Contrastive Trajectory Optimization: Decoding-Time Experience Reuse for LLM Reasoning

Anonymous ACL submission

Abstract

Test-time search improves LLM reasoning by exploring multiple rollouts, but most trajectory-level findings and failure patterns are discarded, while prompt-level experience reuse only provides implicit guidance through textual hints. We recast experience reuse as decoding-time distributional guidance and introduce Contrastive Trajectory Optimization (CTO), which separates trajectory-derived experience into positive evidence and negative failure modes and uses them as paired decoding branches. CTO reinforces continuations supported by positive experience while suppressing continuations associated with problem-local failure evidence. Using CTO, we analyze experience construction through extraction, source selection, and ranking. Experiments on mathematical reasoning benchmarks show that CTO improves reasoning-oriented models in most settings. Diagnostic analyses further show that gains come from paired branch control and suppression-safe local negative signals, rather than retrieval/ranking complexity or generic anti-expert decoding.

1 Introduction

Large language models increasingly rely on multi-rollout inference procedures for complex reasoning tasks. Rather than producing a single solution, these procedures allocate additional computation at inference time by sampling, refining, searching over, or verifying multiple candidate trajectories (Wang et al., 2022; Yao et al., 2023; Lightman et al., 2023). Reasoning-oriented models such as OpenAI o1 (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025) further make inference-time computation a practical design axis for improving reasoning behavior. This shift raises a practical question: when search produces many trajectories, how should the information discovered during search be organized and reused?

We focus on external test-time search, where the model explores multiple rollouts outside a single generation trajectory. Although such search improves task-solving probability, it often discards intermediate findings, useful strategies, and recurring failure patterns. Recent experience-reuse methods distill these signals into natural-language snippets for later prompts, making search more cumulative. Yet prompt-level reuse leaves utilization implicit: injected experience must compete with the problem statement, current reasoning state, trajectory summaries, and retrieved items under a limited context budget. As the bank grows, mismatched or noisy items can dilute guidance, making reliable experience selection and utilization more important than storage alone.

We address this bottleneck by recasting experience reuse as decoding-time distributional guidance. Instead of treating positive and negative experience as textual hints, Contrastive Trajectory Optimization (CTO) converts them into paired decoding branches: a positive branch conditioned on reusable findings and constraints, and a negative branch that serves as an anti-expert reference for failure-associated continuations. At each decoding step, CTO contrasts the two branches within a plausible-token set, reinforcing continuations supported by positive experience while suppressing continuations associated with known failure modes.

This formulation also makes experience construction a central design axis. Using CTO as a methodological anchor, we study how trajectory experience is extracted, selected, and rendered before decoding-time reuse. Specifically, *extraction* determines how sampled rollouts are converted into positive and negative records in the persistent experience bank; *source selection* determines which bank records form the temporary candidate pool for the current guided round; and *ranking* determines which candidate records are prioritized and rendered into the positive and negative decoding

branches. Across these stages, CTO benefits most from dense low-noise extraction, problem-local candidate pools, and aligned ranking, while noisy or weakly matched experience can weaken or even reverse its gains.

Our contributions are threefold: (i) We propose Contrastive Trajectory Optimization (CTO), which moves experience reuse from prompt-level hinting to token-distribution control by turning trajectory-derived findings and failure modes into paired decoding branches. (ii) We formulate CTO experience construction as extraction, source selection, and ranking, enabling controlled analysis of experience quality, locality, and alignment. (iii) Under matched multi-round search on mathematical reasoning benchmarks, CTO improves reasoning-oriented LLMs in most settings, with diagnostics attributing gains to paired branch control and problem-local negative evidence.

2 Related Work

Inference-Time Search and Multi-Rollout Reasoning. LLM reasoning can be improved by allocating additional computation at inference time, through longer deliberation, repeated sampling, trajectory search, verification, or trajectory selection (Wei et al., 2022; Wang et al., 2022; Yao et al., 2023; Lightman et al., 2023; Snell et al., 2024). These methods expand the space of candidate reasoning trajectories, but many of them rely on reliable selection signals, such as verifiers or reward models, or use unsuccessful rollouts only as failed attempts rather than reusable evidence (Lightman et al., 2023; Brown et al., 2024; Wang et al., 2026).

Memory-Augmented Search and Experience Reuse. Recent work makes test-time search cumulative by reusing information from past trajectories, such as intermediate conclusions, strategies, and failure patterns (Shinn et al., 2023; Zhao et al., 2024; Zhou et al., 2023; Wang et al., 2026). Most existing methods, however, reuse such experience at the prompt level, where natural-language snippets compete with the problem statement and other retrieved items within a limited context budget (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2026; Jiang et al., 2023, 2024).

Contrastive Decoding and Inference-Time Intervention. Decoding-time intervention methods guide generation by modifying next-token scores rather than only changing the prompt. DExperts combines expert and anti-expert language models to steer controlled generation (Liu et al., 2021), while Contrastive Decoding contrasts expert and amateur model distributions for open-ended generation (Li et al., 2023). Related methods also contrast internal or conditional signals: DoLa contrasts layer-wise logits to improve factuality (Chuang et al., 2023), and classifier-free guidance combines conditional and unconditional scores to improve topic control (Sanchez et al., 2023).

3 Method

Multi-rollout search history and experience extraction. We instantiate Contrastive Trajectory Optimization (CTO) as an experience-conditioned contrastive decoder within a fixed multi-round inference-time search protocol. As illustrated in Figure 1, CTO uses prior-round rollouts as search history and converts accumulated experience into candidate records, paired branches, and contrastive token scores. For any round t, we denote the generated trajectory pool as

$$Y^{(t)} = \{y_1^{(t)}, \ldots, y_R^{(t)}\}, \qquad y_i^{(t)} = (s_i^{(t)}, a_i^{(t)}),$$

where $s_i^{(t)}$ is the generated solution trace and $a_i^{(t)}$ is the extracted final answer. After a round finishes, an experience extractor D maps each trajectory into positive and negative experience items:

$$D(q, y_i^{(t)}) \rightarrow \left(E_{i,+}^{(t)}, E_{i,-}^{(t)}\right). \tag{1}$$

Positive items record reusable findings, constraints, or intermediate conclusions, while negative items record pitfalls such as invalid transformations, misleading assumptions, or ineffective solving strategies. Before guided round $r \geq 1$, only prior-round items are available:

$$\mathcal{E}^{(<r)} = \{(E_{i,+}^{(t)}, E_{i,-}^{(t)}) : 0 \leq t < r\}.$$

The pre-round bank for round r is

$$B^{(r)} = \mathrm{Merge}\left(\mathcal{E}^{(<r)}\right). \tag{2}$$

It stores experience records merged from prior-round annotations, where each record may contain positive and/or negative fields. We write $B_+^{(r)}$ and $B_-^{(r)}$ for the polarity-specific views of this record bank used during branch rendering. Conditioned on $B^{(r)}$, CTO generates $Y^{(r)}$; after round r, new records are merged into $B^{(r+1)}$ for later rounds.

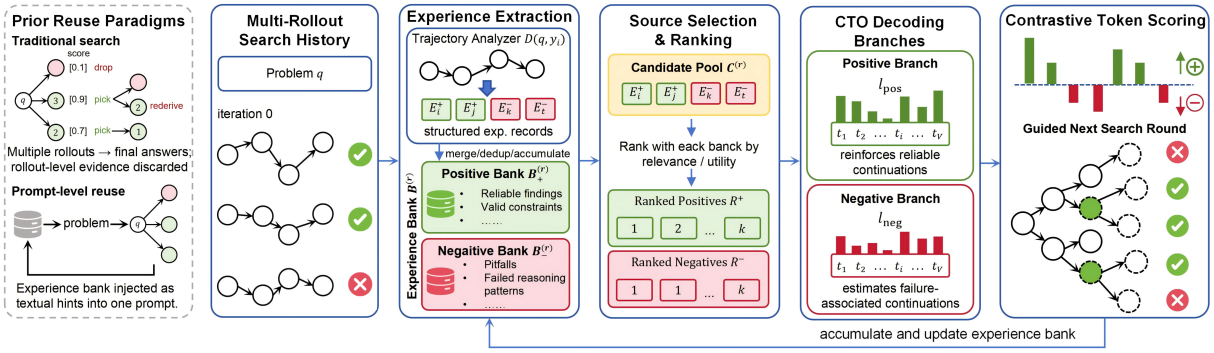

Figure 1: Overview of CTO in a fixed multi-rollout search loop. The left inset positions CTO against prior search/reuse paradigms, which either discard rollout-level evidence after final-answer selection or reuse it only as single-prompt hints. The main pipeline proceeds through multi-rollout search history, experience extraction, source selection and ranking, CTO decoding branches, and contrastive token scoring, then accumulates guided-round experience back into the bank.

Source selection and ranking. Prompt-level experience reuse injects $B_+^{(r)}$ and $B_-^{(r)}$ into a single textual prompt, leaving the model to decide whether and how to follow the provided hints. CTO instead first selects and ranks bank records before branch rendering. Given the problem q and the pre-round bank $B^{(r)}$, CTO first exposes a subset of bank records as the candidate pool:

$$\mathcal{C}^{(r)} = \mathrm{Select}(B^{(r)}, q). \tag{3}$$

It then ranks, aggregates, and truncates candidate records into polarity-specific branch contents:

$$(\mathcal{R}_+^{(r)}, \mathcal{R}_-^{(r)}) = \mathrm{Rank}(\mathcal{C}^{(r)}, q). \tag{4}$$

CTO decoding branches and contrastive token scoring. CTO constructs the positive and negative branch prefixes from these rendered contents:

$$P_b = T_b\left(q, \mathcal{R}_b^{(r)}\right), \quad b \in \{\mathrm{pos}, \mathrm{neg}\}. \tag{5}$$

The positive branch conditions on reliable findings and constraints, while the negative branch serves as an anti-expert reference conditioned on known pitfalls and failed reasoning patterns. It is not instructed to avoid these pitfalls; instead, it estimates which continuations are likely under failure-associated experience, so that such continuations can later be suppressed by contrastive scoring. At decoding step j, both branches share the same generated prefix $x_{<j}$ and produce next-token logits:

$$z_b^{(j)} = g_\theta(P_b, x_{<j}), \quad b \in \{\mathrm{pos}, \mathrm{neg}\}. \tag{6}$$

We convert them to log-probabilities:

$$\ell_b^{(j)} = \log \mathrm{softmax}(z_b^{(j)}), \quad b \in \{\mathrm{pos}, \mathrm{neg}\}. \tag{7}$$

To avoid arbitrary interventions on tokens that the positive branch does not consider plausible, CTO applies the contrastive adjustment only within the top-K_{cto} token set under the positive branch:

$$S^{(j)} = \mathrm{TopK}\left(\ell_{\mathrm{pos}}^{(j)}, K_{\mathrm{cto}}\right). \tag{8}$$

The gated CTO score for token i is

$$s_i^{(j)} = \begin{cases} \ell_{\mathrm{pos},i}^{(j)} - \alpha \ell_{\mathrm{neg},i}^{(j)}, & i \in S^{(j)}, \\ \ell_{\mathrm{pos},i}^{(j)}, & i \notin S^{(j)}. \end{cases} \tag{9}$$

where α controls negative-branch suppression. The shared sampler then applies the same temperature τ, nucleus mass, and sampler top-k filter to $s^{(j)}$. Thus, failure-associated continuations favored by the anti-expert branch are downweighted only within the positive-branch plausibility set.

Experience extraction, source selection, and ranking. Since CTO uses the cumulative experience bank $B^{(r)}$ to construct its decoding branches, its guidance quality depends on how trajectory experience is extracted, selected, and rendered, not only on the contrastive decoding rule. We therefore decompose experience construction into three stages: extraction, source selection, and ranking.

Extraction converts rollout trajectories into structured experience records stored in the experience bank. Each record may contain positive items, such as reusable propositions, constraints, and intermediate conclusions, as well as negative items, such as invalid transformations, misleading assumptions, and failed reasoning patterns. LLM-Judge extraction performs dense per-rollout analysis: each sampled trajectory is independently analyzed to extract

reusable propositions and critical pitfalls without using gold-reference correctness feedback. Direct CF extraction is an oracle-assisted diagnostic variant: it first partitions rollouts into successful and failed groups using reference-answer correctness, and then extracts contrastive items from matched successful–failed divergence pairs. Pseudo-CF replaces oracle correctness labels with answer self-consensus, forming majority/minority groups and adding counterfactual-style supplements to dense per-rollout LLM-Judge extraction.

Source selection defines which bank records are eligible for the current guided round before branch rendering. Sequential same-problem construction (Local-Seq.) admits only records accumulated for the current problem and retains their construction metadata, including search round, rollout index, and extractor output order. Same-problem retrieval (Local-Retr.) uses the same current-problem bank, but forms the candidate pool by retrieving records whose extracted contents are relevant to the current problem. Cross-problem retrieval (Cross-Retr.) expands the admissible source scope to records from other problems and retrieves a candidate pool using both the source problem and the extracted experience contents. All source-selection variants use the same branch budget, decoding hyperparameters, and rendering templates. This keeps the input budget and branch format fixed while varying the admissible experience scope and candidate-pool construction before branch rendering.

Ranking determines how records in a fixed candidate pool are prioritized and rendered into positive and negative branch prefixes. After source selection, the extracted propositions, constraints, and pitfalls are separated by polarity, then prioritized, aggregated, and truncated before entering P_{pos} and P_{neg}. Original preserves the construction order inherited from the experience bank: search round, rollout index, and extractor output order. Embedding retrieval ranks candidate records by semantic similarity to the current problem, using the problem statement as the query and the extracted propositions and pitfalls as retrieval content. Embedding retrieval with cross-encoder reranking (Emb.+Rerank) uses a two-stage procedure: embedding retrieval first forms an expanded candidate set from the fixed candidate pool, and a cross-encoder reranker then selects the final records for branch rendering. After prioritization, selected records are unpacked into polarity-specific contents, which are directly concatenated within each branch and truncated under fixed branch-visible rendering caps.

This decomposition isolates where experience quality enters CTO: extraction writes polarized records into the persistent bank, source selection forms the temporary candidate pool, and ranking determines which records become branch-specific prompts. It lets us test whether CTO benefits from better rollout-level extraction, more local candidate pools, or more aligned ranking and rendering before contrastive decoding.

4 Experiment

4.1 Experiment Setup

Unless otherwise stated, we follow the RSE evaluation protocol (Wang et al., 2026), aligning benchmark splits, rollout budget, decoding configuration, answer extraction, and reference-based final-answer evaluation for all compared rows. We evaluate CTO on Qwen3-4B-Thinking-2507 and Qwen3-30B-A3B-Thinking-2507, with Qwen3-4B-Instruct-2507 (Yang et al., 2025) as a non-Thinking control. Our main benchmarks are HMMT24 and HMMT25 from the RSE evaluation protocol, with additional results on HLE-Math-text (Phan et al., 2025). We use a three-iteration outer loop: Iter0 performs standard sampling (Wang et al., 2022), while Iter1–Iter2 synthesize experience from previous rollouts before guided decoding. Unless otherwise stated, CTO-base uses LLM-Judge extraction, same-problem retrieval (Local-Retr.) for source selection, embedding retrieval with cross-encoder reranking (Emb.+Rerank) for ranking, and direct concatenation for branch rendering. During contrastive decoding, CTO-base uses suppression coefficient $\alpha = 0.55$ and positive-branch plausibility filtering with $K = 8$. CTO-base keeps the RSE search protocol fixed, but replaces prompt-level reuse with paired contrastive decoding branches. Figure 2 compares CTO-base with Standard Sampling (Wang et al., 2022) for unguided multi-rollout performance, Self-Refine (Madaan et al., 2023) for iterative self-feedback refinement, PaCoRe (Hu et al., 2026) for direct concatenation of historical reasoning information, and prompt-level RSE (Wang et al., 2026) for single-prompt experience reuse. Baseline provenance is detailed in Appendix A.6: external contextual baselines are inherited when protocol fields exactly match, while the Standard Sampling, RSE, and CTO-base rows used for the direct utilization-mechanism com-

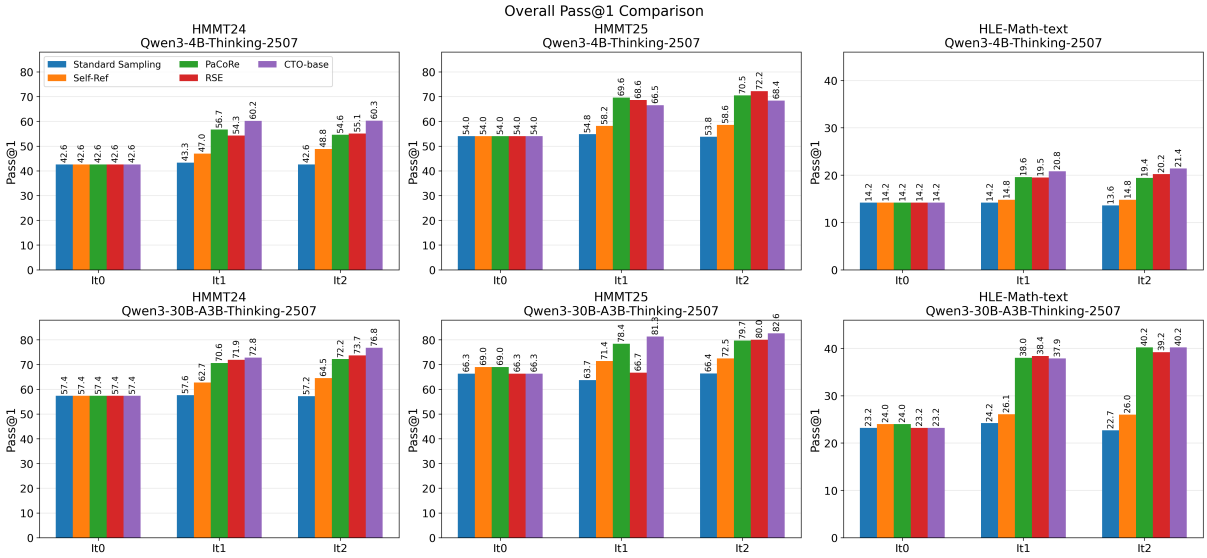

Figure 2: Overall Pass@1 under matched multi-round search. Bars are grouped by iteration and labeled with exact values. CTO-base is best or near-best in most final-iteration settings, while remaining non-uniform across benchmarks; this motivates the subsequent analysis of extraction, source selection, and ranking.

parison are locally evaluated in the same pipeline. Unless stated otherwise, we report Pass@1 after each iteration using 32 rollouts per problem. Appendix A provides the protocol, hyperparameters, retrieval/deduplication rules, scoring details, and paired bootstrap estimates for the final CTO-base–RSE comparison.

4.2 Main Results

4.2.1 Overall Performance

Figure 2 reports the protocol-matched main comparison on the two Thinking backbones. CTO-base achieves the best or tied-best final Pass@1 in most settings, showing strong overall performance across model scales and mathematical reasoning benchmarks. This suggests that converting accumulated rollout experience into contrastive decoding branches can effectively guide later search. However, CTO-base is not a uniformly dominant replacement for prompt-level reuse; it is most effective when selected experience is aligned enough for positive guidance and safe enough for negative suppression. We next isolate these conditions through experience extraction, source selection, and ranking ablations. For completeness, we also evaluate Qwen3-4B-Instruct-2507 as a non-Thinking control and report its full results in Appendix C.1. The Instruct-model results are less consistent than those on Thinking backbones, suggesting that experience-guided iteration benefits more from rollouts with richer intermediate reasoning traces.

Table 1: Effect of experience extraction from rollouts. LLM-Judge performs dense per-rollout extraction, Direct CF uses oracle successful–failed pairs, and Pseudo-CF adds self-consensus counterfactual supplements. Averages show that LLM-Judge is the most stable extraction strategy.

Method	HMMT24			HMMT25			Avg.	
	iter0	iter1	iter2	iter0	iter1	iter2	iter1	iter2
Qwen3-4B-Thinking-2507								
Standard Sampling	42.6	43.3	42.6	54.0	54.8	53.8	49.1	48.2
RSE	42.6	54.3	55.1	54.0	**68.6**	**72.2**	61.5	63.7
CTO w/ LLM-Judge	42.6	**60.2**	**60.3**	54.0	66.5	68.4	**63.4**	**64.4**
CTO w/ Direct CF	42.6	42.6	43.3	54.0	58.5	57.6	50.6	50.5
CTO w/ Pseudo-CF	42.6	56.6	57.3	54.0	66.7	70.3	61.7	63.8
Qwen3-30B-A3B-Thinking-2507								
Standard Sampling	57.4	57.6	57.2	66.3	63.7	66.4	60.7	61.8
RSE	57.4	71.9	73.7	66.3	66.7	80.0	69.3	76.9
CTO w/ LLM-Judge	57.4	**72.8**	**76.8**	66.3	**81.3**	**82.6**	**77.1**	**79.7**
CTO w/ Direct CF	57.4	59.4	56.2	66.3	65.5	70.2	62.5	63.2
CTO w/ Pseudo-CF	57.4	72.7	75.4	66.3	76.2	78.5	74.5	77.0

4.2.2 Experience Extraction: Safer Negative Signals Matter

Table 1 compares three extraction strategies for constructing positive and negative experience from rollouts. LLM-Judge is the most stable strategy, achieving the best Iter1 and Iter2 averages for both Thinking backbones. Direct CF performs substantially worse despite using oracle successful–failed partitions, suggesting that correctness labels alone do not produce safe negative signals. Because Direct CF extracts local divergence patterns from a small number of successful–failed pairs, it may capture incidental trajectory differences rather than stable failure modes; once placed in the negative branch, such items can suppress useful reasoning

Table 2: Effect of source-selection policies under fixed CTO decoding, branch budget, and rendering templates. Because policies vary both admissible source scope and candidate-pool construction, this is a policy-level locality analysis. Same-problem policies are more reliable than cross-problem retrieval.

Method	HMMT24			HMMT25			Avg.	
	iter0	iter1	iter2	iter0	iter1	iter2	iter1	iter2
Qwen3-4B-Thinking-2507								
Standard Sampling	42.6	43.3	42.6	54.0	54.8	53.8	49.1	48.2
RSE	42.6	54.3	55.1	54.0	68.6	72.2	61.5	63.7
CTO w/ Local-Seq.	42.6	54.4	54.4	54.0	**69.3**	**73.8**	61.9	64.1
CTO w/ Local-Retr.	42.6	**60.2**	**60.3**	54.0	66.5	68.4	**63.4**	**64.4**
CTO w/ Cross-Retr.	42.6	48.7	48.4	54.0	50.4	55.6	49.6	52.0
Qwen3-30B-A3B-Thinking-2507								
Standard Sampling	57.4	57.6	57.2	66.3	63.7	66.4	60.7	61.8
RSE	57.4	71.9	73.7	66.3	66.7	80.0	69.3	76.9
CTO w/ Local-Seq.	57.4	71.5	68.3	66.3	79.1	80.0	75.3	74.2
CTO w/ Local-Retr.	57.4	**72.8**	**76.8**	66.3	**81.3**	**82.6**	**77.1**	**79.7**
CTO w/ Cross-Retr.	57.4	55.0	56.0	66.3	62.7	67.5	58.9	61.8

steps. Pseudo-CF recovers much of the performance of dense extraction, but does not consistently surpass LLM-Judge. Thus, extraction should prioritize dense coverage and suppression-safe negatives over merely forming explicit positive–negative contrasts.

4.2.3 Source Selection: Locality Matters More Than Pool Size

Table 2 compares source-selection policies under fixed outer-loop, decoding, branch-budget, and rendering settings. By varying the admissible experience scope and candidate-pool construction before branch rendering, it examines how locality and pool composition affect CTO guidance. The results show a clear locality preference: same-problem source variants are generally more reliable than cross-problem retrieval, indicating that expanding the candidate pool does not automatically improve decoding-time guidance. Within the local variants, Local-Retr. and Local-Seq. show complementary behavior rather than a uniform ordering. Local-Retr. can help when relevance filtering improves the alignment between accumulated same-problem records and the current decoding round, while Local-Seq. remains competitive when chronological same-problem experience is already local and low-noise. By contrast, Cross-Retr. gives weaker or less stable gains, suggesting that transferred propositions and pitfalls can mismatch the current problem's constraints, solution structure, or failure modes. Thus, locality matters because it improves problem alignment and suppression safety, not because it simply changes candidate-pool size.

Table 3: Effect of experience ranking before CTO decoding. Original preserves bank order, Embedding ranks by semantic similarity, and Emb.+Rerank adds cross-encoder reranking. Ranking helps when it improves problem alignment, but gains are not monotonic.

Method	HMMT24			HMMT25			Avg.	
	iter0	iter1	iter2	iter0	iter1	iter2	iter1	iter2
Qwen3-4B-Thinking-2507								
Standard Sampling	42.6	43.3	42.6	54.0	54.8	53.8	49.1	48.2
RSE	42.6	54.3	55.1	54.0	68.6	72.2	61.5	63.7
CTO w/ Original	42.6	54.4	54.4	54.0	**69.3**	**73.8**	61.9	64.1
CTO w/ Embedding	42.6	54.3	56.4	54.0	65.9	67.4	60.1	61.9
CTO w/ Emb.+Rerank	42.6	**60.2**	**60.3**	54.0	66.5	68.4	**63.4**	**64.4**
Qwen3-30B-A3B-Thinking-2507								
Standard Sampling	57.4	57.6	57.2	66.3	63.7	66.4	60.7	61.8
RSE	57.4	71.9	73.7	66.3	66.7	80.0	69.3	76.9
CTO w/ Original	57.4	71.5	68.3	66.3	79.1	80.0	75.3	74.2
CTO w/ Embedding	57.4	**75.4**	73.8	66.3	80.2	81.0	**77.8**	77.4
CTO w/ Emb.+Rerank	57.4	72.8	**76.8**	66.3	**81.3**	**82.6**	77.1	**79.7**

4.2.4 Experience Ranking: Alignment Matters More Than Complexity

Table 3 compares three ranking strategies for organizing experience before CTO decoding. The results show that more complex ranking does not monotonically improve CTO. For Qwen3-4B, Emb.+Rerank gives the best average results, mainly by improving HMMT24, but Original remains stronger on HMMT25. For Qwen3-30B, Embedding achieves the best Iter1 average, while Emb.+Rerank gives the best Iter2 average, showing that reranking is useful but not uniformly dominant. These mixed results suggest that ranking helps when it improves the alignment between selected experience and the current problem, rather than simply because it adds retrieval complexity. In particular, plain embedding ranking can help the stronger model by filtering same-problem records, but it underperforms Original on Qwen3-4B when used without reranking, indicating that semantic similarity alone may still prioritize plausible-looking but weakly causal items. Thus, ranking is best viewed as an alignment filter for branch construction: it helps only when it preserves problem-relevant and polarity-specific evidence before contrastive decoding.

5 Diagnostic Analysis Beyond Pass@1

Pass@1 provides a compact measure of final-answer accuracy, but it does not explain why experience-guided search improves or fails. In particular, it does not reveal whether gains come from higher compute cost, safer negative signals, or the interaction between positive-branch reinforcement and negative-branch suppression. We therefore complement Pass@1 with compute-aware account-

Table 4: Compute-aware accounting on HMMT24 with Qwen3-4B-Thinking. Gen. tok. and Proc. tok. denote generated and total processed tokens; P_0 and P_m denote initial and final Pass@1, reported as percentages. ΔP is reported in percentage points for readability, while $G_{\text{tok}} = \Delta P_{\text{frac}}/(C/10^6)$ is computed from the fractional Pass@1 gain per million processed tokens. CTO yields the largest absolute gain but costs more processed tokens, whereas RSE is more token-efficient.

Method	Gen. tok. (10^6)	Proc. tok. (10^6)	P_0 (%)	P_m (%)	ΔP (pp)	G_{tok}
Standard	79.489	79.500	42.6	42.6	0	0
RSE	71.340	71.820	42.6	55.1	+12.5	$+1.7\times10^{-3}$
CTO-base	56.130	129.093	42.6	60.3	+17.7	$+1.4\times10^{-3}$

ing, experience-quality diagnostics, and branch-sensitivity ablations over branch usage, suppression strength, and plausibility filtering.

5.1 Compute-Aware Efficiency

We quantify inference cost through offline token accounting on HMMT24 with the Qwen3-4B-Thinking backbone. For each method m, let C_m denote the total processed tokens accumulated by its evaluated search schedule. Standard Sampling uses a single unguided stream, $C_{\text{Std}} = C_{\text{prompt}} + C_{\text{gen}}$; prompt-level reuse methods such as RSE use one experience-augmented stream, $C_{\text{RSE}} = C_{\text{prompt+exp}} + C_{\text{gen}}$; and CTO evaluates two experience-conditioned branches before the shared continuation:

$$C_{\text{CTO}} = C_{\text{pos}} + C_{\text{neg}} + C_{\text{gen}}.$$

We report the cost-normalized gain

$$G_{\text{tok}} = \frac{P_m - P_0}{C_m/10^6},$$

where P_m and P_0 denote final and initial Pass@1. For G_{tok}, we use fractional Pass@1 values in $[0, 1]$; the table reports P_0 and P_m as percentages and ΔP in percentage points only for readability.

Table 4 shows that CTO incurs higher processed-token cost than prompt-level reuse because both experience branches are evaluated during decoding. The generated–processed token gap clarifies the source of this cost. Standard Sampling has nearly identical generated and processed tokens because it uses a single unguided stream; RSE has a larger gap because accumulated experience is prefixed into the guided prompt; CTO has the largest gap because positive and negative branches are both evaluated during decoding. This extra cost is not reflected in longer visible traces: CTO generates fewer output tokens than both Standard Sampling and RSE, while achieving the highest final Pass@1 in this setting. Thus, CTO trades token efficiency for stronger absolute improvement: RSE is more efficient under G_{tok}, while paired decoding uses additional processed tokens to obtain larger accuracy gains.

Method	P_- ↑	R_{fs} ↓	R_{contr} ↓
CTO w/ Direct CF	**0.915**	0.298	**0.002**
CTO w/ LLM-Judge	0.885	0.240	0.008
CTO w/ Pseudo-CF	0.880	**0.203**	0.010

Table 5: Experience-quality diagnostics on HMMT24 with Qwen3-4B-Thinking. P_- measures valid negative pitfalls, R_{fs} measures false-suppression risk, and R_{contr} measures within-problem contradiction. Direct CF has high negative precision but the highest false-suppression risk.

5.2 Experience Quality

We further audit the deduplicated experience bank on HMMT24 with Qwen3-4B-Thinking-2507 to examine whether different annotation strategies produce reliable negative signals. All independent audit judgments in this section use Kimi-K2.5 as the judge model. The audit judge evaluates sampled proposition-level pitfalls under the corresponding problem context. Negative precision P_- measures whether sampled negative items are valid pitfalls: $P_- = n_{\text{valid}}/n_-$, where n_{valid} and n_- denote the numbers of valid negative items and sampled negative items. False-suppression rate R_{fs} measures whether a negative item is also supported by correct rollouts of the same problem: $R_{\text{fs}} = n_{\text{fs}}/n_-$. Higher R_{fs} indicates lower safety when negative experience is used as a CTO suppression signal. We also report within-problem contradiction R_{contr}, which measures inconsistent warnings among comparable same-problem negative-item pairs. Full prompts, sampling rules, and metric details are provided in Appendix C.2.

Table 5 shows that CTO performance is better explained by suppression safety than by surface-level negative validity alone. Direct CF achieves the

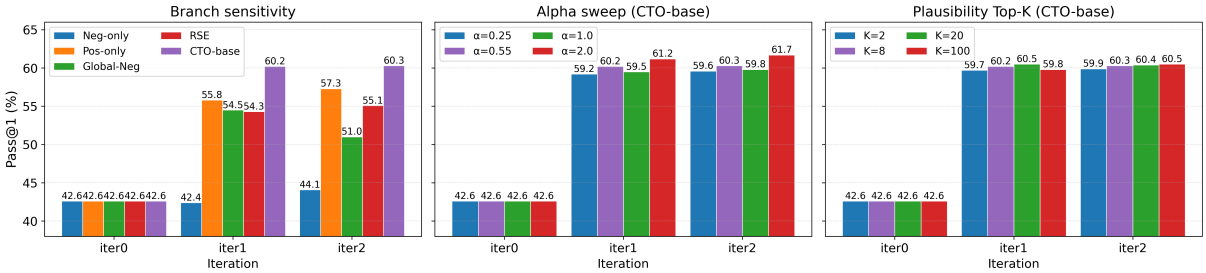

Figure 3: Reuse-mechanism ablation on HMMT24 with Qwen3-4B-Thinking-2507 under matched search settings. (Left) Full CTO outperforms branch-usage ablations, showing that paired branch control requires problem-local negative evidence rather than prompt-level reuse or generic cross-problem suppression. (Middle) The α sweep shows that stronger suppression can help when negative items are reliable. (Right) Sweeping the plausibility range K shows mild sensitivity, suggesting that the positive branch mainly acts as a token-level plausibility gate for contrastive intervention.

highest P_-, indicating that its negative items are often valid local descriptions of failed trajectories. However, Direct CF also has the highest R_{fs}, meaning that many of its pitfalls overlap with reasoning patterns used by correct rollouts. Once placed in the negative branch, such items can suppress useful continuations, explaining why Direct CF performs poorly despite high negative precision. Pseudo-CF gives the lowest R_{fs}, suggesting that agreement-based grouping filters part of the incidental trajectory noise introduced by direct successful–failed contrasts. The uniformly low R_{contr} values indicate that explicit within-problem contradiction is rare; the practical lesson is that negative experience should be audited for false suppression, not only for whether it describes a real failure.

5.3 Branch Sensitivity

To isolate the source of CTO's gains, Figure 3 compares branch-usage variants under the same outer-loop search protocol. CTO-base uses the full paired-branch mechanism with local positive evidence and problem-local negative suppression. Neg-only CTO uses only the negative suppression branch, Pos-only CTO uses only the positive guidance branch, RSE renders positive and negative experience as single-prompt textual hints, and Global-Neg CTO keeps the local positive branch but replaces problem-local negatives with cross-problem negatives. We also sweep the suppression coefficient α and the plausibility range K.

The results show that CTO requires both positive guidance and problem-local negative suppression. Neg-only CTO stays close to the unguided baseline, while Pos-only CTO improves over RSE but remains below CTO-base, indicating that positive branch guidance is useful but incomplete. Global-Neg CTO substantially weakens performance despite using the same local positive branch and contrastive scoring rule, showing that negative-branch gains come from current-problem failure evidence rather than generic anti-expert pressure. The α sweep further shows that stronger suppression is useful only when negative items are reliable, and the K sweep shows mild sensitivity, consistent with the positive branch acting mainly as a plausibility gate. Thus, CTO is not simply contrastive decoding with an anti-expert; it works when positive guidance is paired with problem-local, suppression-safe negative evidence.

6 Conclusion

We introduced Contrastive Trajectory Optimization (CTO), a decoding-time mechanism that converts trajectory-derived positive and negative experience into paired contrastive branches for memory-augmented inference-time search. Unlike prompt-level reuse, CTO uses positive experience to guide plausible continuations and problem-local negative experience to suppress failure-associated ones. Across mathematical reasoning benchmarks, CTO improves reasoning-oriented models in most settings. Overall, our diagnostics suggest that CTO gains from paired branch control: positive experience guides plausible continuations, while local, suppression-safe negative experience enables suppression, rather than broader retrieval, ranking complexity, or generic anti-expert pressure.

Limitations

Due to experimental cost and space constraints, our experiments cover a limited set of model families

and mathematical reasoning benchmarks, report the main experience-guided search results up to Iter2, and provide paired problem-level bootstrap estimates in place of attaching full confidence intervals to every main-table entry. Our analysis focuses on CTO design choices rather than exhaustive comparisons with all contrastive decoding variants, although Global-Neg CTO tests whether CTO reduces to generic anti-expert suppression. CTO also incurs higher processed-token cost than prompt-level reuse, motivating more efficient branch evaluation, caching, routing, and selective activation.

References

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*.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James Glass, and Pengcheng He. 2023. Dola: Decoding by contrasting layers improves factuality in large language models. *arXiv preprint arXiv:2309.03883*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Jingcheng Hu, Yinmin Zhang, Shijie Shang, Xiaobo Yang, Yue Peng, Zhewei Huang, Hebin Zhou, Xin Wu, Jie Cheng, Fanqi Wan, and 1 others. 2026. Pacore: Learning to scale test-time compute with parallel coordinated reasoning. *arXiv preprint arXiv:2601.05593*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 13358–13376.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori B Hashimoto, Luke Zettlemoyer, and Mike Lewis. 2023. Contrastive decoding: Open-ended text generation as optimization. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 12286–12312.

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*.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A Smith, and Yejin Choi. 2021. Dexperts: Decoding-time controlled text generation with experts and anti-experts. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6691–6706.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, and 1 others. 2025. Humanity's last exam. *arXiv preprint arXiv:2501.14249*.

Guillaume Sanchez, Honglu Fan, Alexander Spangher, Elad Levi, Pawan Sasanka Ammanamanchi, and Stella Biderman. 2023. Stay on topic with classifier-free guidance. *arXiv preprint arXiv:2306.17806*.

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

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

Xinglin Wang, Jiayi Shi, Shaoxiong Feng, Peiwen Yuan, Yiwei Li, Yueqi Zhang, Chuyi Tan, Ji Zhang, Boyuan Pan, Yao Hu, and 1 others. 2026. Do not waste your rollouts: Recycling search experience for efficient test-time scaling. *arXiv preprint arXiv:2601.21684*.

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*.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom 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.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

A Replication Protocol, Evaluation, and Baseline Provenance

This appendix records implementation details needed to reproduce the experiments in Section 4. The main paper already specifies the evaluated models, benchmark suites, baseline set, three-iteration search protocol, and headline CTO-base configuration. Here we focus on the operational details that are not fully expanded in the main text: rollout scheduling, experience-bank construction, retrieval and deduplication rules, scoring, uncertainty estimation, and compute accounting. Unless explicitly stated otherwise in an ablation, all methods follow the same RSE (Wang et al., 2026) benchmark splits, answer-normalization scripts, rollout budget, decoding hyperparameters, answer extraction, and reference-based final-answer evaluation.

A.1 Run schedule and experience flow

Iteration 0. For each benchmark instance, we sample 32 independent unguided trajectories using the decoding hyperparameters in Table 6. These Iter0 trajectories form the initial search history. In protocol-matched comparisons between CTO-base and prompt-level reuse, the methods use the same Iter0 trajectory pool whenever possible; any row requiring an independent random seed is explicitly noted.

Quantity	Default value
Sampling temperature τ	0.6
Nucleus mass top-p	0.95
Trajectories per item per problem	32
Max. fresh tokens per trajectory	38,912
Retrieval width before truncation	48 records per question
Rendering caps	96 propositions and 96 pitfalls
CTO suppression coefficient α	0.55
Sampling clipped top-k	20
CTO plausibility gate top-K	8
Deduplication cosine threshold	0.8

Table 6: Core hyperparameters for headline CTO-base runs. Ablations modify only the factor under study while keeping the remaining rollout, decoding, retrieval, and scoring settings fixed unless explicitly stated.

Experience synthesis. After each stochastic batch terminates, every trajectory is processed by the same answer parser used for evaluation and is then passed to the experience extractor. For CTO-base and other non-oracle variants, the extractor receives only the problem statement, the sampled trajectory, and the extracted final answer. It does not receive the gold answer, the reference solution, or a gold-derived correctness label. Gold references are used only by the final-answer evaluator. Oracle correctness labels are used only in diagnostic counterfactual variants, where successful–failed trajectory pairs must be formed by construction.

Guided iterations. Before guided round $r \in \{1, 2\}$, experience records extracted from earlier rounds are merged into a pre-round record bank:

$$B^{(r)} = \mathrm{Merge}\Big(\mathcal{E}^{(<r)}\Big) .$$

Each record may contain positive and/or negative fields. We write $B_{+}^{(r)}$ and $B_{-}^{(r)}$ for the polarity-specific views used during rendering. Prompt-level reuse renders this bank into a single experience-augmented prompt. CTO instead renders the positive and negative banks into paired decoding branches. Both mechanisms then generate another 32 trajectories per benchmark instance under the same sampling temperature, nucleus mass, top-k filter, and maximum generation length. After Iter1, newly generated trajectories are annotated, merged with the surviving bank, and used to construct the Iter2 guided run.

A.2 Experience-bank construction

Deduplication. We embed extracted free-text items with the `all-MiniLM-L6-v2` sentence encoder. A new item is merged into an existing centroid if its cosine similarity to that centroid exceeds 0.8, or if one item's surface string is substring-dominated by the other. Accepted centroids preserve first-seen order, making downstream truncation deterministic when retrieval scores tie.

Retrieval and reranking. For CTO-base, source selection defines the admissible source scope and candidate-construction pipeline. Specifically, Local-Retr. restricts admissible records to the current problem history and uses record-level MiniLM retrieval as the first-stage candidate construction step. Emb.+Rerank denotes the ordering/reranking stage within this pipeline: it uses an expanded MiniLM shortlist followed by cross-encoder selection before final polarity unpacking and rendering. Without reranking, MiniLM cosine retrieval selects up to 48 candidate records for rendering. When cross-encoder reranking is enabled, MiniLM first retrieves an expanded record-level candidate set of up to 8×48 records, and an MS MARCO MiniLM pairwise reranker then selects up to 48 final records for rendering. After record-level retrieval and optional reranking, the selected records are unpacked by polarity and trimmed to at most 96 affirmed propositions and 96 pitfalls before entering the branch templates.

Budget interpretation. The 48-record retrieval width refers to the final records passed to rendering; when reranking is enabled, the 8×48 expanded candidate set is used only as an intermediate pool for cross-encoder selection. The 96/96 caps are applied after the final records are unpacked into polarity-specific propositions and pitfalls. Thus, the retrieval width controls the number of selected experience records, while the rendering caps control the amount of branch-visible positive and negative content. No additional length cap is imposed on synthesized experience beyond the decoding and rendering limits listed in Table 6.

A.3 Decoding substrate

All decoding runs use causal Transformer backbones served through batched stochastic inference. Prompt-level reuse uses a single decoding stream with an experience-augmented prefix. CTO evaluates positive and negative branch prefixes against the same generated continuation prefix, applies the contrastive scoring rule within the positive-branch plausibility set, and then samples from the resulting gated score vector using the shared sampler. No model weights are updated during this process.

For CTO-base, the suppression coefficient is $\alpha = 0.55$, and the plausibility gate uses the top $K = 8$ tokens under the positive branch. Prompt-level reuse and CTO share the same sampling entropy, trajectory count, decoder horizon, answer parser, normalization script, and final-answer evaluator. The only intended difference is the experience-utilization mechanism: single-prompt reuse versus paired branch-level distributional guidance.

We distinguish two top-k quantities that operate at different stages of decoding. The sampler clipped top-k, denoted $k_{\text{sampler}} = 20$, is a standard stochastic decoding hyperparameter shared by Standard Sampling, prompt-level reuse, and CTO. After a method-specific next-token score vector has been formed, the shared sampler applies the same temperature, nucleus mass, and sampler top-k filtering before sampling the next token.

By contrast, the CTO plausibility gate $K_{\text{cto}} = 8$ is specific to paired contrastive decoding. It determines the positive-branch top-K_{cto} token set on which the contrastive adjustment is applied. Tokens outside this set are not contrastively modified; they fall back to the positive-branch scores and are then handled by the same downstream sampler. Thus, k_{sampler} controls stochastic sampling diversity, whereas K_{cto} controls where negative-branch suppression is allowed to intervene.

A.4 Scoring and uncertainty estimation

For each trajectory, we extract the final answer with a deterministic parser, normalize it with the benchmark-specific normalization script, and compare it against the gold reference using the shared math-equivalence checker from the RSE evaluation protocol. The checker evaluates mathematical equivalence after normalization rather than exact string identity. The same parser–normalizer–checker pipeline is used for all methods and all iterations unless explicitly stated otherwise.

For method m and problem i, let

$$p_i^{(m)} = \frac{1}{R} \sum_{j=1}^{R} \mathbb{1}\left[\hat{a}_{ij}^{(m)} \equiv a_i\right],$$

where $R = 32$, $\hat{a}_{ij}^{(m)}$ is the extracted answer from rollout j, a_i is the gold answer, and $\equiv$ denotes

equivalence under the evaluator. The reported Pass@1 is

$$\mathrm{Pass@1}^{(m)} = \frac{1}{N}\sum_{i=1}^{N} p_i^{(m)}.$$

All main results use gold-reference adjudication unless explicitly labeled otherwise.

Because methods are evaluated on matched benchmark instances, we estimate uncertainty with paired bootstrap over problems. We resample the N benchmark problems with replacement for 10,000 bootstrap trials and recompute both Pass@1 and the paired difference

$$\Delta = \mathrm{Pass@1}_{\mathrm{CTO}} - \mathrm{Pass@1}_{\mathrm{RSE}}.$$

We report percentile 95% confidence intervals and the paired bootstrap win rate $\Pr(\Delta > 0)$. The bootstrap unit is the problem, not the individual rollout, so all rollouts from the same problem remain grouped within each resampled replicate.

A.5 Engineering parity and compute budget

The headline Thinking-model results stop after Iter2, yielding three stochastic checkpoints: Iter0, Iter1, and Iter2. Auxiliary studies that launch an additional guided round, change the suppression coefficient, or alter the plausibility range are treated as separate diagnostic runs and are not mixed with the headline protocol.

For long completions, some 30B prompt-reuse jobs require wider serving contexts to avoid truncating heterogeneous traces. Such serving changes affect only engine-side capacity management; they do not change the sampling temperature, nucleus mass, top-k, trajectory count, maximum generation length, answer extraction, or evaluation script. CTO uses a separate serving policy that reserves a larger effective context window for paired branch evaluation and KV-cache stability.

All experiments were run on 8× NVIDIA H20 141 GB or 8× NVIDIA A800 80 GB GPU nodes. The full experimental sweep consumed approximately 900 GPU-hours, including Iter0 sampling, Iter1–Iter2 guided decoding, LLM-Judge experience extraction, embedding retrieval, cross-encoder reranking, and diagnostic ablations over extraction, source selection, ranking, branch usage, α, and plausibility filtering K. As a reference point, one complete HMMT24/Qwen3-4B-Thinking CTO-base run from Iter0 through Iter2, covering 30 problems with 32 rollouts per problem per iteration, required roughly 40 GPU-hours on a single GPU when experience banks were not shared across conditions.

Artifact use. We use publicly available model, benchmark, and evaluation artifacts, including Qwen3 models, the RSE evaluation protocol, HMMT24/HMMT25 splits, HLE-Math-text, answer-normalization scripts, and embedding/reranking models. These artifacts are used only for research evaluation under their intended benchmark or model-evaluation settings. We do not collect new human-subject data or introduce datasets containing personal identifiers; the evaluated data consist of mathematical reasoning problems.

A.6 Baseline Sources and Local Re-evaluation

This appendix clarifies the provenance of baseline numbers used in the main comparison. Our goal is to keep the comparison aligned with the RSE (Wang et al., 2026) protocol while avoiding unnecessary reimplementation variance.

For baselines whose reported settings exactly match ours, including the model, benchmark split, rollout budget, iteration schedule, answer extraction, and evaluation script, we inherit the corresponding numbers from the RSE report. These inherited rows are used as protocol-matched contextual baselines: they follow the same evaluation fields, but they are not local reimplementations. This applies to external contextual baselines such as Self-Refine and PaCoRe; RSE rows are inherited only when not used for paired local mechanism diagnostics.

We locally re-evaluate rows needed for the direct mechanism comparison between prompt-level and decoding-time experience reuse. In particular, we rerun Standard Sampling across Iter0–Iter2 when a repeated-sampling control is needed beyond the initial round reported by RSE. We also locally re-evaluate RSE in settings where implementation-dependent deviations could affect the direct comparison with CTO-base. In these cases, Standard Sampling, RSE, and CTO-base are produced by the same local pipeline, using the same answer parser, normalization script, and reference-based evaluator.

Thus, "same evaluation protocol" has a specific meaning in this paper. It means that compared rows are aligned in problem split, rollout budget, iteration schedule, answer extraction, and reference-

based scoring whenever applicable. It does not imply that every external baseline was reimplemented in our codebase. Accordingly, Figure 2 combines inherited protocol-matched baselines for context with locally re-evaluated Standard/RSE/CTO rows for the controlled utilization-mechanism comparison.

A.7 Paired Problem-Level Bootstrap Confidence Intervals

This section reports paired problem-level bootstrap estimates for the final-iteration CTO-base–RSE comparison. The bootstrap procedure follows Appendix A.4: benchmark problems are resampled with replacement, all rollouts from the same problem remain grouped, and the paired difference is computed as

$$\Delta = \text{Pass@1}_{\text{CTO}} - \text{Pass@1}_{\text{RSE}}.$$

We report Δ and its percentile 95% confidence interval in percentage points, together with the bootstrap win rate $\Pr(\Delta > 0)$.

Table 7 shows a generally positive but non-uniform final-iteration advantage for CTO-base over RSE. CTO-base has a positive point estimate in five of the six settings, but the confidence interval is fully above zero only on HMMT24 with the 4B model and HMMT25 with the 30B model. The HMMT24 result with the 30B model also has a high bootstrap win rate, although its interval crosses zero. By contrast, the HMMT25 result with the 4B model favors RSE, and both HLE-Math-text settings have wide intervals, indicating higher uncertainty on the heterogeneous benchmark.

Overall, these estimates support the main observation that CTO-base often improves over prompt-level RSE, while also showing that the strength of evidence varies across benchmarks and model scales.

B Additional Method Details

This appendix gives implementation details for the method variants used in the controlled ablations. We avoid restating the full CTO decoding rule from Section 3 and instead specify how each variant constructs the experience shown to the decoder.

B.1 Prompt-Level Experience Reuse Baseline

We use RSE (Wang et al., 2026) as the matched prompt-level experience-reuse baseline. At guided round r, RSE renders the accumulated positive and negative banks into a single natural-language prompt,

$$P_{\text{RSE}} = \text{Template}_{\text{RSE}}(q, B_{+}^{(r)}, B_{-}^{(r)}).$$

Positive items are presented as reusable findings, constraints, or intermediate conclusions; negative items are presented as pitfalls or failure patterns to avoid. All outer-loop quantities are kept matched with CTO, including the problem split, iteration schedule, rollout budget, decoding hyperparameters, maximum generation length, answer extraction, and evaluation script. Thus, the RSE–CTO comparison changes only the utilization mechanism: single-prompt instruction following versus paired branch-level decoding guidance.

B.2 Experience Source

We compare three source policies while keeping the branch budget and rendering format fixed.

Local-Seq. Local-Seq. uses only deduplicated experience accumulated from earlier rounds of the same problem. It performs no explicit retrieval. Items are ordered by search round, rollout index, and extractor output order before truncation. This setting preserves the strongest problem locality and serves as the chronological same-problem source policy.

Local-Retr. Local-Retr. also restricts the admissible source scope to the current problem history, but constructs the candidate pool by retrieving experience records before branch rendering. Consistent with the boundary definition in Appendix A.2, Local-Retr. specifies the admissible same-problem scope and the record-level candidate-construction pipeline, while Emb.+Rerank specifies the ordering and reranking procedure used within that pipeline. The problem statement q is used as the retrieval query, and the extracted contents of each record are used as retrieval content. Retrieval therefore operates at the record level: it selects candidate records from the current-problem bank, optionally through an expanded MiniLM shortlist followed by cross-encoder selection. After retrieval and optional reranking, the selected records are unpacked into polarity-specific propositions and pitfalls, separated into positive and negative branch contents, and then truncated under the corresponding branch budgets.

Cross-Retr. Cross-Retr. expands the admissible pool to experience from other problems. The re-

Table 7: Final-iteration paired problem-level bootstrap estimates for CTO-base against RSE. Δ denotes CTO-base minus RSE in percentage points.

Benchmark	**Model**	**Pass@1 CTO / RSE**	**Δ (pp)**	**95% CI (pp)**	$\Pr(\Delta > 0)$
HMMT24	4B	60.3/55.1	+5.2	[+1.6, +5.3]	100.0%
HMMT25	4B	68.4/72.2	−3.8	[−16.0, +6.7]	21.0%
HLE-Math-text	4B	21.4/20.2	+1.2	[−7.2, +19.3]	80.1%
HMMT24	30B	76.8/73.7	+3.1	[−0.9, +21.5]	96.2%
HMMT25	30B	82.6/80.0	+2.6	[+1.8, +4.5]	98.7%
HLE-Math-text	30B	40.0/39.2	+0.8	[−8.2, +8.7]	52.0%

trieval key includes both the source problem and the extracted experience text. To reduce leakage or contamination, we filter source problems with the same benchmark identifier or exact source-problem match when metadata is available; otherwise, we embed source-problem statements with all-MiniLM-L6-v2 and filter near-duplicate source problems using the same cosine threshold of 0.8 as bank deduplication. Because the branch budget is kept identical to local-source settings, this variant tests cross-problem transfer of experience rather than the effect of a larger input budget.

B.3 Experience Annotation

We instantiate the following annotation variants. All variants use the same rollout pool and item budget unless explicitly stated otherwise.

LLM-Judge. LLM-Judge analyzes each rollout independently and extracts positive items, such as reusable propositions, constraints, and local conclusions, together with negative items, such as invalid transformations, misleading assumptions, and critical pitfalls. For CTO-base, the judge receives only the problem, generated solution trace, and extracted final answer. It does not receive the gold answer, reference solution, or gold-derived correctness label. This is the default annotation scheme.

Direct CF. Direct CF is an oracle-assisted diagnostic variant. It first partitions rollouts into successful and failed groups using reference-answer correctness, then forms successful–failed trajectory pairs. Differences leading toward the successful trajectory are converted into positive experience, while differences associated with the failed trajectory are converted into negative experience. This variant tests whether explicit successful–failed contrast is sufficient without an independent semantic judge.

Pseudo-CF. Pseudo-CF keeps dense per-rollout LLM-Judge extraction as the base extractor and adds counterfactual-style supplements derived from answer self-consensus. Instead of using oracle successful–failed labels, it groups trajectories by their extracted answers: trajectories agreeing with the majority extracted answer provide pseudo-positive support, while trajectories producing minority answers provide pseudo-negative contrast candidates. Shared reasoning patterns from the pseudo-positive group and divergent patterns from the pseudo-negative group are converted into supplementary positive and negative items, respectively. These supplements are merged with the LLM-Judge extracted items before deduplication and bank construction. This variant tests whether weak agreement-based supervision can add counterfactual-style evidence in an online test-time setting while retaining the dense coverage of LLM-Judge extraction.

B.4 Experience Organization

The organization axis controls how selected experience records are ordered, compressed, and truncated before entering the positive and negative branch prompts.

Ranking. Original ordering preserves the record order inherited from the bank: search round, rollout index, and extractor output order. Embedding ranking scores each candidate record against the current problem statement, using the record's extracted propositions and pitfalls as retrieval content, and ranks records before branch rendering. Emb.+Rerank first retrieves an expanded set of candidate records by embedding similarity, then applies a cross-encoder reranker to select the final records before truncation. After record-level ranking, selected records are unpacked into polarity-specific propositions and pitfalls, which are then aggregated and truncated under the positive and negative branch budgets.

C Supplementary Results and Diagnostic Protocols

This appendix groups supplementary results and diagnostic protocols that support the main empirical claims without changing the core replication protocol.

C.1 Results on the Instruct Model

Table 8 reports supplementary results on Qwen3-4B-Instruct-2507. The pattern differs from the Thinking-model results in Section 4: experience-guided iteration is not consistently beneficial on this non-Thinking backbone. At the final iteration, CTO-base slightly exceeds Standard Sampling on HMMT24, but underperforms Standard Sampling on HMMT25 and HLE-Math-text. RSE also fails to improve over Standard Sampling at the final iteration on all three benchmarks.

These results suggest that experience reuse is sensitive to the structure of the underlying rollouts. CTO relies on extracted positive and negative experience being aligned with the actual reasoning process. When the base model produces less stable long-horizon traces, the extracted pitfalls may become noisier or less causally tied to the errors that should be suppressed. This is especially consequential for CTO because negative experience is used as a decoding-time suppression signal rather than only as textual advice.

We therefore treat the Instruct-model results as an operating-condition check rather than as the main target setting for CTO. Applying CTO to non-Thinking instruction-tuned models may require stronger trajectory filtering, higher-precision experience annotation, or weaker suppression coefficients.

C.2 Experience Quality Audit Protocol

This appendix details the audit protocol for the experience-quality diagnostics in Table 5. The audit is conducted on HMMT24 using the post-deduplication pre-Iter2 bank $B^{(2)}$, formed after Iter1, produced by Qwen3-4B-Thinking-2507 under the same outer search protocol as the main experiments. For each extraction variant, we audit the negative items stored as critical pitfalls in the per-problem aggregate records. These items are proposition-level failure-mode statements rather than full trajectory summaries.

Audited variants. We audit the three extraction variants reported in Table 1: LLM-Judge, Direct CF, and Pseudo-CF. All variants use the same rollout pool, iteration schedule, intra-problem source restriction, and deduplication procedure as the corresponding main experiment. The audit is performed on the deduplicated pre-Iter2 bank $B^{(2)}$, after Iter1 experience has been merged and before Iter2 decoding, so it evaluates the actual negative items available to CTO during the last guided round.

Sampling. For each problem and each extraction variant, we uniformly sample up to five critical pitfalls without replacement from the deduplicated pre-Iter2 bank $B^{(2)}$. If a problem contains fewer than five pitfalls, all available pitfalls are used. The same sampled pitfalls are used to compute P_-, R_{fs}, and R_{contr}, except that pairwise metrics are computed over the corresponding item pairs induced by the sampled set. This keeps the three reported diagnostics comparable within each extraction variant.

Audit judge. All independent audit judgments use Kimi-K2.5 via the Moonshot OpenAI-compatible API with temperature fixed to 1.0. The judge receives only the information required by the corresponding audit: the problem statement, the audited pitfall, and, when needed, comparison materials such as correct continuations or another proposition. The judge is not given reference solutions or gold final answers. Each call is constrained to return a JSON object containing a discrete label and a short rationale. Malformed outputs are discarded and resampled before aggregation.

Negative precision. For P_-, the judge determines whether a sampled pitfall is a valid failure mode under the problem context. The prompt asks whether the pitfall identifies a genuine risky step, invalid inference, misleading assumption, or ineffective strategy, rather than an incidental wording difference or an overly generic warning. The output label is either `valid` or `invalid`. We compute

$$P_- = \frac{n_{\text{valid}}}{n_-},$$

where n_{valid} is the number of sampled pitfalls judged valid and n_- is the number of sampled pitfalls.

False suppression. For R_{fs}, we test whether a negative pitfall would suppress reasoning patterns that also appear in correct rollouts. For each problem, we collect the Iter0 trajectories judged correct by the gold-reference evaluator and extract up

Table 8: Supplementary results on Qwen3-4B-Instruct-2507. The table compares standard sampling, RSE, and CTO-base across three iterations to test whether experience-guided search transfers to a non-Thinking model.

Method	**HMMT24**			**HMMT25**			**HLE-Math-text**		
	It0	It1	It2	It0	It1	It2	It0	It1	It2
Qwen3-4B-Instruct-2507									
Standard Sampling	19.2	24.7	26.2	21.8	29.9	30.4	6.5	8.8	8.6
RSE	19.2	23.0	24.6	21.8	29.6	30.1	6.5	8.8	7.9
CTO-base	19.2	30.4	27.0	21.8	27.7	28.3	6.5	6.7	5.9

to three continuations whose local decision context is relevant to the audited pitfall. The judge receives the problem, the pitfall, and these correct continuations, then labels the pitfall as either `safe_to_suppress` or `false_suppression`. We compute

$$R_{\text{fs}} = \frac{n_{\text{fs}}}{n_{-}},$$

where n_{fs} is the number of sampled pitfalls judged to risk false suppression.

Within-problem contradiction. For R_{contr}, we first construct comparable pitfall pairs within the same problem. Two pitfalls are considered comparable if they refer to the same local decision context, such as the same setup step, variable definition, transformation, case split, or geometric configuration. The judge then determines whether the two pitfalls give mutually inconsistent warnings about that decision point. We compute

$$R_{\text{contr}} = \frac{n_{\text{contr}}}{n_{\text{pair}}},$$

where n_{contr} is the number of contradictory comparable pairs and n_{pair} is the total number of comparable pairs. Problems with no comparable pairs are excluded from this denominator.