Thinking Traps in Long Chain-of-Thought: A Measurable Study and Trap-Aware Adaptive Restart

Anonymous ACL submission

Abstract

Scaling test-time compute via Long Chain-of-Thought (Long-CoT) significantly enhances reasoning capabilities, yet extended generation does not guarantee correctness: after an early wrong commitment, models may keep elaborating a self-consistent but incorrect prefix. Through fine-grained trajectory analysis, we identify **Thinking Traps**, prefix-dominant deadlocks where later reflection, alternative attempts, or verification fails to revise the root error. On a curated subset of DAPO-MATH, 89% of failures exhibit such traps. To solve this problem, we introduce **TAAR** (Trap-Aware Adaptive Restart), a test-time control framework that trains a diagnostic policy to predict two signals from partial trajectories: a trap index for *where* to truncate and an escape probability for *whether and how strongly* to intervene. At inference time, TAAR truncates the trajectory before the predicted trap segment and adaptively restarts decoding; for severely trapped cases, it applies stronger perturbations, including higher-temperature resampling and an optional structured reboot suffix. Experiments on challenging mathematical and scientific reasoning benchmarks (AIME24, AIME25, GPQA-Diamond, HMMT25, BRUMO25) show that TAAR improves reasoning performance without fine-tuning base model parameters.

1 Introduction

Recently, large reasoning models (LRMs) such as DeepSeek-R1 (DeepSeek-AI, 2025) and OpenAI o1 (OpenAI, 2024) leverage Long Chain-of-Thought (Long-CoT) to scale test-time computation. By allocating more tokens, long traces can expose latent structure, enable step-by-step verification, and improve performance on difficult problems. However, longer reasoning also demands the ability to revisit and revise early assumptions; otherwise, additional compute may reinforce incorrect reasoning rather than improve correctness. This raises a practical question: when does "thinking longer" genuinely help, and when does it waste computation?

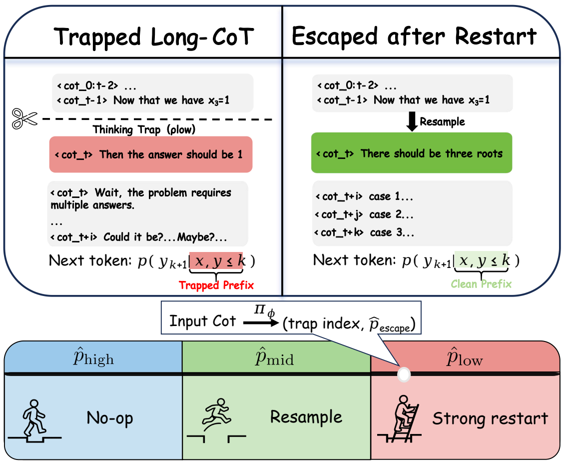


Figure 1: Conceptual illustration of Thinking Traps. A reasoner facing a trap can leverage Diagnostic Policy Model to choose an appropriate escape strategy: step over (no intervention), jump (mild intervention), or use a ladder (strong intervention).

Through fine-grained trajectory analysis of Long-CoT failures, we identify a recurring structural pattern (Ding et al., 2025) that explains much wasted compute. In many failed traces, an early wrong commitment dominates the continuation. Even when the model later attempts reflection or verification, these efforts often fail to revise the root cause and instead elaborate a self-consistent but incorrect prefix. We term this prefix-dominant deadlock a **Thinking Trap** (Figure 1). Our pilot study on DAPO-MATH-17K (BytedTsinghua-SIA, 2025) finds that thinking traps are pervasive: 89% of reasoning errors across four models exhibit such traps (Section 4.2). Crucially, traps are not merely another error category but a bottleneck for test-time scaling, consuming substantial token budgets without yielding correctness.

To study and mitigate this phenomenon, we build a controlled framework for quantifying thinking traps. We curate a hard subset of DAPO-MATH-17K (BytedTsinghua-SIA, 2025) and collect Long-CoT trajectories from four reasoning models (Qwen Team, 2025; DeepSeek-AI, 2025; OpenAI, 2025b,a). Each trajectory is segmented into an indexable sequence. We operationalize the trap index as the earliest segment containing a wrong commitment, identify self-repair windows where the model attempts verification, and define an escape probability via compute-matched resampling and automatic verification. This framework enables measurable study of trap prevalence, position, diagnosability, and escape behavior under fixed test-time budgets.

Building on this framework, we propose Trap-Aware Adaptive Restart (TAAR), a diagnostic-guided intervention strategy. TAAR trains a lightweight policy to predict two signals from partial trajectories: a trap index $\hat{t}$ for truncation and an escape probability $\hat{p}$ for gating intervention strength. At inference, TAAR truncates before the predicted trap segment $s_{\hat{t}}$ and adaptively restarts generation (Figure 1), applying stronger perturbations (higher temperature or structured reboot suffix) for severe traps with low $\hat{p}$.

Experiments on five reasoning benchmarks validate TAAR improves accuracy and token efficiency without fine-tuning base models. Our main contributions can be summarized:

- We build a controlled, measurable study settings for thinking traps in long-CoT failures.
- We propose TAAR, a trap-escape strategy to decide where and how strongly to intervene.
- We have conducted systematic controlled experiments to validate the causal role of removing thinking traps and to demonstrate performance and token-efficiency gains on challenging benchmarks.

2 Related Work

2.1 Analysis of Long-CoT Reasoning Limitations

Long-CoT scales reasoning capabilities (DeepSeek-AI, 2025; OpenAI, 2024) but creates a vulnerability to hallucination snowballing" (Zhang et al., 2023; Xu et al., 2024). In this phenomenon, a single early deviation cascades into a coherent but factually incorrect narrative. Recent studies attribute this to "Prefix Dominance" (Luo et al., 2025) or "Thought Anchors" (Bogdan et al., 2025), where initial mistakes rigidly constrain the model's attention mechanism. Consequently, models tend to rationalize their errors rather than correct them, making standard self-correction less effective (Huang et al., 2023; Stechly et al., 2023). While Process Reward Models (PRMs) provide step-level verification (Lightman et al., 2023; Wang et al., 2024), they often fail to identify errors that are locally logical but globally flawed due to a corrupted premise. We define this recursive deadlock as a Thinking Trap, positing that effective recovery requires pruning the history rather than merely continuing generation.

Inference Model	**# Errors**	**Trap Ratio (%)**
Qwen3-4B-Instruct	169	92.90
DeepSeek-R1-Distill-Qwen-8B	121	91.74
GPT-OSS-20B	108	87.04
GPT-OSS-120B	86	80.23
Total	484	89.05

Source: DAPO-MATH Subset (Section 4.2)

Table 1: Prevalence of Thinking Traps. We report the total number of errors (**# Errors**) and the percentage of errors classified as Thinking Traps (**Trap Ratio**) on DAPO-sample for prevalence analysis.

2.2 Interventions during Inference

Inference strategies have evolved from sampling-based approaches to active structural control. While massive repeated sampling (Wang et al., 2022; Brown et al., 2024) and structured search algorithms like Tree of Thoughts (Yao et al., 2023) demonstrate that scaling test-time compute follows specific scaling laws, they often incur high costs via brute-force coverage or complex state management. Consequently, recent works propose more targeted interventions: parallel reasoning via thought exchange (Luo et al., 2025) or inserting prompts to stimulate deeper deliberation (Zhang et al., 2025a). Other methods manipulate the reasoning medium itself: selecting optimal languages (Zhang et al., 2024) or injecting cross-lingual perturbations at high language uncertainty points (Li et al., 2025). TAAR synthesizes these insights into a framework of diagnostic control. Distinct from untargeted or heuristic perturbations, our core innovation is trap localization: explicitly identifying the *trap segment* to enable precise intervention. By truncating the corrupted effective prefix at its source and selectively triggering recovery, TAAR ensures computation is efficiently allocated to fixing root errors rather than continuing on flawed paths.

3 Thinking Traps and Escape Probability

This section provides formal definitions of the core concepts that underpin TAAR. The operational procedures for obtaining these labels are described in Section 4.2.

3.1 Trap Index as a Wrong Commitment

Given an input x, the model generates a Long-CoT trace Y. We structure this trace as a discrete trajectory $Y = (s_1, \ldots, s_T)$ using a segmentation function that splits the text at natural paragraph boundaries (mainly by `\n\n`; details in Appendix A). This segmentation transforms the continuous stream into an indexable sequence, enabling precise localization-based interventions.

We define **trap index** t^* as the index of the earliest segment containing a *wrong commitment*: an erroneous assumption, unjustified leap, or improper simplification that substantially restricts future reasoning. Note that t^* is distinguishable from minor arithmetic slips; it acts as a structural *branching point* where the trajectory becomes *prefix-dominant*. Once the wrong commitment is anchored, subsequent computation tends to refine the error's consequences rather than revise the root cause. If no such error occurs, we set $t^* = \varnothing$.

3.2 Self-Repair Windows and Escape Probability

Even after a wrong commitment (identified by t^*), the model may actively attempt to correct itself. We identify a set of segments $W \subseteq \{t^* + 1, \ldots, T\}$ as **self-repair windows**. To ensure precision, we include a segment in W only if it explicitly challenges the trap assumption or its consequences (e.g., through verification or by proposing an alternative approach). Routine downstream calculations that do not address the root error are excluded.

While t^* localizes the error, it does not determine the severity of the deadlock. To quantify the likelihood of recovery, we define the **escape probability** $p_{\text{escape}} \in [0, 1]$. Intuitively, this metric answers: *if we truncate the trajectory at a self-repair attempt and resample the continuation, how often does the model succeed?* Formally, given a verifier $\textsc{Correct}(\cdot)$ and a budget of N resampled trials from valid cut points (prioritizing W), we estimate:

$$p_{\text{escape}} = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}\left[\textsc{Correct}\left(\hat{y}^{(n)}\right)\right], \quad (1)$$

where each $\hat{y}^{(n)}$ is a continuation sampled from the truncated prefix. High p_{escape} implies the trap is shallow (escapable via resampling), while low p_{escape} indicates a deep deadlock requiring stronger intervention. When W is empty or provides insufficient distinct cut points, we supplement with random post-trap cut points (Section 4.2).

4 Trap-Aware Adaptive Restart (TAAR)

We now present TAAR, a test-time control framework that operationalizes the trap diagnostics defined in Section 3. TAAR reallocates compute away from trapped continuations toward counterfactual re-derivations (Figure 2). The framework comprises two components: (i) a *diagnostic policy* that predicts (t^*, p_{escape}) from partial reasoning, and (ii) an *adaptive restart controller* that maps these predictions to intervention strategies.

We first describe the control mechanism (§4.1), then detail the dataset construction for training (§4.2), followed by the policy model (§4.3) and the adaptive restart controller (§4.4).

4.1 Control Mechanism

Given an instance x and a (partial or complete) segmented trajectory $Y = (s_1, \ldots, s_T)$, TAAR predicts $(\hat{t}, \hat{p})$, where $\hat{t}$ estimates the trap index and $\hat{p}$ estimates the escape probability. These two signals control restart decisions:

Where to restart. If intervention is triggered, TAAR truncates the trajectory *before* the predicted trap segment, keeping prefix $Y_{<\hat{t}} = (s_1, \ldots, s_{\hat{t}-1})$ and regenerating a continuation from that prefix.

How strongly to restart. Not all traps are equally severe. TAAR uses $\hat{p}$ as a control signal to choose a restart operator: mild restarts encourage light exploration (e.g., default-temperature resampling), while strong restarts apply stronger perturbations (e.g., higher-temperature resampling with an optional structured reboot suffix).

4.2 Dataset Construction

Training the diagnostic policy requires a dataset annotated with trap indices and escape probabilities. We construct this dataset through a pipeline of trajectory generation, offline LLM annotation with human verification, and Monte Carlo estimation.

Source Trajectories and Annotation. We build a challenging subset ("DAPO-hard") from DAPO-MATH-17K, collecting 6,000 Long-CoT trajec-

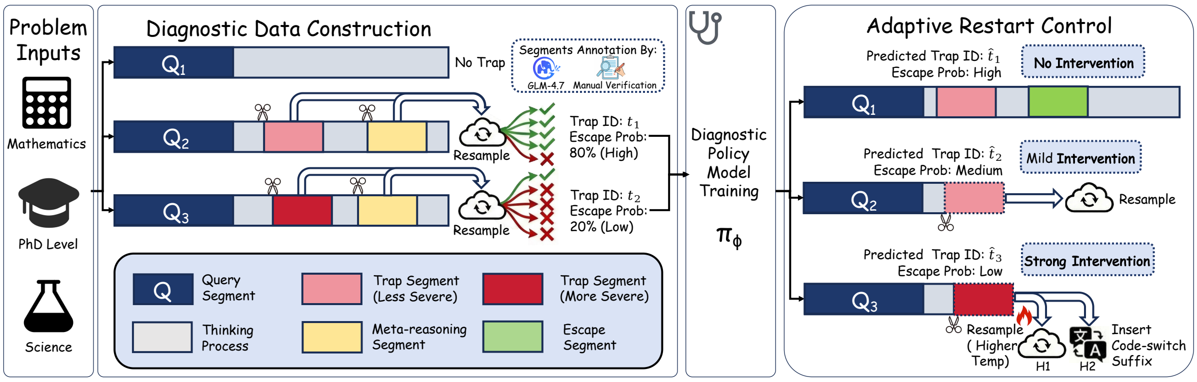


Figure 2: Overview of the TAAR framework. **Left**: Diagnostic Data Construction pipeline that segments trajectories and labels trap indices and escape probabilities via GLM-4.7 annotation with manual verification. **Middle**: Training of the diagnostic policy π_ϕ. **Right**: Adaptive restart controller selects intervention based on $\hat{p}$.

tories from four reasoning models (4B to 120B) on 1,500 problems where not all models succeed. Each trajectory is segmented into an indexable sequence based on paragraph boundaries. We then employ an offline LLM judge (GLM-4.7 (Z.ai, 2025)) to analyze each segmented trace. To ensure annotation quality, two independent human annotators verify 100 random instances sampled from the LLM judgments, achieving 93% agreement. The judge identifies the trap index t^* and extracts valid self-repair windows W where the model attempts verification. Ground-truth answers are provided to the judge solely to enhance offline annotation precision; TAAR does not utilize ground truth during test-time inference.

Escape Probability Estimation. To quantify the severity of the identified traps, we estimate the escape probability p_{escape} via compute-matched resampling. For each trajectory, we generate $N = 36$ continuations from prefixes truncated at the identified self-repair windows W (supplemented by random post-trap points if W is sparse). We verify these continuations using an automatic verifier (math-verify) under a fixed compute budget (temperature 0.7, max 32k tokens). This process yields a robust empirical estimate of the trajectory's ability to self-repair via plain continuation. To ensure training quality, we further apply filtering criteria to select high-quality training samples; details are provided in Appendix F.

4.3 Diagnostic Policy Model

We train a policy model π_ϕ to output (i) a distribution over segment indices for trap localization, and (ii) an escape score for $\hat{p}$. The policy input concatenates the problem statement and the segmented reasoning prefix with segment labels, enabling pointer-style localization.

Training Setup. We supervise π_ϕ using the offline labels (t^*, p_{escape}) from §4.2. To make localization robust to varying amounts of post-trap "idling", we apply *random truncation augmentation*: for each labeled trajectory, we sample an offset δ and provide the prefix up to $t^* + \delta$ as input. Formally,

$$x_{\text{diag}} = \mathcal{T}_{\text{in}}(x, Y_{1:t^*+\delta}), \quad y_{\text{diag}} = \mathcal{T}_{\text{out}}(t^*, p_{\text{escape}}) \qquad (2)$$

where $\mathcal{T}_{\text{in}}(\cdot)$ and $\mathcal{T}_{\text{out}}(\cdot)$ are formatting templates (Appendix J).

4.4 Adaptive Restart Controller

At test time, TAAR operates under a fixed compute budget (e.g., a limited number of sampled paths). Given $(\hat{t}, \hat{p})$, we choose among three intervention strengths: **No Intervention:** if $\hat{p}$ is high, the trajectory is likely to self-repair; we keep the current continuation. **Mild Intervention:** if $\hat{p}$ is moderate, we restart from $Y_{<\hat{t}}$ and resample with the default decoding configuration. **Strong Intervention:** if $\hat{p}$ is low, we restart from $Y_{<\hat{t}}$ and apply a stronger perturbation, such as higher-temperature resampling and an optional *structured reboot suffix*. The suffix is written in the *same language as the prompt* (English in our main experiments) and explicitly requests (i) re-derivation from scratch and (ii) a checklist of key constraints before finalizing the answer. We provide the exact suffix template in Appendix C.

Concretely, we use $\hat{p} \geq 0.6$ for no intervention, $0.1 < \hat{p} < 0.6$ for mild intervention (resample), and $\hat{p} \leq 0.1$ for strong intervention (temperature=1.0 or reboot suffix).

Inference Model	Method	AIME 24 Acc (%)	AIME 25 Acc (%)	BRUMO 25 Acc (%)	HMMT 25 Acc (%)	GPQA Acc (%)	Avg. (%)
Qwen3-4B-Instruct	AVG@4	59.2	**44.2**	54.2	28.3	58.5	48.9
	PRM@4	60.9	43.5	**57.1**	29.0	59.7	50.0
	Autocap	60.0	36.7	46.7	**33.3**	58.1	47.0
	TAAR (Ours)	**64.2**	**44.2**	55.0	27.5	**62.0**	**50.6**
DeepSeek-R1-Distill-Qwen-8B	AVG@4	75.0	67.5	68.3	50.0	61.7	64.5
	PRM@4	75.5	**71.0**	68.1	51.2	64.1	66.0
	Autocap	70.0	63.3	66.7	46.7	59.6	61.3
	TAAR (Ours)	**80.0**	69.2	**74.2**	**57.5**	**64.5**	**69.1**
GPT-OSS-20B	AVG@4	75.8	75.0	75.8	49.2	59.3	67.0
	PRM@4	76.0	77.4	76.6	44.3	58.8	66.6
	Autocap	**80.0**	76.7	70.0	**53.3**	**61.1**	68.2
	TAAR (Ours)	78.3	**77.5**	**80.8**	46.7	59.0	**68.5**
GPT-OSS-120B	AVG@4	**89.2**	84.2	**86.7**	68.3	74.7	**80.6**
	PRM@4	87.3	86.0	85.5	69.0	73.8	80.3
	Autocap	83.3	86.7	76.7	**73.3**	**75.8**	79.2
	TAAR (Ours)	84.2	**87.5**	81.7	69.2	73.4	79.2

Table 2: Main results on five challenging benchmarks. All methods operate under a fixed computational budget of $K = 4$ paths. **TAAR** results are highlighted in gray. **Bold**: best; Underline: second best.

5 Experiments

In this section, we evaluate the effectiveness of the **TAAR** framework. We first describe the experimental setup in §5.1. Then, we present the main results in §5.2.

5.1 Experimental Setup

Base Reasoning Models. We evaluate on the same four reasoning models used for trajectory generation (Section 4.2): `Qwen3-4B-Instruct` (Qwen Team, 2025), `DeepSeek-R1-Distill-Qwen-8B` (DeepSeek-AI, 2025), `GPT-OSS-20B` (OpenAI, 2025b), and `GPT-OSS-120B` (OpenAI, 2025a), spanning 4B to 120B parameters.

Policy Model. We use `Qwen3-4B-Instruct` as the backbone for the diagnostic policy π_ϕ. The model is fine-tuned via supervised fine-tuning (SFT) on the dataset constructed in Section 4.2, comprising 3,661 trajectories with trap indices and escape probability labels. To enable dynamic CoT window handling, we augment the dataset via upsampling, resulting in 16,748 training instances. We use LlamaFactory for training on 8×H20 GPUs with learning rate 1e-5, full fine-tuning, epoch=1, and maximum sequence length of 36k tokens.

Evaluation Benchmarks. We evaluate on five challenging reasoning benchmarks: AIME24 (Zhang and Math-AI, 2024), AIME25 (Zhang and Math-AI, 2025), GPQA-Diamond (Rein et al., 2023), HMMT25 (MathArena, 2025b), and BRUMO25 (MathArena, 2025a).

Baselines. We compare TAAR against the following baselines: **Base Long-CoT (Avg@4):** Standard independent sampling with 4 trajectories per problem, reporting average accuracy. **PRM:** We use `Qwen2.5-Math-PRM-7B`[1] (Zhang et al., 2025b) to score 4 trajectories by averaging step-level rewards, then select the highest-scoring candidate. **AutoCap:** Adaptive routing from (Zhang et al., 2024) that dynamically selects the optimal reasoning language or capability based on input problem distribution.

5.2 Main Results

Table 2 reports accuracy on five challenging reasoning benchmarks under a matched test-time compute budget ($N = 4$ sampled trajectories). TAAR consistently improves over standard multi-sample averaging (Avg@4) for small and mid-scale models: +1.7 points on 4B model and +4.6 points on 8B model on average. Gains are most pronounced on the hardest math benchmarks (+7.5 on HMMT25, +5.9 on BRUMO25 for the 8B model), suggesting that trap-aware restart reallocates compute from trapped continuations toward genuinely different solution paths.

Compared with outcome-based selection (PRM@4), TAAR remains competitive or stronger on the mid-scale setting (+3.1 average points on

---

[1] `https://qwenlm.github.io/blog/qwen2.5-math-prm/`

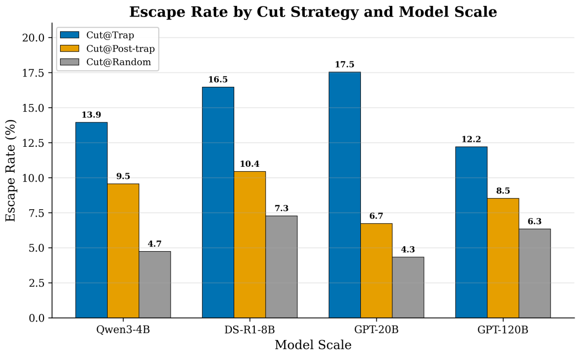


Figure 3: Escape rate by cut strategy. Truncating at the trap segment (Cut@Trap) achieves significantly higher escape rates than keeping the trap and attempting downstream correction (Cut@Post-trap).

8B) without modifying base model parameters. For the 120B model, TAAR is competitive but does not consistently outperform Avg@4 or PRM@4, indicating that when base reasoning is already strong, the benefit of restart is limited by imperfect trap localization and the cost of perturbing near-correct prefixes. We further analyze these effects in Section 6.

6 Analysis

6.1 RQ1: Where and When Should We Restart?

The core design principle of TAAR is that effective recovery requires removing the erroneous commitment from the effective prefix, rather than attempting downstream self-correction. We define p_{escape} as the proportion of restarted trajectories that reach the correct answer under a fixed resampling budget. Under a compute-matched setting, we compare three cut-point strategies: **Cut@Trap** (truncate at the predicted trap index), **Cut@Post-trap** (truncate at post-trap self-repair windows such as reflection/verification segments), and **Cut@Random** (truncate at uniformly sampled positions).

Figure 3 shows that Cut@Trap consistently yields the highest escape rate across all model scales. For instance, on 20B model, restarting at the trap segment achieves an escape rate of 17.5%, while restarting from post-trap windows achieves only 6.7% (and 4.3% for random cuts). Similar gaps hold for 4B model (13.9% vs. 9.5% vs. 4.7%) and 8B model (16.5% vs. 10.4% vs. 7.3%). This indicates that once a wrong commitment is made, subsequent reasoning is often prefix-dominant and self-consistent around the error, making downstream “fixes” largely ineffective.

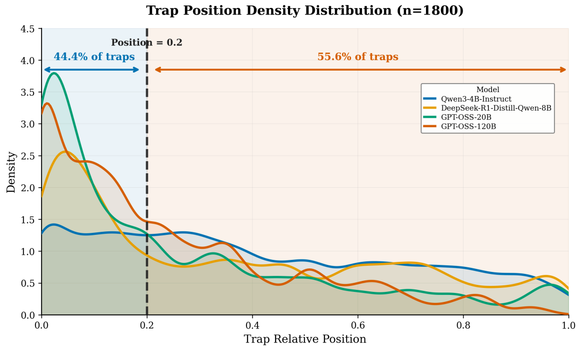


Figure 4: Trap position distribution. Traps concentrate in the early portion of trajectories, with 44.4% occurring before relative position 0.2.

Model	Dataset	Prefix@20%	Prefix@80%	Full
4B	AIME24	60.83	63.33	**64.2**
	AIME25	41.67	**46.67**	44.2
	BRUMO25	54.17	**53.33**	**55.0**
	GPQA	59.85	60.73	**62.0**
20B	AIME24	74.17	75.83	**78.3**
	AIME25	76.67	75.00	**77.5**
	BRUMO25	79.17	81.67	**80.8**
	GPQA	**60.61**	**60.61**	59.0

Table 3: Early diagnosis efficiency. Prefix@X%: the diagnostic policy receives only the first X% of the trajectory. TAAR achieves comparable performance even with partial observations.

We further observe that traps concentrate in the early portion of trajectories (Figure 4), with 44.4% occurring before relative position 0.2. Together, these results validate TAAR’s localization-first intervention: truncate the trap itself, rather than relying on late-stage correction attempts.

6.2 RQ2: How Early Can We Diagnose Traps for Control?

We further test early diagnosis by providing only a prefix of the reasoning trajectory to the policy (Table 3). TAAR retains comparable downstream accuracy even with partial observations. For example, for 4B model on AIME24, performance increases from 60.83 (Prefix@20%) to 64.2 (Full), and for 20B model on BRUMO25, performance is already strong at Prefix@20% (79.17) and remains comparable at Full (80.8).

Figure 5 corroborates this trend: trap detection AUC-ROC exceeds 0.7 with only 20% of the trajectory and saturates around 40–60%, suggesting that diagnosis can be performed online without waiting for full generation. This is a key practical advan-

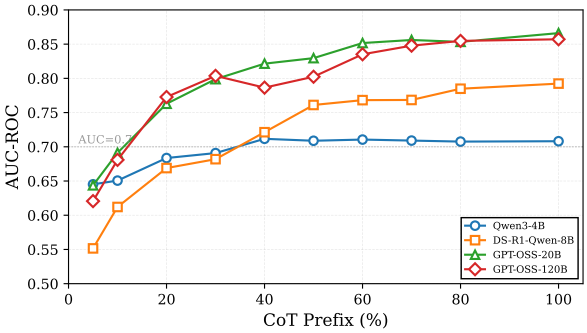


Figure 5: Trap detection AUC-ROC vs. CoT prefix length. Detection exceeds 0.7 AUC with only 20% of the trajectory and saturates around 40–60%.

Strategy	AIME24	AIME25	BRUMO25	GPQA	Avg.
		4B Model			
Cut@25%	59.2	43.3	53.3	60.7	54.1
Cut@50%	60.8	36.7	53.3	60.1	52.7
Cut@75%	58.3	39.2	54.2	61.0	53.2
TAAR (Ours)	**64.2**	**44.2**	**55.0**	**62.0**	**56.4**
		20B Model			
Cut@25%	74.2	75.8	79.2	**61.1**	72.6
Cut@50%	76.7	75.8	77.5	60.4	72.6
Cut@75%	75.8	76.7	80.0	59.7	73.1
TAAR (Ours)	**78.3**	**77.5**	**80.8**	59.0	**73.9**

Table 4: Cut position ablation on AIME24/AIME25/BRUMO25/GPQA. Fixed-position cuts show inconsistent results, while TAAR's adaptive prediction achieves the best average performance.

tage: TAAR does not require completing the entire long chain before deciding whether to intervene.

6.3 RQ3: Does Adaptive Cut Position Outperform Fixed Heuristics?

A natural question is whether content-aware cut prediction provides value over simple fixed heuristics. We compare against fixed-position baselines that truncate the trajectory at predetermined relative positions (25%, 50%, 75%), regardless of reasoning content.

Table 4 shows that **no single fixed cut position works consistently across datasets**. For the 4B model, fixed cuts yield average accuracies of 54.1 (25%), 52.7 (50%), and 53.2 (75%), whereas TAAR achieves 56.4. For the 20B model, fixed cuts range from 72.6 to 73.1 on average, while TAAR reaches 73.9. The variability arises because trap location is instance-dependent: some problems go wrong early, others mid-trajectory. Adaptive localization is necessary to robustly remove the erroneous commitment without excessive truncation.

This motivates the next question: given a predicted cut position, can an additional control signal decide *when* to restart and *how strongly* to perturb decoding? We address this in RQ4 using the escape probability $\hat{p}$.

Strategy	AIME24	AIME25	BRUMO25	GPQA	Avg.
		4B Model			
Cut@AllTraps	65.0	42.5	55.0	57.1	54.9
Random $\hat{p}$	64.2	35.8	55.8	60.9	54.2
TAAR (Ours)	**64.2**	**44.2**	**55.0**	**62.0**	**56.4**
		20B Model			
Cut@AllTraps	67.5	76.7	76.7	55.3	69.1
Random $\hat{p}$	69.2	79.2	80.0	59.1	71.9
TAAR (Ours)	**78.3**	**77.5**	**80.8**	**59.0**	**73.9**

Table 5: Escape probability ablation. Cut@AllTraps ignores $\hat{p}$ entirely; Random $\hat{p}$ uses predicted position but randomizes escape probability. TAAR's predicted $\hat{p}$ improves control decisions.

6.4 RQ4: Does Escape Probability Improve Control?

Beyond trap localization, does the predicted escape probability $\hat{p}$ provide additional control benefit? We find that $\hat{p}$ is discriminative of trajectory correctness (see Appendix M for distribution analysis), motivating its use for adaptive intervention.

To isolate the contribution of $\hat{p}$, we compare three variants: (i) **Cut@AllTraps**, which restarts at every detected trap regardless of $\hat{p}$; (ii) **Random $\hat{p}$**, which uses TAAR's predicted cut position but randomizes the escape probability to remove the gating effect; and (iii) **TAAR**, which uses both predicted position and $\hat{p}$ to adaptively select intervention strength.

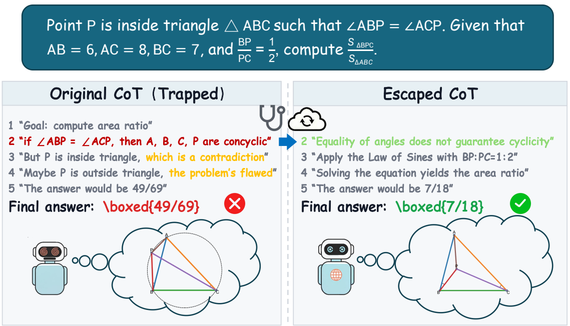
As shown in Table 5, $\hat{p}$ improves control decisions. On the 20B model, TAAR achieves the best average accuracy (73.9), outperforming Random $\hat{p}$ (71.9) and Cut@AllTraps (69.1). This indicates that **not all detected traps warrant the same intervention**: some trajectories can self-repair with continuation or mild resampling, while severely trapped cases benefit from stronger restarts.

Computational efficiency. Escape probability also improves computational efficiency via adaptive gating. The "Baseline" column in Table 6 shows total tokens for 4-sample averaging; "TAAR" and "Ablation" columns show *extra tokens beyond baseline* incurred by each method.

Table 6 shows TAAR incurs only 31.2% additional tokens beyond baseline, compared to 54.3% for the ablation that cuts at all detected traps. This yields a 42.6% reduction in extra overhead. The result demonstrates that $\hat{p}$ is not only a performance

Model	Baseline (Avg@4)	Extra (TAAR)	Extra/Base	Extra (Ablation)	Extra/Base	Savings
4B	1,735,110	576,918	33.2%	1,377,418	79.4%	58.1%
8B	4,388,791	1,279,756	29.2%	2,353,417	53.6%	45.6%
20B	5,617,370	2,190,737	39.0%	2,725,979	48.5%	19.6%
120B	3,500,167	704,376	20.1%	1,818,700	52.0%	61.3%
Total	15,241,438	4,751,787	31.2%	8,275,514	54.3%	42.6%

Table 6: Token efficiency. “Extra” columns show additional tokens beyond baseline incurred by each method. TAAR incurs 31.2% extra tokens relative to baseline, while the ablation (Cut@AllTraps) incurs 54.3%, yielding 42.6% savings.


Point P is inside triangle $\triangle ABC$ such that $\angle ABP = \angle ACP$. Given that $AB = 6, AC = 8, BC = 7$, and $\frac{BP}{PC} = \frac{1}{2}$, compute $\frac{S_{\Delta BPC}}{S_{\Delta ABC}}$.

Original CoT (Trapped)

1 “Goal: compute area ratio”
2 “if ∠ABP = ∠ACP, then A, B, C, P are concyclic”
3 “But P is inside triangle, which is a contradiction”
4 “Maybe P is outside triangle, the problem's flawed”
5 “The answer would be 49/69”
Final answer: \boxed{49/69}

Escaped CoT

2 “Equality of angles does not guarantee cyclicity”
3 “Apply the Law of Sines with BP:PC=1:2”
4 “Solving the equation yields the area ratio”
5 “The answer would be 7/18”
Final answer: \boxed{7/18}

Figure 6: Geometry case study illustrating a prefix-dominant trap and how TAAR restarts by cutting before the wrong commitment.

signal but also a practical knob for controlling test-time cost. This supports our thesis that test-time scaling should be viewed as controlling effective prefix, not merely generating longer continuations.

6.5 RQ5: What Changes After Restart, and Why Does It Work?

We analyze qualitative reasoning dynamics before and after restart to understand why TAAR improves performance. Across cases, a common pattern emerges: once an early wrong commitment enters the context, later deliberation tends to remain prefix-dominant and rationalize the error. TAAR breaks this deadlock by truncating the corrupted prefix and prompting a counterfactual re-derivation.

We analyze qualitative reasoning dynamics before and after restart to understand why TAAR improves performance. A recurring pattern in trapped trajectories is that once an early wrong commitment enters the context, later deliberation becomes prefix-dominant and tends to rationalize the initial error rather than revise it. TAAR breaks this deadlock by truncating the corrupted prefix and prompting a counterfactual re-derivation from the remaining clean context.

Case study. Figure 6 shows a representative geometry problem where the model incorrectly assumes a concyclic polygon configuration at segment 12. The diagnostic policy predicts $\hat{t} = 11$, truncating the trajectory immediately before the erroneous assumption is introduced. After restart, the model re-derives the configuration without imposing concyclicity, identifies the correct irregular structure, and reaches the correct answer.

This case illustrates that restart is effective not because it adds more downstream computation, but because it changes the effective prefix that conditions generation, enabling the model to explore a different reasoning branch. In practice, TAAR shortens the verify–correct loop by preventing prolonged self-consistent but incorrect continuations that are anchored to an early mistaken premise.

7 Discussion and Conclusion

TAAR as test-time diagnostic control. Our results suggest that many Long-CoT failures are not caused by insufficient test-time compute, but by misallocated compute after an early wrong commitment enters the context. TAAR addresses this by predicting two control-relevant signals from partial trajectories: a trap index indicating where to remove the corrupted prefix, and an escape probability indicating whether and how strongly to restart. Under a fixed sampling budget, this simple truncation-and-adaptive-restart mechanism reallocates compute toward counterfactual re-derivations rather than extending trapped continuations.

When does TAAR help, and how does it relate to other methods? TAAR tends to help most on harder benchmarks and for small-to-mid scale reasoners, where prefix-dominant traps are common and restarting can substantially change the explored solution path. For very strong models, gains may be smaller and less consistent under small budgets, since many prefixes are already near-correct and aggressive perturbations can discard useful work. TAAR is complementary to outcome-based selection and structured search: it aims to change the candidate distribution by removing wrong commitments, after which verifiers or ranking can be applied for final selection.

In summary, we formalize Thinking Traps as prefix-dominant deadlocks and show that effective recovery often requires modifying the effective prefix. TAAR provides a lightweight test-time controller that localizes traps and adaptively restarts decoding, improving reasoning performance under a fixed compute budget without updating base model parameters.

8 Limitations

TAAR has several limitations. First, trap localization operates over paragraph-based segments, introducing boundary ambiguity that can lead to over- or under-truncation when wrong commitments span multiple segments. Second, our offline supervision relies on an LLM judge with limited manual auditing, so label noise and judge-specific biases may propagate into the diagnostic policy. Third, escape probability estimation depends on automatic verifiers, which works well for math but may not transfer to open-ended tasks where correctness is subjective. Fourth, for very strong models where prefixes are often near-correct, aggressive restarts can discard useful work and reduce net gains. Finally, TAAR assumes access to explicit Long-CoT traces that can be segmented and truncated, which may not hold for systems with hidden reasoning or constrained APIs.

References

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

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

BytedTsinghua-SIA. 2025. DAPO-Math-17k. Hugging Face Datasets.

DeepSeek-AI. 2025. deepseek-ai/deepseek-r1-0528-qwen3-8b. `https://huggingface.co/deepseek-ai/DeepSeek-R1-0528-Qwen3-8B`.

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

Bowen Ding, Yuhan Chen, Futing Wang, Lingfeng Ming, and Tao Lin. 2025. Do thinking tokens help or trap? towards more efficient large reasoning model. *arXiv preprint arXiv:2506.23840*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Yihao Li, Jiayi Xin, Miranda Muqing Miao, Qi Long, and Lyle Ungar. 2025. The impact of language mixing on bilingual llm reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 32519–32536.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Tongxu Luo, Wenyu Du, Jiaxi Bi, Stephen Chung, Zhengyang Tang, Hao Yang, Min Zhang, and Benyou Wang. 2025. Learning from peers in reasoning models. *arXiv preprint arXiv:2505.07787*.

MathArena. 2025a. Brumo 2025 (matharena dataset).

MathArena. 2025b. Hmmt february 2025 (matharena dataset).

OpenAI. 2024. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*.

OpenAI. 2025a. openai/gpt-oss-120b. `https://huggingface.co/openai/gpt-oss-120b`.

OpenAI. 2025b. openai/gpt-oss-20b. `https://huggingface.co/openai/gpt-oss-20b`.

Qwen Team. 2025. Qwen/qwen3-4b-instruct-2507. `https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507`.

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

Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. 2023. Gpt-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems. *arXiv preprint arXiv:2310.12397*.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439.

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

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. 2024. Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint arXiv:2401.11817*.

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

Z.ai. 2025. GLM-4.7. Hugging Face model.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A Smith. 2023. How language model hallucinations can snowball. *arXiv preprint arXiv:2305.13534*.

Xichen Zhang, Sitong Wu, Haoru Tan, Shaozuo Yu, Yinghao Zhu, Ziyi He, and Jiaya Jia. 2025a. Smartswitch: Advancing llm reasoning by overcoming underthinking via promoting deeper thought exploration. *arXiv preprint arXiv:2510.19767*.

Yifan Zhang and Team Math-AI. 2024. American invitational mathematics examination (aime) 2024. Hugging Face dataset.

Yifan Zhang and Team Math-AI. 2025. American invitational mathematics examination (aime) 2025. Hugging Face dataset.

Yongheng Zhang, Qiguang Chen, Min Li, Wanxiang Che, and Libo Qin. 2024. Autocap: Towards automatic cross-lingual alignment planning for zero-shot chain-of-thought. *arXiv preprint arXiv:2406.13940*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025b. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*.

A Segmentation Function

We extract the reasoning portion of each output (e.g., the content inside `<think>...</think>` when available) and segment it into T continuous segments using paragraph boundaries (`\n\n`). To stabilize index-based localization, we enforce a minimum segment length 200 characters by merging short chunks with adjacent segments.

B LLM Judge Prompt for Trap and Window Annotation

The following is the prompt template used with GLM-4.7 to annotate trap indices, escape points, and self-repair windows.

```
You are a "long-CoT reasoning trap locator."

[Problem]
{problem}

[Input]
A long CoT text segmented with labels:
{segmented_cot}

[Ground Truth Answer]
{ground_truth}

[Task]
1. Identify exactly one trap in the CoT text:
   a. A "trap" is the earliest critical erroneous assumption,
      unjustified leap, or improper simplification that "locks"
      or severely restricts subsequent reasoning.
   b. Consequence: subsequent reasoning space becomes
     significantly constrained, leading to failure or deviation.
   c. If multiple candidates exist, select the earliest and
      most restrictive one.

2. In the entire text, find segments directly related to the
   identified trap (only output labels without repeating their
   contents):

  High-precision eligibility (MUST satisfy; otherwise exclude):
   A segment is eligible ONLY IF it explicitly contains
   meta-reasoning cues targeting the trap, i.e. it explicitly
   does at least one of:
   - Reflection points: explicitly doubt/question the trap
     assumption itself OR a direct consequence of it, but fail
     to correct it.
   - New approach points: explicitly propose a different
     method/representation/strategy to escape, but still rely
     on the trap assumption (do not fix it).
   - Verification points: explicitly check the trap assumption
    OR a direct consequence via examples/boundaries/calculations,
     but miss the key flaw.

   NOT eligible: segments that merely continue routine
  computation/derivation along the trapped path WITHOUT explicit
   doubt / alternative attempt / verification.

   Relevance ranking (internal; do NOT output scores):
   - For each eligible candidate, assign rel in {3,2,1}:
     rel=3: explicitly target the trap assumption itself
            (name/restatement/check) OR explicitly attempt
            to escape it.
     rel=2: explicitly target a direct consequence that
            critically depends on the trap, with
            doubt/alternative/check.
     rel=1: weak/implicit relation -> EXCLUDE (do not output).
   - Keep ONLY candidates with rel >= 2.
  - Each list must be sorted by (rel descending, index ascending).
     Output labels only.

   Selection constraints (precision-first):
   - Do NOT include the trap segment itself; all points must
     satisfy index > trap index.
   - No duplicates; a label can appear in at most ONE list.
   - If a segment fits multiple categories, assign it to the
     most specific with priority:
   new_approach_points > verification_points > reflection_points.
   - Hard caps (no total cap): reflection_points <= 3,
     new_approach_points <= 3, verification_points <= 3.
   - (These arrays may be empty; it is OK to output [].)

3. Determine if escaped:
  a. If any later segment explicitly corrects the trap assumption
      and breaks free from the erroneous path, set
     trap_type="escaped successfully" and "escape_point" to the
      earliest correcting segment.
   b. Otherwise, set trap_type="did not escape" and
      "escape_point"="".

[Output]
Output only valid JSON (no explanations or extra text):
{
  "trap": "cot_x" or "",
  "trap_type": "escaped successfully" or "did not escape" or "",
  "escape_point": "cot_y" or "",
  "reflection_points": ["cot_i", ...],
  "new_approach_points": ["cot_j", ...],
  "verification_points": ["cot_m", ...]
}

[Empty Output]
If no trap satisfying "maximum causal influence/strongest lock"
is found:
{
  "trap": "",
  "trap_type": "",
  "escape_point": "",
  "reflection_points": [],
  "new_approach_points": [],
  "verification_points": []
}
```

C Reboot Suffix Templates

For hard restarts, we append a structured reboot suffix to the truncated prefix. The main experiments use the English suffix (same language as the prompt). We also provide multilingual variants for optional code-switch experiments.

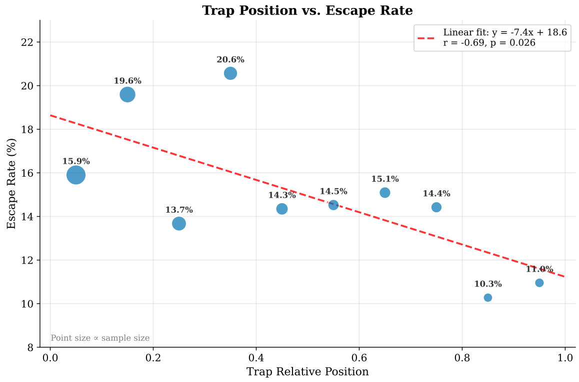


Figure 7: Relationship between trap relative position and escape rate on a subset of the offline data construction dataset (Section 4.2). Earlier traps exhibit higher escape rates, while later traps are harder to recover from. The negative correlation ($r = -0.69$, $p = 0.026$) suggests that early wrong commitments leave more room for self-correction, whereas late traps constrain the remaining trajectory too severely.

English suffix (used in main experiments).

```
Wait, let me completely rethink this problem
in English. The previous train of thought
might be limited, so I need to reorganize
my thoughts in English and analyze from
scratch.
```

Multilingual variants. We also provide translated variants of the English reboot suffix in Chinese, Korean, Russian, Arabic, and French for code-switch experiments (see Appendix L).

D Additional Details for Escape Probability Estimation

We use a total resampling budget of $N = 36$ per trajectory. We sample from post-trap windows (reflection, new-approach, and verification; each capped at 3) and allocate leftover samples to random cut points uniformly drawn from $[t^* + 1, T - 1]$. Decoding uses temperature=0.7, top-p disabled, and total context+generation length $\leq$32k tokens. Correctness is evaluated by the math-verify verifier.

E Trap Position and Escape Rate Analysis

Figure 7 shows the relationship between trap position and empirical escape rate on a subset of our constructed dataset. We observe a significant negative correlation: traps occurring in the early portion of trajectories (relative position < 0.2) have escape rates around 16–20%, while traps in the late portion (relative position > 0.8) drop to 10–11%. Earlier traps leave more remaining trajectory for the model to reflect, reconsider, and potentially self-correct; later traps leave little room for recovery before the final answer.

F Filtering Rules and Loss Breakdown

Preprocess filtering. We remove records with API call errors, invalid JSON, or missing required fields. Across all raw records (6,000):

- API errors: 1,367 (22.4%)
- JSON parse errors: 52 (0.9%)

Consistency filtering. From 4,680 preprocessed records, we remove:

- (i) trap + did-not-escape + correct: 406 (8.7%)
- (ii) trap + escaped + incorrect: 343 (7.3%)
- (iii) no-trap + incorrect: 270 (5.8%)

Total removed by consistency filtering: 1,019 (21.8%).

G Pattern-Based Difficulty and Dataset Composition

We define a 4-bit correctness pattern [120B][20B][8B][4B], where 1 indicates correct final answer and 0 indicates incorrect. We bucket problems by difficulty level based on the number of correct models.

Difficulty level	# trajectories	Trap rate
1 (3/4 correct)	591	53.0%
2 (2/4 correct)	1,394	73.5%
3 (1/4 correct)	1,125	89.0%
4 (0/4 correct)	551	100.0%

Table 7: Difficulty distribution and trap rates in the final dataset (n=3,661).

H Train/Dev/Test Split

We split problems into Train/Dev/Test with a ratio of 80/10/10 using a fixed random seed (42). For each problem, models at different scales (4B, 8B, 20B, and 120B) generate independent reasoning trajectories; all such trajectories are assigned to the same split as the underlying problem, ensuring no cross-split leakage across model sizes. Table 9 reports the per-pattern split counts.

Pattern	Total	With trap	No trap	Trap rate
1100	699	483	216	69.1%
1000	645	583	62	90.4%
0000	551	551	0	100.0%
1110	421	231	190	54.9%
1010	367	274	93	74.7%
1001	191	161	30	84.3%
0010	173	134	39	77.5%
1101	170	82	88	48.2%
0001	169	161	8	95.3%
0100	138	123	15	89.1%
0110	88	72	16	81.8%
0101	49	35	14	71.4%

Table 8: Pattern distribution in the final dataset (n=3,661).

Pattern	Train	Dev	Test	Total
1110	116	14	10	140
1101	48	6	6	60
1100	189	14	32	235
1010	105	17	11	133
1001	70	5	2	77
0110	33	0	2	35
0101	17	3	0	20
1000	185	26	21	232
0100	47	2	6	55
0010	68	12	10	90
0001	60	9	4	73
0000	262	42	46	350
Total	1200	150	150	1500

Table 9: Problem split by pattern (1,500 problems).

I Manual Audit Protocol

We randomly sample 100 instances for manual recheck of trap index and window eligibility to sanity-check LLM annotations.

Audit procedure. Two annotators independently reviewed each sampled instance, checking:

- Whether the identified trap segment is indeed the earliest wrong commitment
- Whether the self-repair windows are correctly classified (reflection, new-approach, or verification)
- Whether the escape point (if any) is correctly identified

J Diagnostic Policy Input/Output Templates

This section describes the formatting templates $\mathcal{T}_{\text{in}}(\cdot)$ and $\mathcal{T}_{\text{out}}(\cdot)$ used to construct training data for the diagnostic policy π_ϕ.

J.1 Input Template $\mathcal{T}_{\text{in}}$

The input to the diagnostic policy concatenates the model identifier, problem statement, and the segmented reasoning trace with explicit segment labels:

```
Please identify and locate the trap in the current problem's
reasoning process, and provide the escape action.

[Model]
{model_name}

[Problem]
{problem_statement}

[Reasoning Process]
<cot_0>
{segment_0_text}

<cot_1>
{segment_1_text}

...

<cot_K>
{segment_K_text}

Output your analysis in JSON format:
```

J.2 Output Template $\mathcal{T}_{\text{out}}$

The output format encodes the trap index and escape probability in JSON:

```
{
  "trap_index": "t*",
  "escape_probability": "p_escape",
  "extra": {extra information}
}
```

During training, we provide gold labels (t^*, p_{escape}) from the offline annotation pipeline. During inference, the policy outputs predictions $(\hat{t}, \hat{p})$ which are used by the adaptive restart controller.

K Diagnostic Policy Evaluation Details

We evaluate the diagnostic policy on test samples spanning four model scales. This section provides detailed breakdowns beyond the summary in Section 6.2.

Trap detection by model scale. Table 10 shows that detection rates vary across model scales. The policy achieves near-perfect detection on 4B trajectories (98.9%) but lower rates on larger models (55–67%). This is expected: larger models produce more subtle errors that are harder for an external diagnostic model to identify.

Localization accuracy. Among detected traps, Table 11 reports position prediction accuracy. While Top-1 exact match is modest (29.1%), the mean absolute error of 9.46 segments represents

Model	Total	Detected	Rate
4B	180	178	98.9%
8B	45	30	66.7%
20B	43	24	55.8%
120B	90	50	55.6%
Overall	358	282	78.8%

Table 10: Trap detection rate by model scale. Detection is easier for smaller models whose errors tend to be more explicit.

only 17.0% of the average trajectory length (55.6 segments), and Within ±3 reaches 55.3%.

Metric	Value
Top-1 Accuracy	29.1%
Within ±3	55.3%
Mean $\lvert \hat{t} - t^* \rvert$	9.46
Avg. CoT segments	55.6
Relative Error	17.0%

Table 11: Overall localization accuracy on detected traps.

Localization by distance to truncation point. Table 12 shows that localization accuracy degrades as the trap occurs further from the truncation point. When the trap is within 1 step of truncation, Top-1 reaches 62.3%; when it is more than 20 steps away, Top-1 drops to 9.7%. This motivates early diagnosis: the sooner we detect a trap after it occurs, the more accurately we can localize it.

Distance	N	Top-1	Within ±3	$\lvert \hat{t} - t^* \rvert$
1 step	53	62.3%	77.4%	4.30
2–3 steps	33	27.3%	84.8%	3.27
4–5 steps	22	40.9%	72.7%	10.86
6–10 steps	53	18.9%	56.6%	5.53
11–20 steps	49	28.6%	44.9%	10.90
>20 steps	72	9.7%	26.4%	17.57

Table 12: Localization accuracy by distance from trap to truncation point.

Localization by input length. Table 13 shows that shorter input sequences yield better localization. For sequences with ≤10 segments, Top-1 reaches 66.1% and Within ±3 reaches 93.2%. Performance degrades for longer sequences due to increased search space and noise.

Escape probability prediction. Table 14 reports the accuracy of escape probability predictions. The positive correlation ($r = 0.336$) indicate that predicted $\hat{p}$ provides a useful signal for adaptive gating.

Input Length	N	Top-1	Within ±3	$\lvert \hat{t} - t^* \rvert$
≤10	59	66.1%	93.2%	0.69
11–20	48	41.7%	68.8%	3.31
21–40	75	22.7%	52.0%	7.12
41–60	45	6.7%	24.4%	13.16
>60	55	5.5%	32.7%	24.38

Table 13: Localization accuracy by input sequence length (number of segments).

Metric	Value
Correlation	0.336

Table 14: Escape probability prediction accuracy.

L Code-Switch Experiments

We conduct exploratory experiments using multilingual reboot suffixes (code-switching) as an alternative hard restart operator. The hypothesis is that switching the reasoning language may provide a stronger perturbation to break free from prefix-dominant traps.

Setup. For trajectories flagged for hard restart, we compare three conditions: (i) **Mono**: same-language reboot suffix (English, as in main experiments), (ii) **Switch**: multilingual reboot suffix (Chinese for English prompts), (iii) **High-T**: higher-temperature resampling without suffix.

Preliminary findings. Code-switching provides marginal improvements over same-language restarts on some model–task combinations, but the effect is inconsistent across scales. We observe that the primary driver of recovery is the truncation point selection (cutting before the trap), with

Mdl	**Data**	**Mono**	**Sw-ar**	**Sw-fr**	**Sw-zh**	**Hi-T**
4B	AIME24	64.2	60.0	59.2	60.0	65.7
	AIME25	44.2	43.3	45.0	43.3	44.2
	BRUMO25	55.0	53.3	54.2	54.2	53.3
20B	AIME24	78.3	77.5	76.7	77.5	75.0
	AIME25	77.5	76.7	79.2	78.3	80.0
	BRUMO25	80.0	81.7	83.3	80.8	85.5

Table 15: Comparison of hard restart strategies across multilingual perturbations. Mono: monolingual reboot suffix; Sw-ar/fr/zh: code-switching to Arabic, French, or Chinese; Hi-T: high-temperature resampling. No single strategy consistently dominates

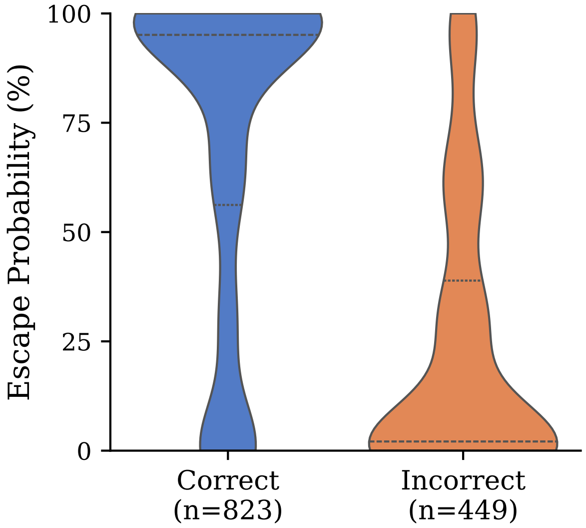


Figure 8: Correlation between Escape Probability and Correctness in Trap Cases

the choice of restart operator contributing a secondary effect. Given this, our main experiments use same-language suffixes for simplicity and reproducibility.

M Correlation between Escape Probability and Correctness in Trap Cases

The predicted escape probability $\hat{p}$ is discriminative (Figure 8): correct trajectories concentrate at high $\hat{p}$, while incorrect ones spread toward lower values.