)			Length		
		Base	Full-suppress	EDS	Base	Full-suppress	EDS
Ours	AIME24	74.58	70.63 (-3.95)	72.92 (-1.66)	14605	12734 (-12.8%)	13296 (-9.0%)
	AIME25	67.71	66.67 (-1.04)	70.00 (+2.29)	17133	15713 (-8.3%)	16086 (-6.1%)
	AMC	95.62	96.25 (+0.63)	98.75 (+3.13)	8091	6564 (-18.9%)	6893 (-14.8%)
	MATH500	95.80	95.20 (-0.60)	97.20 (+1.40)	4939	3935 (-20.3%)	4110 (-16.8%)
	Olympiad Bench	80.42	79.53 (-0.89)	79.82 (-0.60)	10480	9540 (-9.0%)	9739 (-7.1%)
	Average	80.95	79.56 (-1.39)	81.43 (+0.48)	11361	10071 (-11.4%)	10386 (-8.6%)
FTM	Base	Base	Truncate all	Selective truncate	Base	Truncate all	Selective truncate
	Average [†]	84.70	80.90 (-3.80)	81.80 (-2.90)	15125	10601 (-29.9%)	11414 (-24.5%)
LTCGS	Base	Base	Threshold 0.9	Threshold 0.3	Base	Threshold 0.9	Threshold 0.3
	Average	80.95	78.38 (-2.57)	80.90 (-0.05)	11361	9355 (-17.7%)	10977 (-3.4%)
DEER	Base	Base	DEER	DEER Pro	Base	DEER	DEER Pro
	Average	80.95	76.40 (-4.55)	77.19 (-3.76)	11361	7294 (-35.8%)	7760 (-31.7%)
CGRS	Base	Base	threshold 0.5	threshold 0.9	Base	threshold 0.5	threshold 0.9
	Average	80.95	75.56 (-5.39)	80.27 (-0.68)	11361	7317 (-35.6%)	8317 (-26.8%)

Table 2: Accuracy (Pass@1) and average reasoning length on Qwen3-8B. We report our recheck-aware settings, including Base, Full-suppress, and EDS, together with inference-time reasoning-reduction baselines FTM, LTCGS, DEER, and CGRS. Numbers in parentheses indicate changes relative to the corresponding Base setting. [†] indicates numbers reported in Kang et al. (2025). Results on additional models are provided in Appendix D.

while preserving later reasoning.

5.3 Accuracy-Efficiency Trade-off

Figure 5 illustrates how the behavior of EDS varies as we adjust the voting threshold τ , which controls how strongly retrieved experience must favor a confirmatory outcome before suppression is applied. Increasing τ makes suppression more conservative, requiring stronger historical evidence of low observed corrective utility before intervention.

As shown in Figure 5, accuracy increases steadily with larger τ and saturates around $\tau \approx 0.95$, as higher thresholds reduce the likelihood of suppressing rechecks that could be corrective. In contrast, length reduction decreases as τ increases. In particular, when τ lies in the range $0.7 \leq \tau \leq 0.9$, length reduction drops rapidly. Together, these trends reveal a clear accuracy-efficiency trade-off: lower thresholds favor stronger suppression and greater reductions in reasoning length, while higher thresholds preserve more verification behavior and yield higher accuracy. This trade-off allows EDS to be flexibly configured according to the desired balance between token efficiency and reasoning reliability.

5.4 Case Study

To qualitatively illustrate how verification suppression affects model behavior, Figure 7 presents representative examples under three settings: no intervention, strong suppression, and weak suppression.

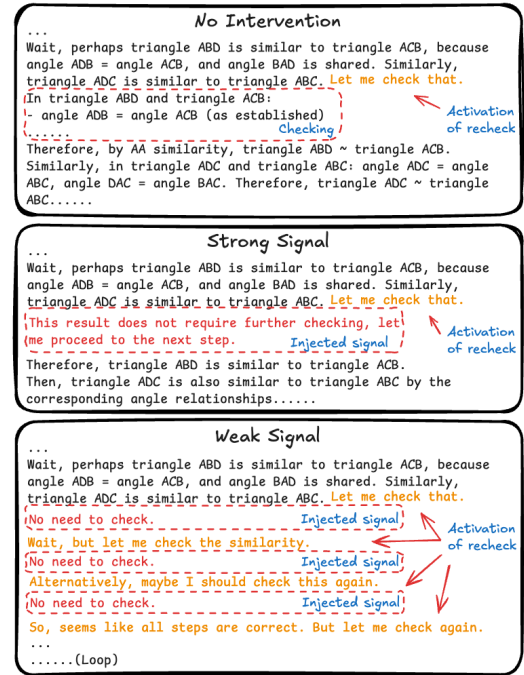


Figure 7: Success and failure cases.

Without intervention, once a recheck is triggered, the model typically proceeds with explicit verification. Injecting a strong suppression signal that asserts epistemic closure and provides forward redirection is more effective at terminating the current verification process. The model accepts the signal as closing the uncertainty and continues with subsequent reasoning. In contrast, weak suppression signals fail to close the verification discourse and may instead provoke repeated recheck attempts,

leading to activation loops in which verification is repeatedly re-initiated.

Figure 6 shows that strong signals achieve a higher rate of successful suppression, although looping still occurs. The cooldown mechanism described in Section 4.2 helps prevent such immediate re-activation of verification.

6 Conclusion

We revisit reflection in LRMs through the lens of self-verification. Our analysis shows that reflection is not monolithic: a substantial portion consists of rechecks that are frequent but mostly confirmatory in reasoning trajectories. Motivated by this finding, we introduce EDS as a conservative test-time controller that uses historical verification outcomes to decide when a local recheck can be skipped while preserving later reasoning. Experiments show that behavior-specific control can moderately reduce reasoning length with limited, model-dependent accuracy changes, suggesting that efficient reflective reasoning should distinguish what kind of reflection is being performed rather than suppressing reflection uniformly.

7 Limitations

This work studies selective control of local self-verification in mathematical reasoning, where recheck behavior is frequent and explicitly observable through recomputation, constraint checking, and validation of intermediate results. The rethink–recheck distinction may also arise in code generation, scientific reasoning, and tool-use agents, where local verification could correspond to rerunning a test, rechecking a constraint, or repeatedly validating a tool result. However, the current detector and experience pool should not be assumed to transfer directly: each domain requires a domain-specific definition of verification onset and corrective outcome.

EDS is a conservative test-time controller, not a maximal token-compression method. It targets local recheck rather than stopping reasoning globally or suppressing reflection triggers broadly. We instantiate experience retrieval with BM25 and simple label aggregation for efficiency and interpretability, while leaving more advanced retrieval or intervention policies to future work.

A confirmatory recheck may still have a stabilizing or preventive role even when it does not change the observed trajectory. Consequently, a

recheck labeled confirmatory in historical data is not guaranteed to be safe to suppress in another rollout, particularly under distribution shift. EDS mitigates rather than eliminates this risk through conservative retrieval-based intervention; stronger guarantees remain future work.

8 Acknowledgment

This research / project is supported by the National Research Foundation, Singapore under its National Large Language Models Funding Initiative, (AISG Award No: AISG-NMLP-2024-005), and under its National Cybersecurity R&D Programme (CRPO-GC5-NTU-011). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore.

References

- AIME. 2025. AIME Problems and Solutions, 2025. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed: January 19, 2026.
- AMC12. 2023. AMC 12 Problems and Solutions. https://artofproblemsolving.com/wiki/index.php/AMC_12_Problems_and_Solutions. Accessed: January 19, 2026.
- Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, et al. 2025. *Llama-nemotron: Efficient reasoning models*. *Preprint*, arXiv:2505.00949.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, et al. 2025. *Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning*. *Preprint*, arXiv:2501.12948.
- Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. *Thinkless: Llm learns when to think*. *Advances in neural information processing systems*, 38:151268–151295.
- Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. *OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems*. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, Bangkok, Thailand. Association for Computational Linguistics.

- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. [Measuring mathematical problem solving with the math dataset](#). *arXiv preprint arXiv:2103.03874*.
- Jiameng Huang, Baijiong Lin, Guhao Feng, Jierun Chen, Di He, and Lu Hou. 2026. [Efficient reasoning for large reasoning language models via certainty-guided reflection suppression](#). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 31176–31184.
- Lingjie Jiang, Xun Wu, Shaohan Huang, Qingxiu Dong, Zewen Chi, Li Dong, Xingxing Zhang, Tengchao Lv, Lei Cui, and Furu Wei. 2025. [Think only when you need with large hybrid-reasoning models](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Liwei Kang, Yue Deng, Yao Xiao, Zhanfeng Mo, Wee Sun Lee, and Lidong Bing. 2025. [First try matters: Revisiting the role of reflection in reasoning models](#). *Preprint*, arXiv:2510.08308.
- Kaiyuan Liu, Chen Shen, Zhanwei Zhang, Junjie Liu, Xiaosong Yuan, and Jieping ye. 2025a. [Efficient reasoning through suppression of self-affirmation reflections in large reasoning models](#). *Preprint*, arXiv:2506.12353.
- Xin Liu and Lu Wang. 2025. [Answer convergence as a signal for early stopping in reasoning](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17896–17907, Suzhou, China. Association for Computational Linguistics.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. [Roberta: A robustly optimized BERT pretraining approach](#). *CoRR*, abs/1907.11692.
- Zichen Liu, Changyu Chen, Wenjun Li, Tianyu Pang, Chao Du, and Min Lin. 2025b. There may not be aha moment in rl-zero-like training — a pilot study. <https://oatllm.notion.site/oat-zero>. Notion Blog.
- Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. 2025. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. <https://pretty-radio-b75.notion.site/DeepScaler-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2>. Notion Blog.
- MAA. 2024. American Invitational Mathematics Examination (AIME) 2024. <https://maa.org/math-competitions/american-invitational-mathematics-examination-aime>. Accessed: January 19, 2026.
- Minjia Mao, Bowen Yin, Yu Zhu, and Xiao Fang. 2025. [Early stopping chain-of-thoughts in large language models](#). *Preprint*, arXiv:2509.14004.
- OpenAI. 2024. [Openai o1 system card](#). *Preprint*, arXiv:2412.16720.
- OpenAI. 2025. [gpt-oss-120b & gpt-oss-20b model card](#). *Preprint*, arXiv:2508.10925.
- OpenAI. 2026. [Openai gpt-5 system card](#). *Preprint*, arXiv:2601.03267.
- Stephen Robertson and Hugo Zaragoza. 2009. [The probabilistic relevance framework: Bm25 and beyond](#). *Foundations and trends® in information retrieval*, 4(1-2):1–174.
- Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. 2025. [DAST: Difficulty-adaptive slow-thinking for large reasoning models](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 2322–2331, Suzhou (China). Association for Computational Linguistics.
- Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, and Xia Hu. 2025. [Stop overthinking: A survey on efficient reasoning for large language models](#). *Preprint*, arXiv:2503.16419.
- Renliang Sun, Wei Cheng, Dawei Li, Haifeng Chen, and Wei Wang. 2025. [Stop when enough: Adaptive early-stopping for chain-of-thought reasoning](#). *Preprint*, arXiv:2510.10103.
- Qwen Team. 2025. [Qwq-32b: Embracing the power of reinforcement learning](#).
- Yi Wang, Junxiao Liu, Shimao Zhang, Jiajun Chen, and Shujian Huang. 2025. [Pats: Process-level adaptive thinking mode switching](#). *Preprint*, arXiv:2505.19250.
- Siye Wu, Jian Xie, Yikai Zhang, Aili Chen, Kai Zhang, Yu Su, and Yanghua Xiao. 2025. [ARM: Adaptive reasoning model](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025a. [Qwen3 technical report](#). *arXiv preprint arXiv:2505.09388*.
- Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weipinng Wang. 2026. [Dynamic early exit in reasoning models](#). In *International Conference on Learning Representations*, volume 2026, pages 88170–88210.
- Shu Yang, Junchao Wu, Xin Chen, Yunze Xiao, Xinyi Yang, Derek F. Wong, and Di Wang. 2025b. [Understanding aha moments: from external observations to internal mechanisms](#). *Preprint*, arXiv:2504.02956.

- Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. 2025a. [Reasoning models know when they're right: Probing hidden states for self-verification](#). *Preprint*, arXiv:2504.05419.
- Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. 2025b. [AdaptThink: Reasoning models can learn when to think](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3716–3730, Suzhou, China. Association for Computational Linguistics.
- Shengjia Zhang, Junjie Wu, Jiawei Chen, Changwang Zhang, Zhe Li, Xingyu Lou, Wangchunshu Zhou, Sheng Zhou, Can Wang, and Jun Wang. 2026. [Othink-r1: Intrinsic fast/slow thinking mode switching for over-reasoning mitigation](#). *Preprint*, arXiv:2506.02397.
- Xiaoyun Zhang, Jingqing Ruan, Xing Ma, Yawen Zhu, Haodong Zhao, Hao Li, Jiansong Chen, Ke Zeng, and Xunliang Cai. 2025c. [When to continue thinking: Adaptive thinking mode switching for efficient reasoning](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5808–5828, Suzhou, China. Association for Computational Linguistics.
- Jiachen Zhao, Yiyu Sun, Weiyan Shi, and Dawn Song. 2026. [Can aha moments be fake? towards quantifying decorative and true thinking in chain-of-thought](#). *Preprint*, arXiv:2510.24941.

A Benchmark Statistics and Setups

Table 3: Statistics of Datasets and Rollouts

Dataset	Problems	Rollouts per Problem	Total Rollouts
AIME24	30	16	480
AIME25	30	16	480
AMC	40	4	160
MATH500	500	1	500
Olympiad Bench	674	1	674
Total	1274	-	2294

For EDS, we construct the experience pool using reasoning rollouts collected from the Deepscaler dataset (Luo et al., 2025), which consists of AIME and AMC problems prior to 2023. Using GPT-5² (OpenAI, 2026), we extract recheck activation set $\mathcal{S}$ and annotate whether each recheck is corrective or confirmatory, resulting in an experience pool of 10k units with a confirmatory-to-corrective ratio of approximately 7:3. Recheck activations during inference are detected using a RoBERTa-base (Liu et al., 2019). The classifier is trained on 4,245 instances derived from $\mathcal{S}$, with a positive-to-negative ratio of 1:3 and a balanced split between hard and easy negatives. Easy negatives are uniformly sampled from non-verification sentences in the original rollouts. We train the classifier for 3 epochs with a learning rate of $2e-5$, achieving 95.31% precision, 93.12% recall, and 94.20% F1 on the held-out split. A missed recheck primarily reduces potential savings, whereas a false-positive detection does not directly trigger suppression because it must additionally pass the retrieval-based utility threshold. During EDS inference, we retrieve the top-30 most similar experience units using BM25 (Robertson and Zaragoza, 2009) and suppress a recheck if the proportion of confirmatory retrieved episodes exceeds $\tau = 0.8$. This threshold is set above the base confirmatory rate of the experience pool (approximately 0.7).

B Reflection Type Statistics

To investigate the semantic composition of reflection steps, we analyze four recent models—Qwen3-8B (Yang et al., 2025a), QwQ-32B (Team, 2025), gpt-oss-20b (OpenAI, 2025), and DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI et al., 2025)—across a diverse set of benchmarks, including AIME24 (MAA, 2024), AIME25 (AIME, 2025), AMC (AMC12, 2023), and MATH500 (Hendrycks et al., 2021). This setup allows us to examine how reflection behaviors vary across model scales and problem domains.

From Figure 8, we observe a persistent inefficiency in self-verification behaviors across models and benchmarks. Specifically, confirmatory rechecks substantially outnumber corrective rechecks across models and benchmarks. This pattern suggests that current reasoning models frequently perform local verification even when the observed recheck does not lead to a correction.

C Human Evaluation

To evaluate the reliability of the LLM-based reflection classifier, we conducted a human evaluation on a random subset of 1,000 reflection steps. Each step was independently annotated by a human expert following the same reflection schema. We compared LLM and human annotations in terms of label distributions and computed both overall agreement and Cohen’s κ to quantify alignment beyond chance.

Table 4 shows that the LLM’s predicted label distribution closely matches that of the human annotator across the primary reflection categories, with minor discrepancies attributable to the LLM’s additional “Unable to Classify” label. The confusion matrix in

²We use the model gpt-5-2025-08-07.

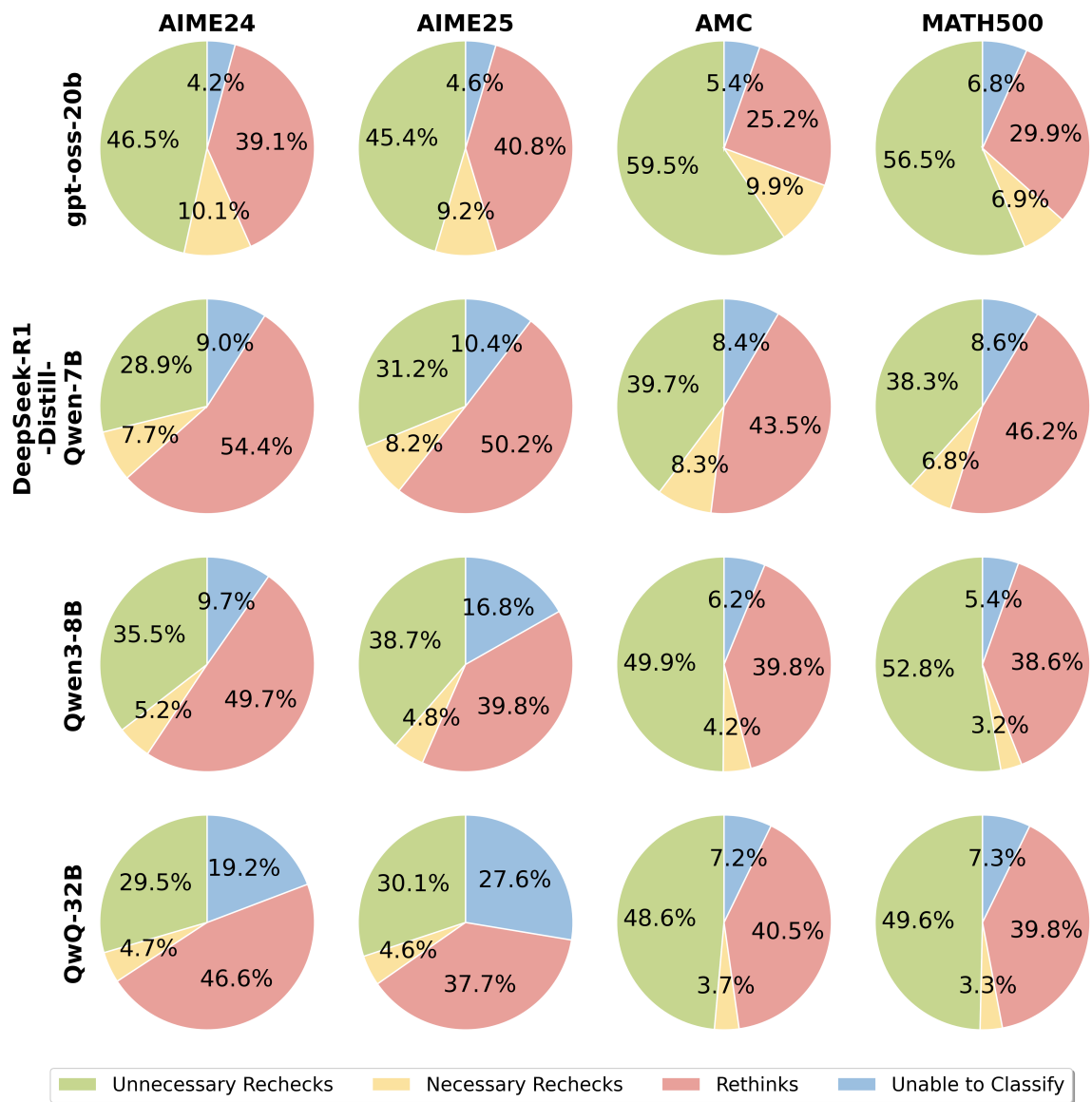


Figure 8: Proportion of reflection types for Qwen3-8B, QwQ-32B, Deepseek-R1-Distilled-Qwen-7B and gpt-oss-20b annotated by GPT-5

Label	Human	LLM
Corrective Rechecks	54 (5.4%)	48 (4.8%)
Confirmatory Rechecks	428 (42.8%)	374 (37.4%)
Rethink	518 (51.8%)	489 (48.9%)
Unable to Classify	-	89 (8.9%)

Table 4: Distribution of reflection types in human and LLM annotations ($n = 1000$).

Human \ LLM	Corrective	Confirmatory	Rethinks	Unable to Classify
Corrective Rechecks	47	1	1	5
Confirmatory Rechecks	1	369	17	41
Rethinks	0	4	471	43
Unable to Classify	0	0	0	0

Table 5: Confusion matrix between human annotations (rows) and LLM predictions (columns) over 1,000 reflection steps.

Table 5 further demonstrates that most disagreements occur between conceptually adjacent categories (e.g., corrective vs. confirmatory rechecks), indicating that misclassifications primarily arise from boundary ambiguity rather than systematic errors.

Quantitatively, the LLM achieves an overall agreement of 88.7% with human annotations and a Cohen’s κ of 0.81, suggesting strong concordance beyond chance. Notably, the model rarely confuses non-adjacent reflection types, reinforcing the semantic coherence of its predictions.

These results show substantial agreement between the LLM annotations and the human annotator, supporting the use of the LLM-based annotation pipeline for the large-scale analyses in this study.

D Additional results

D.1 More Models

Table 6: Accuracy (Pass@1) and average reasoning length for Base, Full-suppress, and Experience-Driven Suppression (EDS) on additional backbones. EDS consistently reduces reasoning length, while its accuracy effect is model- and dataset-dependent. Compared with Full-suppress, selective experience-guided suppression generally yields a more conservative accuracy–efficiency trade-off.

Model	Dataset	Accuracy (%)			Length		
		Base	Full-suppress	EDS	Base	Full-suppress	EDS
DeepSeek-7B	AIME24	57.50	56.67 (-0.83)	58.75 (+1.25)	11237	10105 (-10.1%)	10478 (-6.8%)
	AIME25	39.38	35.42 (-3.96)	36.46 (-2.92)	12489	11221 (-10.1%)	11680 (-7.4%)
	AMC	91.25	90.00 (-1.25)	90.63 (-0.62)	5401	5067 (-6.2%)	5145 (-4.7%)
	MATH500	90.60	87.20 (-3.40)	89.80 (-0.80)	3303	2726 (-17.5%)	2891 (-12.5%)
	Olympiad Bench	69.00	66.91 (-2.09)	67.95 (-1.05)	7913	7002 (-11.5%)	7183 (-9.2%)
	Average	66.66	64.21 (-2.45)	65.78 (-0.88)	8386	7467 (-11.0%)	7736 (-7.75%)
QwQ-32B	AIME24	79.17	78.75 (-0.42)	83.33 (+4.16)	12532	10852 (-13.4%)	11340 (-9.5%)
	AIME25	68.54	64.16 (-4.38)	65.63 (-2.91)	15811	14133 (-10.6%)	14908 (-5.7%)
	AMC	97.50	93.75 (-3.75)	95.00 (-2.50)	7542	6526 (-13.5%)	6719 (-10.9%)
	MATH500	97.00	95.60 (-1.40)	97.00 (-0.00)	4659	3768 (-19.1%)	3940 (-15.4%)
	Olympiad Bench	81.90	81.45 (-0.45)	83.53 (+1.63)	9602	8454 (-12.0%)	8710 (-9.3%)
	Average	82.91	81.21 (-1.70)	83.48 (+0.57)	10293	8988 (-12.7%)	9379 (-8.9%)

We further evaluate EDS on DeepSeek-R1-Distill-Qwen-7B and QwQ-32B, with results shown in Table 6. Across both additional backbones, EDS consistently reduces average reasoning length, although the corresponding accuracy changes are model-dependent. On QwQ-32B, EDS reduces average reasoning length by 8.9% with a +0.57-point average accuracy change. On DeepSeek-7B, EDS reduces length by 7.75% with a -0.88-point accuracy change. Importantly, the degradation on DeepSeek-7B is smaller than

that of Full-suppress (-2.45 points), consistent with our interpretation of EDS as a conservative selective controller rather than a method that guarantees accuracy improvement.

D.2 Retrieval Validation and Ablations

EDS relies on the assumption that recheck episodes with similar local reasoning contexts provide informative evidence about whether a current recheck is likely to be corrective or confirmatory. We first directly evaluate this assumption through an intrinsic retrieval experiment. For each annotated recheck episode, we exclude the episode itself from the experience pool, retrieve $k = 30$ neighboring episodes, and predict its corrective/confirmatory label by majority vote over the retrieved labels. For the random baseline, we uniformly sample the same number of episodes from the remaining pool and apply the same majority-vote procedure. As shown in Table 7, BM25 retrieval achieves 92.50% prediction accuracy, compared with 72.30% under random retrieval. The relatively high random baseline reflects the imbalance of the experience pool toward confirmatory rechecks. Nevertheless, the substantial improvement obtained by BM25 indicates that local-context similarity contains predictive information about observed recheck outcomes beyond the marginal label distribution.

Table 7: Intrinsic validation of experience retrieval. For each held-out recheck episode, its corrective/confirmatory outcome is predicted by majority vote over retrieved neighboring episodes.

Retrieval Strategy	Majority-vote Accuracy (%)
Random retrieval	72.30
BM25 retrieval	92.50

We further examine the sensitivity of EDS to its retrieval configuration on AIME24 with Qwen3-8B. Table 8 reports ablations over the experience-pool size, retrieval method, and the number of retrieved episodes k . Each block varies one component while keeping the remaining configuration fixed. The default EDS configuration uses a 10k experience pool, BM25 retrieval, and $k = 30$.

For the experience-pool size, reducing the default pool from 10k to 2k results in only small changes in both Pass@1 (72.92 vs. 72.49) and reasoning length (13,296 vs. 13,145). This suggests that EDS is not highly sensitive to pool size within the evaluated range. We use the 10k pool as the default configuration in the main experiments. Similarly, varying the number of retrieved episodes from $k = 30$ to $k = 10$ produces only minor changes in accuracy and reasoning length, indicating limited sensitivity to the exact retrieval-set size in this range.

The choice of retrieval method has a larger effect. BM25 achieves higher Pass@1 than DPR (72.92 vs. 69.90), while DPR produces somewhat shorter reasoning traces (12,914 vs. 13,296 tokens). One possible explanation is that local mathematical rechecks often contain exact numbers, symbolic expressions, and arithmetic structures, for which lexical overlap can provide a useful matching signal. We therefore use BM25 as a simple and lightweight instantiation of experience retrieval, while not claiming that it is generally optimal across models or domains.

D.3 Runtime and Computational Overhead

Reducing generated reasoning length does not necessarily translate into lower wall-clock runtime, since EDS introduces additional inference-time components, including recheck detection, experience retrieval, and intervention. We therefore measure wall-clock runtime for the complete AIME24 evaluation with Qwen3-8B under a $4 \times A100$ setup. We evaluate all 30 problems with one rollout per problem and report the total runtime normalized by the number of problems.

As shown in Table 9, the complete evaluation corresponds to 57.1 seconds per question for Base and 68.5 seconds per question for EDS after normalization by the 30 evaluated problems. Thus, although EDS reduces generated reasoning length, the current implementation introduces approximately 11.4 additional seconds per question under this setup. The RoBERTa-based recheck detector is lightweight, and the additional overhead mainly comes from retrieval over the experience pool. Importantly, EDS does not invoke any additional LLM calls during inference.

Table 8: Retrieval ablations on AIME24 with Qwen3-8B. Each block varies one component while keeping the remaining EDS configuration fixed. The default configuration uses a 10k experience pool, BM25 retrieval, and $k = 30$.

Component	Setting	Pass@1 (%)	Length
Pool size	2k	72.49	13,145
	10k	72.92	13,296
Retriever	DPR	69.90	12,914
	BM25	72.92	13,296
Top- k	10	72.77	13,445
	30	72.92	13,296

Table 9: Average wall-clock runtime on AIME24 with Qwen3-8B under a $4 \times A100$ setup. Results are measured over 30 problems with one rollout per problem.

Setting	Time / Question (s)
Base	57.1
EDS	68.5

These results clarify that EDS should not be interpreted as an optimized wall-clock acceleration system in its current implementation. Rather, the current experiments establish that historical verification outcomes can be used to selectively reduce generated reasoning while preserving the opportunity for later reasoning. The runtime overhead is modular and may be reduced through engineering improvements such as retrieval caching, batching, asynchronous retrieval, or more efficient indexing, which we leave to future work.

E Example of Experience Unit and Retrieval

Query Example (recheck activation and upper context)

Then pairwise intersections:

Numbers divisible by 3 and 23: $\text{lcm}(3,23)=69$. $\text{floor}(2003 / 69)$. $69*29 = 2001$, so $\text{floor}(2003 / 69) = 29$. Correct.

Numbers divisible by 3 and 29: $\text{lcm}(3,29)=87$. $\text{floor}(2003 / 87)$. $87*23 = 2001$, so $\text{floor}(2003 / 87) = 23$. Correct.

Numbers divisible by 3 and 2003: $\text{lcm}(3,2003)=3*2003$. Since $3*2003 = 6009 > 2003$, $\text{floor}(2003 / 6009)=0$. Correct.

Numbers divisible by 23 and 29: $\text{lcm}(23,29)=667$. $\text{floor}(2003 / 667)$. $667*3 = 2001$, so $\text{floor}(2003 / 667)=3$. Correct.

Numbers divisible by 23 and 2003: $\text{lcm}(23,2003)=23*2003$, which is way larger than 2003, so 0. Correct.

Numbers divisible by 29 and 2003: same, 0. Correct.

So sum of pairwise intersections: $29 + 23 + 0 + 3 + 0 + 0 = 55$. Correct.

Then subtract 55 from 824: $824 - 55 = 769$. Correct.

Then triple intersections:

Numbers divisible by 3,23,29: $\text{lcm}(3,23,29)=3*23*29=2001$. $\text{floor}(2003 / 2001)=1$. Correct.

Other triplets: 3,23,2003: $\text{lcm}=3*23*2003$, which is way bigger than 2003, so 0. Similarly others.

So total triple intersections: 1. So add 1: $769 + 1 = 770$. Correct.

Four-way intersection: 0. So total numbers not coprime: 770. Therefore, numbers coprime: $2003 -$

770 = 1233.

Wait, but let me check if that answer makes sense.

Top-1 Example

“Context”:{

So, subtract 55 from the previous total: $824 - 55 = 769$.

Now, add back the triple intersections.

Numbers divisible by $\text{lcm}(3,23,29) = 3*23*29 = 2001$. $\text{floor}(2003 / 2001) = 1$.

Numbers divisible by $\text{lcm}(3,23,2003) = 3*23*2003$, which is way larger than 2003, so 0.

Similarly, numbers divisible by $\text{lcm}(3,29,2003) = 3*29*2003$, also larger than 2003, so 0.

Numbers divisible by $\text{lcm}(23,29,2003) = 23*29*2003$, also larger than 2003, so 0.

Therefore, the triple intersections sum up to $1 + 0 + 0 + 0 = 1$. So we add 1: $769 + 1 = 770$.

Now, subtract the four-way intersection: numbers divisible by $\text{lcm}(3,23,29,2003) = 3*23*29*2003$, which is way larger than 2003, so $\text{floor}(2003 / \text{that number}) = 0$. Therefore, subtract 0: $770 - 0 = 770$.

Therefore, the total number of numbers not coprime to n is 770. Therefore, the number of numbers coprime to n is total numbers (2003) minus 770, which is $2003 - 770 =$ Wait, but hold on. Let me check my calculations again, because that seems a bit high. Let me verify each step again.

First, total numbers: 2003. Correct.

First part: numbers divisible by 3, 23, 29, or 2003: $667 + 87 + 69 + 1 = 824$. Correct.

Then subtract pairwise intersections:

Numbers divisible by 3 & 23: 29. 3 & 29: 23. 3 & 2003: 0. 23 & 29: 3. 23 & 2003: 0. 29 & 2003: 0. Total pairwise: $29 + 23 + 0 + 3 + 0 + 0 = 55$. Correct. So $824 - 55 = 769$.

Then add back triple intersections:

Numbers divisible by 3, 23, 29: 2001. $\text{floor}(2003 / 2001) = 1$. The other triplets: 3,23,2003: $\text{floor}(2003 / (3*23*2003)) = 0$. Similarly others. So total triple intersections: 1. So $769 + 1 = 770$.

}

“Label”:{

“Unnecesasry”

}

F Prompts

Reflection Identifier Prompt

Reflection Step Classification Task

****Task:**** You are given a single step from a model's reasoning process.

****Goal:**** Decide whether this step is a **reflection step**.

Core Rule (Strict)

A step is a reflection ****only if it explicitly refers to the model's own reasoning process**** and is about **monitoring, evaluating, or revising that reasoning.**

It must clearly do ****at least one**** of the following: - Verify, check, audit, or critique earlier reasoning or results - Propose changing, reconsidering, or comparing approaches or strategies

This must be about the ****reasoning itself****, not the problem.

Required Signal

The step should contain ****explicit self-referential or metacognitive language****, such as: - "I should check..." - "This approach might be wrong..." - "Let me reconsider..." - "So far, my reasoning assumes..." - "I may have made a mistake..." - "Another strategy could be..."

If it does ****not**** monitor or revise the reasoning process, it is ****not**** a reflection.

Not a Reflection (Always False)

Do ****not**** classify as reflection if the step is mainly: - Solving the problem - Doing calculations, algebra, or transformations - Applying formulas, rules, or definitions - Deriving new results or conclusions - Explaining why a step works - Stating what to do next without evaluating the reasoning

Even if it uses words like "think", "so", or "therefore", it is ****not**** a reflection unless it critiques or evaluates the reasoning itself.

Output Format (Strict)

Output only:

True or False

No explanations. No extra text.

Reflection Type Classifier Prompt

You are an annotation assistant.

You are given: - an earlier reasoning step - several later lines of context from the same reasoning trace - the reasoning step to be classified

Your task is to classify the step.

You must output exactly one label: a. Successful Correctness Repair Action b. Unsuccessful Correctness Repair Action c. Strategy Exploration d. Unable to classify

Only output the single letter.

Core Concepts

A. Correctness Repair Action (CRA)

A step is a Correctness Repair Action IF AND ONLY IF:

1) There exists a prior calculation, result, assumption, bound, interpretation, or conclusion earlier in the trace.

AND

2) The PURPOSE of the step is to evaluate whether that earlier calculation is correct or incorrect.

AND

3) The step performs or initiates a verification action, such as: - recomputation - checking constraints - plugging values back in - testing cases - searching for counterexamples - validating optimality/minimality/maximality

The action must be aimed at CONFIRMING or REFUTING a prior calculation/result.

B. Strategy Exploration

A step is Strategy Exploration IF AND ONLY IF:

1) Its purpose is to propose, compare, or shift solution approaches.

AND

2) It is NOT primarily verifying or refuting a prior belief.

Examples: - proposing a new method - suggesting an alternative representation - reframing how to approach the problem

C. Successful vs Unsuccessful CRA (STRICT)

ABSOLUTE RULE

A step MUST NOT be labeled ****Successful Correctness Repair Action**** if the resulting calculation/result is the SAME as before.

If the check confirms the earlier calculation/result, it is ALWAYS ****Unsuccessful****.

Successful Correctness Repair Action (a)

Label ****a**** if and only if ALL are true:

1) A prior calculation/result in the trace is objectively WRONG. 2) This step performs a genuine verification or repair attempt. 3) This step REVEALS the calculation/result is wrong. 4) The calculation/result is REPLACED with a correct one. 5) No new error is introduced.

If any condition fails → not successful.

Typical outcomes: - wrong value recomputed correctly - false assumption invalidated - flawed argument repaired

Unsuccessful Correctness Repair Action (b)

Label ****b**** if ANY are true:

- The prior calculation/result was actually correct. - The step checks something but the conclusion does NOT change. - The step fails to detect a real error. - The step checks the wrong thing. - The step introduces a new error.

This explicitly includes: - rechecking and getting the same answer - confirming a correct result - reaffirming a wrong result

Examples (b): - "Let me recompute... I still get 70." - "Plugging it back in, it works." - "Double-checking confirms the previous conclusion."

Lexical Guidance (non-binding)

Verification signals: check, verify, double-check, recompute, re-calculate, plug in, test whether, confirm, see if this holds, counterexample

Strategy signals: maybe, try, consider, alternatively, another approach, instead

Lexical cues do NOT override outcome rules.

Final decision checklist (must follow):

1) Is this step trying to verify or repair a prior belief?

- No → c or d

2) Is it proposing or comparing approaches rather than checking correctness?

- Yes → c

3) Was there an actually wrong prior belief?

- No $\rightarrow$ b
- 4) Did this step expose the error and replace it with a correct belief?
- Yes $\rightarrow$ a
- No $\rightarrow$ b

—
Only output: a, b, c, or d.

Recheck Activation Extraction Prompt

You are an annotation assistant. Your task is to extract ONLY sentences where the model checks, verifies, or attempts to falsify the correctness of something it has ALREADY stated.

We annotate correctness repair actions. We do NOT annotate strategy exploration or forward reasoning.

—
Definition: Correctness Repair Action

A sentence qualifies IF AND ONLY IF:

- 1) There exists a prior belief, result, assumption, bound, interpretation, or conclusion stated earlier in the rollout (explicitly or implicitly).
- 2) The sentence's PURPOSE is to evaluate whether that earlier belief is CORRECT or INCORRECT.
- 3) The sentence performs or initiates a verification action, which may include: - re-computation - checking constraints - testing candidates - searching for counterexamples - validating optimality / minimality / maximality as long as these actions are used to CONFIRM or REFUTE a prior claim.

—
Lexical guidance (strong signals):

Typical verification phrases: check, verify, double-check, re-check, confirm, re-calculate, recompute, plug in, test whether, see if this holds.

If a sentence contains “maybe” or “let me” but does NOT explicitly reference checking correctness (e.g., check / verify / confirm / see if this holds), do NOT annotate it. And do NOT annotate conclusion sentences before starting to check.

—
Examples (annotate these):

1. Re-computation (explicitly redoing a prior step) - "Wait, let me compute this again carefully." - "Let me double-check the arithmetic here."
2. Checking constraints (validating a prior result against conditions) - "Let me check if this satisfies the constraint that X and Y must be inside the square." - "Let me plug this back into the equation to see if it works."
3. Testing candidates (verifying a prior claim via examples or nearby cases) - "Let me test a few cases around this value to confirm it's maximal." - "Let me try a couple of examples to see if this holds."
4. Searching for counterexamples (attempting to falsify a prior claim) - "Let me see if I can find a counterexample." - "Let me check if this ever breaks down."
5. Validating optimality / minimality / maximality - "Let me check if this is indeed the minimum." - "Let me verify that no other placement yields a smaller value."

—
Do NOT annotate:

- Summaries, conclusions: “Therefore, the answer should be...”, “Therefore, this suggests...” - Strategy shifts or planning: “To solve the problem, maybe try...”, “Alternatively, consider...” - Idea generation without correctness intent: “Maybe there is a larger number...” - Forward computation stated for the first time: “Compute...”, “We obtain...”, “Thus...” - Meta-cognition without

verification: "Let me think again."

—
Extraction rules:

- Copy the sentence EXACTLY as it appears in the rollout. - Do NOT annotate fragments or partial clauses. - Do NOT paraphrase, invent, or duplicate sentences.

Recheck Filtering Prompt

You are a strict binary classifier.

Task: Given ONE sentence from an LLM rollout, decide whether this sentence is a "correctness repair action onset": It explicitly initiates or performs a CHECK / VERIFICATION / FALSIFICATION of correctness (including re-computation, checking constraints, testing candidates to confirm/refute, searching for counterexamples, validating optimality).

Label rules:

- Output "1" only if the sentence itself clearly expresses verification intent (e.g., "check", "verify", "double-check", "recompute", "plug in", "test whether this holds", "see if this works", "counterexample", "confirm it's minimal/maximal"). - Output "0" otherwise.

Important:

- Do NOT output 1 for pure planning/strategy ("Maybe try...", "Consider symmetry...", "Let me try coordinates..."). - Do NOT output 1 for generic meta-cognition without checking ("Let me think again."). - Do NOT output 1 for summaries/conclusions ("Therefore...", "Thus...") unless it explicitly says it is checking/validating. - Return ONLY a single character: 1 or 0.

Outcome Annotation Prompt

You are an annotation assistant. You will be given: (1) a recheck onset sentence previously extracted from a rollout, and (2) its surrounding context (a short window of text before and after). Your task: Decide whether the recheck that begins at the onset sentence is NECESSARY or UNNECESSARY, using ONLY evidence in the provided context window.

Definitions (outcome-based)

- NECESSARY (corrective): The recheck discovers an error/inconsistency and leads to a correction. Evidence includes: explicit admission of mistake, retraction, changed equation/value, revised claim, changed plan/approach due to the check, or a different conclusion/answer.
- UNNECESSARY (confirmatory): The recheck does not change anything substantive. It only confirms correctness, repeats the same derivation/result, or proceeds with the same claim after checking.
- INCONCLUSIVE: The excerpt does not contain enough information to tell whether the check caused a correction or merely confirmed correctness (e.g., the check starts but the outcome is not shown).

Rules:

- Use ONLY the provided context window. Do NOT assume anything outside it. - Base the decision on explicit textual evidence (e.g., "I was wrong...", "should be...", changed numbers, "checks out...", "consistent..."). - If there is ANY correction caused by the check (even a small arithmetic fix), label NECESSARY. - If the check finishes and the claim stays the same, label UNNECESSARY. - If unclear, label INCONCLUSIVE.