Sparse Repair in Reasoning Traces: A Structural View of Test-Time Inference

Zhuorui Zhang[1], **Jianqi Yan**[2], **Shanshan Feng**[3], **Fan Li**[1*]
[1] The Hong Kong Polytechnic University, Hong Kong SAR, China;
[2] Sun Yat-sen University, Shenzhen, China; [3] Wuhan University, Wuhan, China
zhuorui.zhang00@gmail.com; yanjianqi.top@gmail.com;
victor_fengss@whu.edu.cn; fan-5.li@polyu.edu.hk

Abstract

Inference-time reasoning improves when models spend more computation through repeated sampling, search, or verification. Existing scaling strategies usually allocate this computation across prompts, candidate answers, or whole trajectories. We study a different structure: how the repair value of additional inference is distributed within a single generated reasoning trace. Across mathematical, scientific, coding, and knowledge-intensive reasoning tasks, recoverable utility is sharply concentrated: oracle replay over coarse trace windows shows that the top-three windows capture 83–90% of replay-recoverable utility. This concentration is not fully explained by position, length, or local uncertainty alone. Building on this finding, we introduce **LEVER**, a sparse local-intervention method for test-time reasoning that separates counterfactual discovery from deployable compute allocation. It uses local replay to estimate window-level recoverable utility, then distills this offline signal into a lightweight selector. At test time, the selector uses only online-safe trace features to decide which traces to activate and which windows receive additional rollouts. On the primary model, LEVER uses $1.31\times$ as many online completion tokens as base decoding on average. Across three models and four task families, it improves performance by an average of 4.2 percentage points over base decoding and achieves the best result on 11 of 12 model–task pairs at the primary operating points, while using less online compute than full-trajectory sampling baselines. These results suggest that effective test-time reasoning depends not only on how much computation is spent, but on whether it reaches the local commitments where repair value is highest.

1 Introduction

Giving language models more computation at inference time has become one of the most effective ways to improve reasoning. Recent work has shown gains from verifier-guided search, candidate selection, repeated sampling, and adaptive reasoning [7, 9, 24, 32]. In parallel, a growing body of work studies how test-time computation should be used efficiently, rather than simply increased [1, 10, 28]. Despite this progress, most methods still allocate additional inference across prompts, candidate solutions, or full trajectories. Far less is known about how its value is distributed within a single generated reasoning trace.

We therefore ask a more targeted question: within a single generated reasoning trace, where does additional inference yield the greatest repair value? Such a trace is not a uniform sequence of equally consequential steps. Some local decisions introduce commitments that shape later reasoning, while other segments are elaborations, restatements, or low-impact continuations. If this structure exists,

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

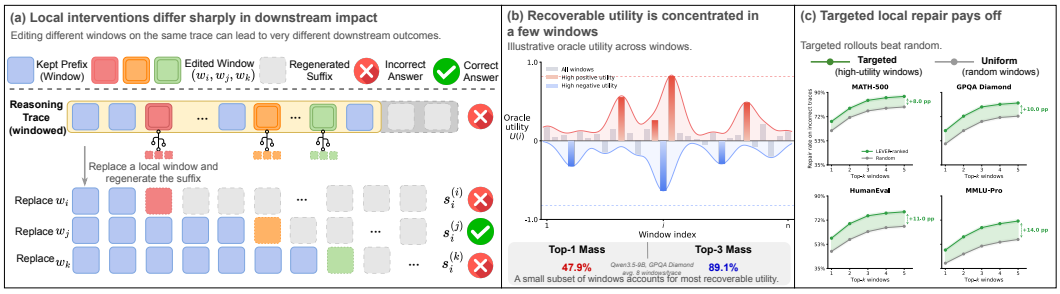

Figure 1: Sparse repair structure in reasoning traces. (a) Local interventions within the same trace can induce sharply different downstream outcomes. (b) Recoverable utility is concentrated in a small subset of windows. (c) Targeting high-utility windows yields higher payoff than unguided local intervention under the same rollout budget.

then spending extra inference uniformly along a trace is inefficient. The useful target is not the entire trajectory, but the small set of regions where intervention can change downstream solvability.

Figure 1 previews the central argument. Different local interventions within the same trace can lead to very different outcomes, and the recoverable value of such interventions is concentrated in a small number of windows. This suggests a different view of test-time reasoning: its value depends not only on how much additional inference is spent, but on whether it reaches the trace regions where repair is most likely.

We formalize these regions as *high-utility reasoning windows*: coarse local segments whose replacement exposes disproportionate recoverable utility after the suffix is regenerated. Concretely, given the prefix before a selected window, we sample an alternative local branch and let the model generate a fresh continuation from the modified prefix, so downstream reasoning can adapt to the changed local commitment. The resulting change in final solvability defines a repair-oriented utility for the selected window. Prior work has studied reasoning traces through step-level supervision and evaluation [17, 34], or through interventions that test whether explanations are faithful [19]. Our focus is different: we ask where local repair is possible within a trace, and whether this structure can guide where extra inference should be spent.

This operational view reveals a consistent pattern: with roughly eight coarse windows per trace, recoverable utility is concentrated in only a few regions. Across four benchmark families, the three highest-ranked windows under oracle replay capture over 80% of replay-recoverable utility, while many other windows have little effect under the same local operator. This concentration cannot be explained by position, length, or local uncertainty alone. Moreover, the most valuable intervention point often appears several windows before the final answer region, suggesting that many reasoning failures originate from earlier commitments whose effects surface only later. In other words, the recoverable value of extra inference is both sparse and temporally delayed.

We then turn this structural observation into **LEVER**, a deployable sparse-intervention method for test-time reasoning. Counterfactual replay estimates window-level recoverable utility and identifies high-utility regions of a trace; the resulting offline signal is distilled into a lightweight selector for deployment. At test time, the selector activates selected traces and ranks candidate windows using available trace features, allowing additional rollouts to focus on selected local regions rather than full-trajectory resampling.

Our contributions can be summarized as follows:

- We reveal a sparse repair structure in reasoning traces: replay-recoverable utility is concentrated in a small number of coarse local windows rather than spread uniformly across the trajectory.
- We show that high-utility windows go beyond simple cues such as position, length, or uncertainty, yet are sufficiently rankable from test-time trace features to support online compute allocation.
- Building on this phenomenon, we introduce LEVER, a sparse local-intervention method that distills replay-derived utilities into a lightweight online selector and improves reasoning by allocating additional rollouts to targeted local repair rather than full-trajectory resampling.

2 Related Work

Inference-time scaling and adaptive compute. A broad line of work improves reasoning through repeated sampling, verifier-guided reranking, explicit search, and adaptive allocation of inference-time compute [7, 9, 16, 24, 29, 32]. Recent work further explores reasoning efficiency and more flexible, compute-aware test-time scaling strategies [1, 10, 20, 28, 31]. Most of these methods allocate additional compute across prompts, candidate solutions, or full reasoning trajectories, with less attention to fine-grained intervention within a trace. We instead ask whether the value of additional computation varies substantially across local regions of a single generated reasoning trace.

Process supervision and evaluation. Process supervision, process reward models, and step-level evaluation study the structure of intermediate reasoning [14, 17, 22, 23, 30, 33, 34]. Related work also explores execution-style supervision and verifier-guided reasoning control [3, 5, 13]. Our focus is different: rather than treating each step as a supervision atom, we ask whether coarse local windows differ in downstream repair value under counterfactual replay.

Counterfactual intervention and reasoning faithfulness. Recent work has examined whether generated rationales or chain-of-thought traces faithfully reflect the model's effective decision process, using intervention-based evaluation and faithfulness analyses [6, 8, 11, 19, 25]. Work such as [2] uses sentence-level replacement and continuation to analyze influential reasoning units within a trace. Learned self-correction has also been explored as a complementary approach to improving reasoning [15]. Our work also uses local counterfactual replay, but for a different purpose. Rather than attributing influence or evaluating faithfulness, we use replay-derived recoverable utility to identify where limited inference-time computation is most likely to improve the final answer.

3 Method

Figure 2 summarizes LEVER. Offline, local counterfactual replay estimates window-level recoverable utility. Online, a factorized selector decides whether to activate a trace and ranks candidate windows for local rollout. The selector is lightweight by design and serves as a deployable approximation of the replay-revealed structure.

3.1 Problem Setup

Given an input problem x and a frozen language model, the test-time goal is to return a single final answer under a limited additional inference budget. We first decode a base trace τ and answer $\hat{y}$. LEVER then decides whether to spend additional rollouts on selected parts of τ and returns either the original answer or a locally intervened candidate. Offline discovery uses ground-truth answers to estimate replay utilities and train the selector; deployment uses only x, τ, and online-safe trace features.

Let the base trace be $\tau = (z_1, \ldots, z_T)$. We partition it into M contiguous windows, denoted by $\tau = (w_1, \ldots, w_M)$. Each window is a coarse local segment of the reasoning process. Unlike a fixed sentence, line, or annotated reasoning step, a window is defined by a split-then-merge construction: the trace is first split into task-appropriate visible units, such as step-like units for mathematical and scientific reasoning or line-level units for code, and adjacent units are then merged into coarse but editable windows. This keeps interventions aligned with logical units without reducing them to token spans or whole-trace revision. Details of the supported segmentation strategies and window-construction policy are provided in Appendix B.1.

For a selected window w_i, write the trace as $\tau = \tau_{<i} \oplus w_i \oplus \tau_{>i}$. A local intervention replaces w_i with an alternative branch $\tilde{w}_i$ sampled conditioned on the task input x, the original prefix $\tau_{<i}$, and the selected window w_i, and then decodes a fresh suffix:

$$\mathcal{I}_i(\tau; \tilde{w}_i) = \tau_{<i} \oplus \tilde{w}_i \oplus \widetilde{\tau}_{>i}.$$

The suffix $\widetilde{\tau}_{>i}$ is regenerated rather than copied, so that the downstream reasoning can adapt to the modified local branch. We denote the final answer from the intervened trajectory by $\hat{y}_i(\tilde{w}_i)$. A window has high repair utility if replacing it and allowing the continuation to re-evolve yields a large

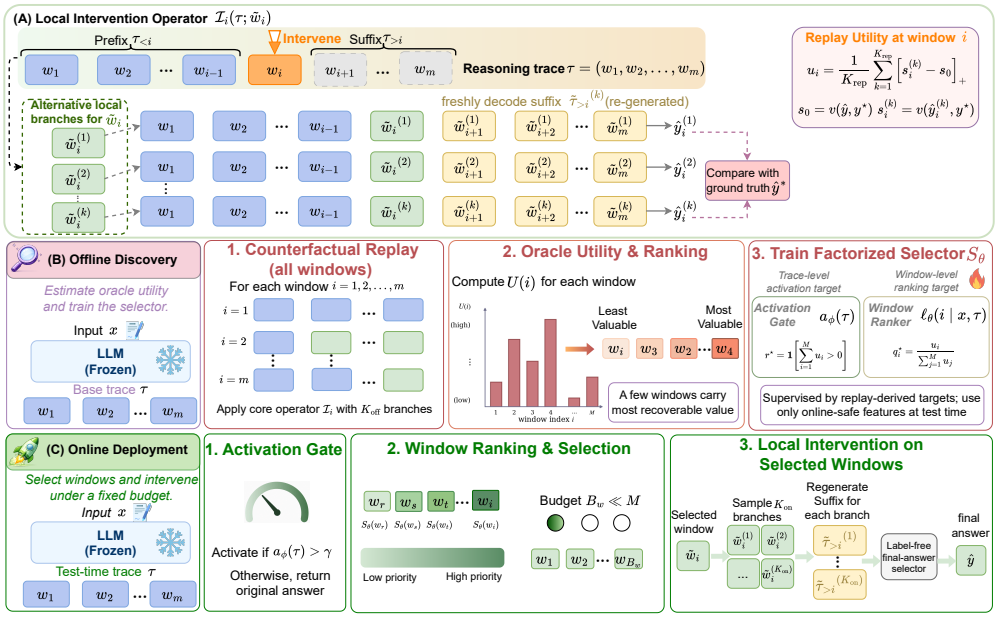

Figure 2: Overview of LEVER. (A) The local intervention operator replaces a selected window with an alternative branch and regenerates the suffix, producing replay utility. (B) Offline discovery estimates window-level utilities and converts them into activation and ranking targets for selector training. (C) Online deployment uses the trained selector to activate traces, rank windows, and allocate rollout budget to the top-ranked windows.

positive improvement in the final outcome. This makes utility an operational measure of repair under the local intervention operator.

3.2 Recoverable Utility

For offline replay, each window w_i is evaluated with K_{rep} alternative local branches $\{\tilde{w}_i^{(k)}\}_{k=1}^{K_{\text{rep}}}$ sampled conditioned on the task input x, the original prefix $\tau_{<i}$, and the original window w_i. Each branch replaces w_i, after which the suffix is regenerated to produce a counterfactual final answer $\hat{y}_i^{(k)}$. Let $s_0 = v(\hat{y}, y^\star)$ denote the outcome score of the original trace, and let $s_i^{(k)} = v(\hat{y}_i^{(k)}, y^\star)$ denote the outcome score of the k-th counterfactual rollout. In our evaluation, v is the official task metric: answer correctness for MATH-500, GPQA Diamond, and MMLU-Pro, and pass/fail execution for HumanEval.

We define the recoverable utility of window i as the average positive improvement exposed by local replay:

$$u_i = \frac{1}{K_{\text{rep}}} \sum_{k=1}^{K_{\text{rep}}} \left[s_i^{(k)} - s_0 \right]_+ .$$

Thus, u_i measures replay-exposed positive improvement at window i. A trace is recoverable under the replay budget if $\sum_i u_i > 0$. This utility is repair-oriented: high-utility windows are not merely influential, but useful targets for additional inference.

To measure whether recoverable utility is concentrated, we compute Mass@k on replay-recoverable traces, i.e., traces with $\sum_i u_i > 0$. For each such trace, we sort windows by u_i in descending order and define

$$m_k(\tau) = \frac{\sum_{j=1}^{\min(k,M)} u_{(j)}}{\sum_{j=1}^{M} u_j},$$

where $u_{(j)}$ is the j-th largest utility within the trace. We report Mass@k as the macro-average of $m_k(\tau)$ over replay-recoverable traces. It therefore measures the typical fraction of replay-recoverable

utility captured by the top-k oracle windows. This ranking serves as a hindsight oracle based on replay outcomes and is not available at deployment; Oracle@3 denotes the three highest-utility windows. Large values at small k indicate that sparse local intervention may be sufficient.

3.3 Listwise Window Ranking

Deployment does not require calibrated absolute utility values across traces. Window ranking is a conditional decision: given a trace to intervene on, the question is which windows should receive the limited local rollout budget. We therefore train a trace-conditioned window ranker from the replay utilities.

For traces with nonzero recoverable utility, the listwise target is

$$q_i^\star = \frac{u_i}{\sum_{j=1}^{M} u_j}.$$

Traces with $\sum_j u_j = 0$ do not define a meaningful recoverable-utility ordering and are excluded from the listwise ranking loss.

The default gate and window ranker use a compact set of online-safe features computed from the base trace. Structural features describe each window's position and extent, while lightweight format cues capture patterns such as equations and code (Appendix C.2). The ranker assigns each window an unnormalized priority logit $\ell_\theta(i \mid \tau)$ based on these features. The induced distribution over windows is

$$p_\theta(i \mid \tau) = \frac{\exp(\ell_\theta(i \mid \tau)/T)}{\sum_{j=1}^{M} \exp(\ell_\theta(j \mid \tau)/T)},$$

where T is a temperature. We train the ranker with the listwise cross-entropy objective

$$\mathcal{L}_{\text{rank}} = -\sum_{\tau \in \mathcal{D}_{\text{rec}}} \sum_{i=1}^{M} q_i^\star \log p_\theta(i \mid \tau),$$

where $\mathcal{D}_{\text{rec}} = \{\tau : \sum_i u_i > 0\}$ denotes replay-recoverable traces.

The ranker is set-wise rather than pointwise: each window is scored relative to shared trace context. Each window feature vector ϕ_i is encoded by a shared window encoder, $h_i = g_\theta(\phi_i)$, and scored relative to pooled trace context:

$$\ell_\theta(i \mid \tau) = f_\theta\big([h_i, \bar{h}, h_{\max}, h_i - \bar{h}]\big),$$

where $\bar{h} = M^{-1} \sum_{j=1}^{M} h_j$ and $h_{\max}$ is the elementwise max-pooled window representation. This set-wise design lets the model compare windows within the same trace without requiring calibrated utility scores across different traces.

3.4 Budgeted Online Intervention

Ranking alone does not decide whether to spend extra computation. Some traces already yield satisfactory answers, and some errors are not locally repairable under the current proposal budget. We therefore factor online allocation into two decisions: whether to activate a trace, and where to intervene if activated.

The activation gate is implemented as a window-level classifier whose outputs are aggregated into a trace-level activation score. For supervision, a window is labeled positive if replay indicates positive recoverable utility and its criticality (a composite measure of intervention-induced changes in correctness, answers, and continuations) exceeds a fixed threshold. The gate is trained only from online-safe window features, while recoverability and criticality are used only to construct offline labels. At test time, it produces a score $c_\phi(i \mid \tau)$ for each window, and the trace-level activation score is obtained by max pooling:

$$a_\phi(\tau) = \max_{i \in \{1,\dots,M\}} c_\phi(i \mid \tau).$$

The threshold γ is selected on the validation split to meet a target realized compute budget. If $a_\phi(\tau) \le \gamma$, LEVER returns the original answer $\hat{y}$. Otherwise, the trace is activated and the window ranker selects the top-B_w windows according to its priority logits:

$$S_{B_w} = \operatorname{TopK}_{i \in \{1,\dots,M\}} \ell_\theta(i \mid \tau).$$

Table 1: Main results across base models and task families at each method's natural primary operating point. Different inference-time methods spend computation in different units: full-trajectory baselines obtain their gains by drawing additional complete samples, while LEVER spends extra computation through sparse local rollouts. Cost is the mean realized completion-token ratio over the reported tasks within each model block. Self-Consistency is not reported on HumanEval because majority voting over free-form code is not well-defined. Scores are reported as percentages: accuracy for MATH-500, GPQA Diamond, and MMLU-Pro, and Pass@1 for HumanEval.

	MATH-500	GPQA Diamond	HumanEval	MMLU-Pro	Cost
Qwen3.5-9B	88.6	79.8	77.4	76.5	1.00×
+ Self-Consistency	90.0	79.3	–	77.7	3.04×
+ Best-of-N	<u>90.4</u>	<u>81.8</u>	78.7	78.2	3.08×
+ Global Revision	89.0	77.3	76.2	77.2	1.48×
+ LLM-Distilled Selector	90.0	80.8	<u>79.3</u>	<u>78.7</u>	1.38×
+ LEVER (ours)	**92.0**	**84.8**	**82.3**	**80.6**	1.31×
Phi-4-Reasoning (14B)	81.8	62.1	79.9	70.7	1.00×
+ Self-Consistency	83.2	63.1	–	71.7	3.07×
+ Best-of-N	84.0	64.6	<u>81.7</u>	72.2	3.10×
+ Global Revision	81.4	61.1	79.3	71.3	1.51×
+ LLM-Distilled Selector	<u>84.6</u>	**66.2**	80.5	<u>72.8</u>	1.40×
+ LEVER (ours)	**86.0**	<u>65.7</u>	**84.1**	**74.8**	1.34×
Mistral-Small-3.1-24B	70.2	40.9	82.3	64.3	1.00×
+ Self-Consistency	70.0	41.4	–	65.3	3.03×
+ Best-of-N	<u>72.2</u>	<u>42.9</u>	<u>84.1</u>	65.9	3.09×
+ Global Revision	69.4	39.9	81.7	65.0	1.47×
+ LLM-Distilled Selector	71.2	42.4	83.5	<u>66.7</u>	1.37×
+ LEVER (ours)	**75.8**	**45.5**	**85.4**	**68.1**	1.28×

Local branch replacement and suffix regeneration are then performed only on windows in S_{B_w}. For each selected window, online deployment samples K_{on} local branches and regenerates one suffix per branch, producing at most $B_w K_{\text{on}}$ intervened candidates. The final output is selected from the original answer and the valid intervened candidates using the shared likelihood-based candidate selection rule.

For methods that select among generated candidates, we use the same non-oracle selection rule. For MATH-500, GPQA Diamond, and MMLU-Pro, we extract normalized final answers and rank valid candidates by the length-normalized generation log-likelihood of the final-answer tokens following the final-answer marker. For HumanEval, we first apply syntax and required-function validity checks, then rank valid programs by the length-normalized generation log-likelihood of the generated code tokens. Self-Consistency is reported only on closed-answer tasks, where it uses normalized answer majority. Replay-derived utilities and ground-truth outcomes are used only for offline discovery and selector supervision, not for test-time candidate selection.

This factorization separates *whether* to spend extra inference from *where* to spend it. For localization metrics such as Mass@k and NDCG@3, we evaluate only the window ranker on replay-recoverable traces, without applying the activation gate. For end-to-end task performance, we evaluate the full activation-then-ranking deployment policy at validation-selected operating points and report realized completion-token cost. Additional details on the activation gate, ranker variants, feature boundary, training hyperparameters, and compute accounting are given in Appendix C.

4 Experiments

4.1 Experimental Setup

Benchmarks and Models. We evaluate on four reasoning-intensive task families: MATH-500 [17], GPQA Diamond [21], HumanEval [4], and MMLU-Pro [27], covering mathematical reasoning, scientific multiple-choice reasoning, code generation, and broad knowledge-intensive reasoning. Results are reported on three open-weight models: Qwen3.5-9B, Phi-4-Reasoning (14B), and Mistral-

Small-3.1-24B-Instruct-2503, covering different model scales and reasoning profiles. All methods produce a single final answer per example. We report realized completion-token cost and compare performance as a function of measured inference cost, rather than forcing all methods into a single artificial budget point. Mechanistic analyses use Qwen3.5-9B as the primary study model and evaluate the same task families, providing a unified protocol for concentration, rankability, and delay analyses.

Baselines and Evaluation Protocol. The base row in each model block of Table 1 corresponds to standard single-path decoding. We compare LEVER with four inference-time baselines and report each method's realized completion-token cost. **Self-Consistency** [26] samples multiple full solutions and aggregates normalized final answers on closed-answer tasks; we omit it on HumanEval because majority voting over free-form code is not well-defined. **Best-of-N** [12] samples multiple full-solution candidates and selects one with the shared non-oracle candidate selector. **Global Revision** [18] revises the full trace and selects between the original and revised candidates using the same selector. **LLM-Distilled Selector** uses the same local intervention pipeline as LEVER, but replaces replay-derived supervision with offline LLM-generated window ratings.

We report accuracy on MATH-500, GPQA Diamond, and MMLU-Pro, and Pass@1 on HumanEval. The primary comparisons follow the benchmark-specific evaluation protocols. Realized cost is measured relative to base decoding using generated completion tokens. The default label-free candidate selector uses token log-probabilities recorded during generation, so it does not introduce additional rollout or oracle-evaluation cost. The appendix also reports additional MMLU-Pro controls and evaluates a prompt-based alternative for candidate selection (Appendix E; Appendix E.4). The compared methods spend extra inference at different granularities: full-trajectory baselines add complete solution trajectories, while LEVER adds local branches and suffix rollouts only on activated traces. Table 1 reports each method at its primary operating point, and Figure 3 shows the measured compute–performance frontier. LEVER targets the low-to-medium cost regime, around 1.3–$1.4\times$ base decoding.

Implementation Details. The online selector is trained from replay-derived supervision but uses only pre-intervention features at test time. It consists of a trace-level activation gate and a set-wise window ranker; the oracle selector is reported only as an analysis upper bound. For selector training, MATH-500, GPQA Diamond, and HumanEval use five-fold example-level splits, whereas MMLU-Pro uses a fixed 70/15/15 split. LEVER uses $K_{\text{rep}} = 3$ replay branches per window for offline supervision and deploys with $B_w = 3$, $K_{\text{on}} = 1$, and a validation-selected activation threshold. This yields a 29.1% trace activation rate and a realized completion-token cost of $1.31\times$ on the primary model; these online ratios exclude one-time replay and selector-training costs, which are reported separately in Appendix E.2. Appendix A reports prompts, extraction rules, decoding settings, and benchmark protocol; Appendix C reports selector hyperparameters and compute accounting.

4.2 Main Results

Overall Performance. Table 1 shows that targeting local windows is a competitive use of inference-time compute. Across the three base models, LEVER gives the strongest average performance while using substantially less compute than full-trajectory sampling. The strongest non-LEVER baselines are Best-of-N and LLM-Distilled Selector, but their gains are smaller or require more generated tokens. Additional comparisons distinguish compute allocation from supervision: at matched online-token budgets, LEVER averages 84.9 versus 81.9 for the LLM-distilled selector; a replay-supervised global-revision control using the same replay data reaches 82.6 at $1.32\times$, versus LEVER's 84.9 at $1.31\times$. These controls use MMLU-Pro-500, not the full MMLU-Pro evaluation (Appendix E).

Table 2: Mechanistic statistics on the primary model. Mass@k is a normalized utility fraction; repair rates are measured on incorrect base traces and reported in percent.

Task	Mass@1	Mass@3	Oracle@3 (%)	LEVER@3 (%)	Delay
MATH	0.44	0.85	86.8	81.6	3.5
GPQA Diamond	0.48	0.89	83.6	76.8	2.9
HumanEval	0.47	0.90	79.4	73.5	2.9
MMLU-Pro	0.26	0.83	74.2	66.4	2.3

Compute–Performance Trade-off. Figure 3 plots performance against realized completion-token cost. Each point in the trade-off curve is placed at its measured token ratio, with performance normalized by the corresponding base score. Because Self-Consistency is omitted on HumanEval, its aggregate point in Figure 3(a) averages only closed-answer tasks; panel (c) uses the strongest

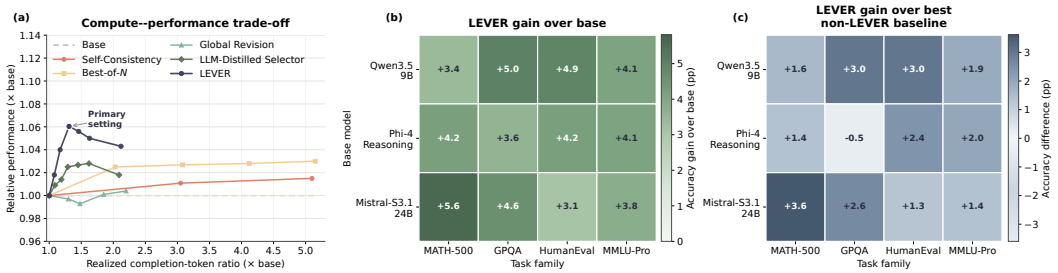

Figure 3: Compute–performance comparison across model–task pairs. (a) Measured trade-off using realized completion-token ratios and base-normalized performance. (b) Gain of LEVER over base decoding. (c) Gain of LEVER over the strongest non-LEVER baseline under primary configurations.

Table 3: Bridge from window-ranking quality to controlled local repair on the primary model. NDCG@3 is computed against oracle recoverable-utility rankings. Repair@3 is the percentage of initially incorrect examples corrected by selecting among the base answer and the Top-3 local rollouts. Bold indicates the best deployable ranker; Oracle is an analysis-only upper bound.

Window ranker	MATH-500		GPQA Diamond		HumanEval		MMLU-Pro	
	NDCG@3	Repair@3	NDCG@3	Repair@3	NDCG@3	Repair@3	NDCG@3	Repair@3
Random	0.38	50.4	0.31	46.7	0.34	44.8	0.30	40.9
LLM-Distilled	0.52	72.8	0.48	67.4	0.50	65.9	0.45	60.3
LEVER	**0.58**	**81.6**	**0.54**	**76.8**	**0.56**	**73.5**	**0.52**	**66.4**
Oracle	1.00	86.8	1.00	83.6	1.00	79.4	1.00	74.2

available non-LEVER baseline for each model–task pair. The realized-token accounting is detailed in Appendix C.5. Whole-trajectory baselines gain from additional complete samples, and their cost naturally increases in coarse units because each candidate requires a full solution trajectory. In contrast, LEVER changes the compute unit: it spends extra computation only on activated traces and selected local rollouts, yielding a stronger low-to-medium cost trade-off. The accompanying heatmaps show that this advantage is broadly distributed across tasks and models, and remains visible against the strongest non-LEVER baseline. Thus, extra inference helps most when it is spent in the right parts of the trace.

4.3 Mechanistic Validation

Concentration of Recoverable Utility. Table 2 summarizes concentration, repairability, and delay on the primary model. Mass@k measures the fraction of replay-recoverable utility captured by the top-k oracle-ranked windows, while Oracle@3 and LEVER@3 report repair rates on originally incorrect base traces. Even the top oracle-ranked window captures a substantial share of the available utility, with Mass@1 ranging from 0.26 to 0.48 across tasks. The top-3 windows capture most of the recoverable utility on MATH-500 (0.85), GPQA Diamond (0.89), HumanEval (0.90), and MMLU-Pro (0.83). This consistent concentration suggests that the recoverable benefit of additional inference is carried by a small number of high-impact commitment windows. Additional analyses show similar concentration under pre-window suffix resampling (four-task Mass@3: 0.82) and on Phi-4-Reasoning (Mass@3: 0.84; Appendix E.3).

Recoverability under Local Intervention. Concentration alone would not justify sparse intervention if incorrect traces were not locally repairable. Table 2 evaluates repair on incorrect base traces. Oracle@3 measures repair when the three replay-best windows are selected, while LEVER@3 uses the top-three windows predicted by the deployable ranker. The gap reflects remaining localization error rather than a budget difference. The mean delay of two to four windows further suggests that many failures originate from earlier commitments whose effects materialize later, rather than from terminal slips.

Figure 4 provides a complementary view. Figure 4(a) shows that recoverable utility rises sharply in the first few oracle-ranked windows across all four task families, well above the uniform-random

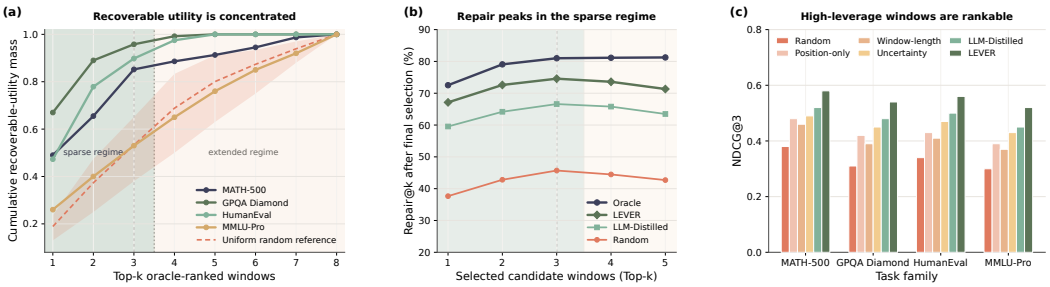

Figure 4: Mechanistic analysis of sparse local intervention. (a) Oracle-ordered recoverable-utility mass shows concentration in top-ranked windows relative to a uniform-random reference. (b) Repair rate on originally incorrect traces under Top-k window selection. LEVER recovers much of the oracle opportunity with few windows. (c) NDCG@3 on originally incorrect, oracle-recoverable traces. Position, length, uncertainty, and LLM-distilled ratings are useful but weaker than LEVER.

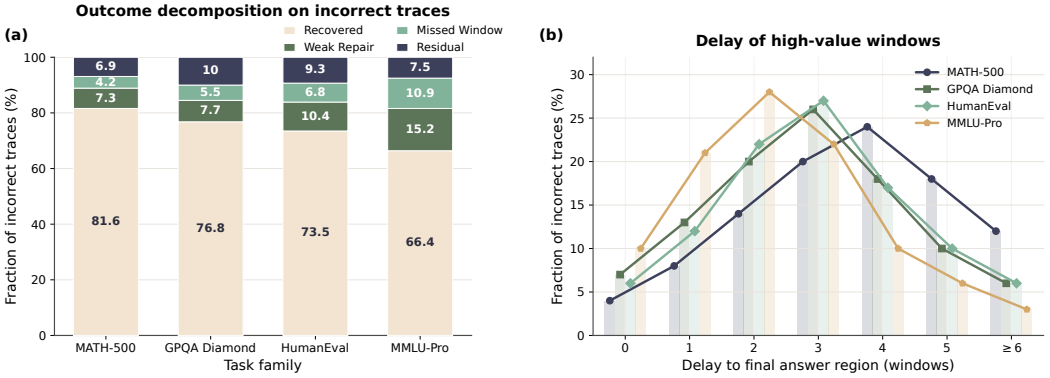

Figure 5: Residual gap and delay analysis on the primary model. (a) Decomposition of originally incorrect traces under the Top-3 local intervention setting. (b) Distribution of the distance between the highest-utility oracle window and the final answer region.

reference; the dashed curve reports the mean reference, and the translucent band shows its range across task families. Figure 4(b) shows that much of this repair opportunity can be captured with a small candidate set, with LEVER consistently closer to the oracle than random or LLM-distilled ranking under the same Top-k budget. Figure 4(c) shows that high-utility windows are not reducible to position, length, or uncertainty: these proxy rankers remain below the LEVER ranker. Together, these results show that sparse intervention works because recoverable utility is concentrated, locally repairable, and sufficiently rankable for online deployment.

From Utility Prediction to Downstream Gain. Table 3 connects ranking quality to controlled local repair. Across all tasks, better alignment with oracle recoverable-utility rankings is accompanied by higher Repair@3 under the same intervention operator and budget. LEVER consistently improves over random and LLM-distilled ranking, while Oracle remains an analysis-only upper bound. This provides the empirical bridge between the replay-discovered utility structure and downstream gains. Additional ranker-variant checks are reported in Appendix D.

4.4 Residual Gap and Failure Structure

Figure 5 separates the remaining gap after Top-3 intervention on originally incorrect traces. Beyond the cases recovered by LEVER, the remaining errors point to three bottlenecks: weak repair, where selected regions do not yield corrective continuations; missed localization, where recoverable regions are not selected; and residual cases that likely require broader or multi-window changes. Figure 5(b) further shows that the highest-utility oracle window often appears several windows before the final answer region. This pattern suggests that many recoverable errors arise from earlier commitments whose effects unfold downstream, rather than from terminal answer slips alone.

Appendix F provides representative cases of successful repair, weak repair, missed localization, and residual failure. Appendix D reports sensitivity analyses for activation thresholds, local rollout budgets, window granularity, and ranker variants. Suffix-resampling and transfer analyses are reported in Appendix E.3.

5 Conclusion

We presented a structural view of test-time reasoning in which recoverable utility is concentrated in a small number of local windows. Using local prefix counterfactual intervention with fresh continuation, we studied where this utility appears and whether it can guide budgeted deployment. Across benchmarks, reasoning traces are not uniformly valuable targets for additional inference: a small subset of windows accounts for a disproportionate share of replay-recoverable utility. This structure is not reducible to position, length, or uncertainty alone, but it is rankable enough for a lightweight online selector to allocate additional computation more effectively in practice. Although constructing the selector requires one-time offline replay, this cost can be amortized across subsequent queries, and the learned selector can be transferred without repeating replay. Ultimately, effective test-time reasoning depends not only on how much additional inference is used, but also on where that computation is spent.

Acknowledgments and Disclosure of Funding

Funding. No third-party funding or support was received for this work. **Competing Interests.** The authors declare no competing interests.

References

[1] Daman Arora and Andrea Zanette. Training language models to reason efficiently. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=AiZxn84Wdo.

[2] Paul C. Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. Thought anchors: Which llm reasoning steps matter?, 2025. URL https://arxiv.org/abs/2506.19143.

[3] Kaiyan Chang, Yonghao Shi, Chenglong Wang, Hang Zhou, Chi Hu, Xiaoqian Liu, Yingfeng Luo, Yuan Ge, Tong Xiao, and JingBo Zhu. Step-level verifier-guided hybrid test-time scaling for large language models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 18462–18477, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.931. URL https://aclanthology.org/2025.emnlp-main.931/.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[5] Nuo Chen, Zehua Li, Keqin Bao, Junyang Lin, and Dayiheng Liu. Chain of execution supervision promotes general reasoning in large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=QjnKsujXVG.

[6] Yanda Chen, Ruiqi Zhong, Narutatsu Ri, Chen Zhao, He He, Jacob Steinhardt, Zhou Yu, and Kathleen McKeown. Do models explain themselves? counterfactual simulatability of natural language explanations. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[7] Yinlam Chow, Guy Tennenholtz, Izzeddin Gur, Vincent Zhuang, Bo Dai, Aviral Kumar, Rishabh Agarwal, Sridhar Thiagarajan, Craig Boutilier, and Aleksandra Faust. Inference-aware fine-tuning for best-of-n sampling in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=77gQUdQhE7.

[8] Yu-Neng Chuang, Guanchu Wang, Chia-Yuan Chang, Ruixiang Tang, Shaochen Zhong, Fan Yang, Andrew Wen, Mengnan Du, Xuanting Cai, Vladimir Braverman, and Xia Hu. FaithLM: Towards faithful explanations for large language models. In Vera Demberg, Kentaro Inui, and Lluís Marquez, editors, *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3802–3824, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-380-7. doi: 10.18653/v1/2026.eacl-long.177. URL https://aclanthology.org/2026.eacl-long.177/.

[9] Shengyu Feng, Xiang Kong, Shuang Ma, Aonan Zhang, Dong Yin, Chong Wang, Ruoming Pang, and Yiming Yang. Step-by-step reasoning for math problems via twisted sequential monte carlo. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Ze4aPP0tIn.

[10] Jonas Geiping, Sean Michael McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=S3GhJooWIC.

[11] Yunseok Han, Yejoon Lee, and Jaeyoung Do. RFEval: Benchmarking reasoning faithfulness under counterfactual reasoning intervention in large reasoning models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=2Gc8aj0afg.

[12] Audrey Huang, Adam Block, Qinghua Liu, Nan Jiang, Akshay Krishnamurthy, and Dylan J Foster. Is best-of-n the best of them? coverage, scaling, and optimality in inference-time alignment. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=QnjfkhrbYK.

[13] Sian-Yao Huang, Li-Hsien Chang, Che-Yu Lin, and Cheng-Lin Yang. Beyond oracle: Verifier-supervision for instruction hierarchy in reasoning and instruction-tuned LLMs. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=IQ513IX1G5.

[14] Zeyu Jia, Alexander Rakhlin, and Tengyang Xie. Do we need to verify step by step? rethinking process supervision from a theoretical perspective. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=4BfaPHfhJ0.

[15] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pages 54523–54549, 2025.

[16] Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In *EMNLP (Findings)*, pages 15964–15978, 2025.

[17] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

[18] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

[19] Katie Matton, Robert Ness, John Guttag, and Emre Kiciman. Walk the talk? measuring the faithfulness of large language model explanations. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4ub9gpx9xw.

[20] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

[21] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

[22] Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=A6Y7AqlzLW.

[23] Shuaijie She, Junxiao Liu, Yifeng Liu, Jiajun Chen, Xin Huang, and Shujian Huang. R-PRM: Reasoning-driven process reward modeling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 13438–13451, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.679. URL https://aclanthology.org/2025.emnlp-main.679/.

[24] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

[25] Martin Tutek, Fateme Hashemi Chaleshtori, Ana Marasovic, and Yonatan Belinkov. Measuring chain of thought faithfulness by unlearning reasoning steps. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 9935–9960, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.504. URL https://aclanthology.org/2025.emnlp-main.504/.

[26] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

[27] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=y10DM6R2r3.

[28] Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. Towards thinking-optimal scaling of test-time compute for LLM reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=6ICFqmixlS.

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

[30] Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024.

[31] Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. Adaptthink: Reasoning models can learn when to think. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3716–3730, 2025.

[32] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Ccwp4tFEtE.

[33] Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 10495–10516, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.547. URL https://aclanthology.org/2025.findings-acl.547/.

[34] Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. ProcessBench: Identifying process errors in mathematical reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1009–1024, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.50. URL https://aclanthology.org/2025.acl-long.50/.

A Evaluation Protocol and Reproducibility Details

A.1 Benchmarks, Splits, and Metrics

Table 4 summarizes the benchmark protocol. All augmented methods ultimately return a single final answer per example. For candidate-generating methods, the final output is selected using the same label-free candidate-selection rule used by the corresponding baseline or deployment policy; ground-truth labels are used only for evaluation and for constructing offline replay supervision.

Table 4: Benchmark protocol. MATH-500, GPQA Diamond, and MMLU-Pro are evaluated by final-answer accuracy; HumanEval is evaluated by unit-test execution.

Benchmark	Task type	Metric	Dataset size	Evaluation detail
MATH-500	Mathematical reasoning	Accuracy	500	Final symbolic or numeric answer matching
GPQA Diamond	Scientific multiple choice	Accuracy	198	Choice-letter extraction and exact match
HumanEval	Code generation	Pass@1	164	Function-level unit-test execution
MMLU-Pro	Knowledge-intensive QA	Accuracy	12032	Choice-letter extraction and exact match

For the primary mechanistic analysis, we use Qwen3.5-9B and the same four task families. This keeps the concentration, rankability, and repair analyses under a unified evaluation protocol. The cross-model main results use the same evaluation scripts, answer extractors, and realized-token accounting. MMLU-Pro is split into training, validation, and test sets using a fixed 70/15/15 split. The primary results use the full test split. Additional matched-budget, equally supervised, transfer, and operator analyses use a fixed 500-example subset of the test split (MMLU-Pro-500), while the 602 replay examples are drawn from the training split.

A.2 Prompting and Answer Extraction

All methods use the same base task prompt for a given benchmark and model. The prompt asks the model to solve the problem with visible reasoning and provide an extractable final answer. The local branch prompt is intentionally different: conditioned on the task input, the prefix before the selected window, and the original window, it asks only for a replacement local reasoning window. The suffix is then regenerated separately, keeping each intervention local.

For MATH-500, we extract the final mathematical answer from the explicit final-answer field when present, falling back to the last non-empty line otherwise. Common formatting wrappers are removed before passing the prediction to the benchmark evaluator. For GPQA Diamond and MMLU-Pro, extraction follows a conservative order: explicit final-answer field, final JSON answer when specified by the run configuration, final standalone option letter, and finally a regular expression over the last few lines. If multiple conflicting choices remain after normalization, the example is marked unparsed. For HumanEval, generated code is parsed into the required function definition or function body and then executed against the benchmark tests.

A.3 Prompt Templates

We use a small set of shared prompt families rather than designing separate prompts for each baseline. Base decoding, self-consistency, and Best-of-N use the same base task prompt; they differ only in the number of sampled traces and the final aggregation or selection rule. Global revision first generates a base trace and then applies a full-trace revision prompt. The LLM-distilled selector uses an additional selector-scoring prompt to choose intervention windows, but after window selection it shares the same local intervention prompts as LEVER. Thus, the LLM-distilled selector and LEVER differ in the window-selection mechanism, not in the local intervention operator.

Base task prompt. The base prompt is instantiated by benchmark family. For mathematical and multiple-choice tasks, it requests step-by-step reasoning followed by a normalized final-answer line. For HumanEval-style code tasks, the final answer is a Python implementation.

Base task prompt

Used by: Base, Self-Consistency, Best-of-N, Global Revision, LLM-Distilled Selector, and LEVER.

Solve the following ⟨benchmark⟩ problem.
Think carefully step by step.
After the reasoning, provide the final answer in the required benchmark-specific format.

Problem:
⟨problem text⟩

Choices, if applicable:
⟨rendered choices⟩

Required final format:
FINAL_ANSWER: <answer> for math-style tasks.
FINAL ANSWER: <LETTER> for multiple-choice tasks.
Python fenced block for code-generation tasks.

Full-trace revision baseline. Global revision is the only baseline that asks the model to revise the entire generated trace.

Global revision prompt

Used by: Global Revision.

Review the following solution attempt.
If it contains an error, revise the reasoning and produce a corrected final answer.
Keep the final answer in the required benchmark-specific format.

Problem:
⟨problem text⟩

Original solution:

⟨base trace⟩

Revised solution:

LLM-distilled selector prompt. The LLM-distilled selector baseline uses a prompt-based teacher to assign window priorities from visible information only. It does not receive the ground-truth answer or replay outcomes.

Selector-scoring prompt

Used by: LLM-Distilled Selector.

Given a problem and a reasoning trace segmented into windows, rate which windows are most likely to benefit from additional local reasoning.
Use only the visible problem statement and generated trace.
Do not use the ground-truth answer.
Assign relative priority scores to the windows.

Problem:
⟨problem text⟩

Windows:
⟨window id⟩: ⟨window text⟩
...

Output:
⟨window id, priority score⟩

Local intervention prompt. The local intervention operator has two stages. The branch stage writes a replacement for the selected window only. The rollout stage continues from the modified prefix and produces the final answer. The same two-stage template is shared by the LLM-distilled selector baseline and LEVER.

Local intervention prompt

Used by: LLM-Distilled Selector and LEVER.

Stage A: local branch:
Given the problem, the prefix before the selected window, and the original window, write a corrective replacement for that window.
The replacement should reconsider the local assumption, equation, comparison, implementation decision, or reasoning choice.
It must remain coherent with the prefix.
Output only the replacement window, not the full solution.

Problem:
⟨problem text⟩

Prefix before selected window:
⟨trace prefix⟩

Original window:
⟨original window⟩

Stage B: suffix rollout:
Continue from the modified prefix and finish the solution.
Keep the final answer in the required benchmark-specific format.

Modified prefix:
⟨trace prefix + replacement window⟩

Continuation:

For multiple-choice tasks, the allowed letter set is rendered from the available choices for the benchmark. For HumanEval-style code tasks, the same local intervention template is instantiated over code windows: the branch stage outputs only a replacement code span, and the rollout stage completes the implementation from the modified partial program.

A.4 Generation Settings

We separate model-level decoding parameters from generation-mode completion-token caps. For each base model, we follow the model-recommended generation configuration when it is specified; parameters not specified by the recommendation are left unset in the decoding call. Within a fixed base model, the same sampling configuration is applied uniformly to all methods. Thus, differences among base decoding, self-consistency, Best-of-N, global revision, the LLM-distilled selector, and LEVER cannot be attributed to method-specific temperature, nucleus sampling, or repetition-control policies. Table 5 reports the model-level sampling settings, and Table 6 reports the completion-token caps used by each generation mode.

Local branch generation is capped more tightly because it is intended to produce only a replacement window, while suffix rollout is capped as a continuation from the modified prefix. For compute comparisons, we report realized completion-token ratios rather than nominal maximum-token limits, since different traces terminate at different lengths under the same cap. All final reported generation experiments were run on a single NVIDIA RTX PRO 6000 GPU with 96GB of VRAM. Model serving was implemented primarily with vLLM, with HuggingFace Transformers used only for auxiliary loading or evaluation utilities when needed.

Table 5: Model-level decoding settings. For each base model, we follow the model-recommended generation configuration when specified. Parameters not explicitly specified by the model recommendation are left unset, allowing the inference engine to use its default values. Within each base model, the same settings are shared by all evaluated methods.

Base model	Temperature	Top-p	Top-k	Pres. pen.	Rep. pen.
Qwen3.5-9B	1.0	0.95	20	1.5	1.0
Phi-4-Reasoning	0.8	0.95	50	–	1.0
Mistral-Small-3.1-24B-Instruct-2503	0.15	–	–	–	1.0

B Window Construction and Counterfactual Replay

This section specifies the text-level unit used by LEVER: a coarse local reasoning window. The goal is not to identify token-level causes, but to obtain stable, editable regions that correspond to local reasoning commitments. Given a base trace, we construct a small ordered set of windows, replace one window at a time, and regenerate the suffix from the modified prefix. All interventions are therefore performed on visible text only.

B.1 Window Construction

Window construction maps a generated trace into a small ordered set of contiguous spans. We support four segmentation strategies: `auto`, `step`, `line`, and `sentence`. The main experiments use `auto` for natural-language reasoning traces and `line` for code traces. The `auto` strategy first looks for explicit reasoning steps, such as numbered steps or step headings, and falls back to non-empty lines when such structure is absent. This makes the segmentation adaptive to the format of the trace while keeping the intervention unit visible and reproducible. For HumanEval, line-level segmentation is more appropriate because code lines already represent compact local implementation commitments.

After the initial split, adjacent units are grouped into coarse windows using a token-balanced rule. The grouping preserves the original order, avoids unnecessarily fragmenting coherent local reasoning, and targets a small number of windows per trace. Very long units may be refined by sentence boundaries when safe, and excessive tail windows are merged to respect the window cap. This design gives a consistent mesoscopic intervention unit while still supporting alternative granularities for robustness checks. The default values are summarized in Table 7, and the induced dataset-level statistics are reported in Table 8.

Table 8 reports the induced window statistics on the final replay traces. These statistics characterize the resulting granularity; all methods that require local windows use the same construction policy.

Table 6: Completion-token caps used for each generation mode. These values are API-level upper bounds rather than realized compute costs. The caps are used to prevent premature truncation, while all compute frontiers and operating points are computed from realized completion-token ratios.

Generation mode	Max completion tokens	Shared by
Base trace generation	4,096	Base, Self-Consistency, Best-of-N, Global Revision, LLM-Distilled Selector, LEVER
Full-solution candidate generation	4,096	Self-Consistency, Best-of-N
Local branch generation	512	LLM-Distilled Selector, LEVER
Suffix rollout	1,024	LLM-Distilled Selector, LEVER
Global revision	4,096	Global Revision
Selector-scoring generation	512	LLM-Distilled Selector

Table 7: Default parameters for window construction and replay.

Component	Parameters	Values	Note
Segmentation	$\mathcal{S}$	auto / step / line / sentence	available strategies
Window count	$(M_{\text{tar}}, M_{\max})$	$(8, 12)$	target / cap
Window budget	$(b_{\min}, b_{\max}, b_{\text{split}})$	$(80, 180, 60)$	soft grouping/refinement
Offline replay branching	K_{rep}	3	replay branches
Online deployment budget	(B_w, K_{on})	$(3, 1)$	selected windows / online branches
Generation caps	$(L_{\text{br}}, L_{\text{ro}})$	$(512, 1024)$	branch / rollout

B.2 Counterfactual Replay

Given windows $W = (w_1, \ldots, w_M)$, replay evaluates whether replacing a local window can improve the final answer. For each window w_i, we sample alternative branches conditioned on the task input x, the original prefix $\tau_{<i}$, and the original window w_i. Algorithm 1 summarizes the procedure.

The replay record is used offline for two purposes: mechanistic analysis and selector supervision. It is not available at deployment time. In particular, branch outcomes, regenerated continuations, counterfactual correctness, and gold-answer information are excluded from the online selector inputs.

B.3 Replay Artifact Fields

Table 9 distinguishes information available during online deployment from artifacts used only in offline replay. The online selector ranks windows using features computed from the base trace. Replay outcomes, utilities, and gold-answer-dependent signals are used only for offline supervision. New branches and continuations are generated online after window selection.

C Selector Training and Online Deployment

The deployable selector is factorized into an activation gate and a window ranker. The activation gate decides whether a trace should receive additional computation; the ranker decides where that computation should be spent within an activated trace. Both components are trained from replay-derived supervision, but are deployed using only pre-intervention information. The information boundary is defined in Appendix B.3: branch outcomes, regenerated continuations, counterfactual correctness, answer-change rates, continuation divergence, and gold-answer-dependent quantities are not used as test-time inputs.

C.1 Factorized Selector

The online selector is factorized into an activation gate and a window ranker. The activation gate makes a trace-level spending decision: if the activation score does not exceed the validation-selected threshold, LEVER returns the base answer without local rollout. The window ranker is only used for activated traces. It assigns a priority score to each window and allocates the local rollout budget to the highest-ranked windows.

The two components are trained from different replay-derived targets. The activation gate is trained at the window level. For each window, the binary activation label is positive when replay exposes positive recoverability for that window and its auxiliary replay criticality exceeds a fixed threshold. Only this label is replay-derived; the gate input consists only of online-visible window features. At deployment, the gate outputs a score for each window, and the trace-level activation score is obtained by max pooling over window scores. The ranker uses the normalized replay utility distribution over

Table 8: Window statistics under the default construction rule. Trace toks., Units, and Windows are averaged over final replay traces. Toks./win. is computed as total trace tokens divided by total constructed windows.

Task	Trace toks.	Units	Windows	Toks./win.
MATH-500	612.5	12.9	7.4	82.8
GPQA Diamond	946.2	13.7	7.8	121.3
HumanEval	224.6	19.8	6.7	33.5
MMLU-Pro	536.4	12.4	7.0	76.6

Algorithm 1 Counterfactual replay for one trace

Require: Problem x, gold answer y, base trace τ, windows $W = (w_1, \ldots, w_M)$, branch count K

Evaluate the base trace and obtain the original prediction $\hat{y}$.
for $i = 1, \ldots, M$ **do**
    Split τ around w_i into prefix p_i and suffix s_i.
    Sample K local alternatives $\{\tilde{w}_i^{(k)}\}_{k=1}^{K}$ conditioned on (x, p_i, w_i).
    for each sampled alternative $\tilde{w}_i^{(k)}$ **do**
        Form the modified prefix $p_i \circ \tilde{w}_i^{(k)}$ and generate a fresh continuation.
        Evaluate the intervened trace and store its prediction and replay diagnostics.
    end for
    Aggregate branch outcomes into the replay utility u_i.
end for
return window-level replay record $\{(w_i, u_i)\}_{i=1}^{M}$.

windows and is trained with a listwise within-trace objective. This separation keeps deployment simple: the max-pooled gate score controls whether to spend compute, while the ranker decides where to spend it.

C.2 Online Selector Features and Proxy Rankers

The default activation gate and numeric window ranker share features computed from the base trace before any intervention. Table 10 lists the inputs. Window position is $(i-1)/(M-1)$ for $M > 1$ (zero for a single window), and relative prefix length is the ratio of prefix tokens to trace tokens. Binary markers indicate equation markup, final-answer markers, explicit step markers, and fenced code; density features measure the fractions of digits and non-alphanumeric symbols within each window.

The proxy rankers in Figure 4(c) each use a single scalar feature: normalized window position, window length in model tokens, or mean token negative log-likelihood (NLL),

$$\mathrm{NLL}(w_i) = -\frac{1}{|w_i|} \sum_{t \in w_i} \log p(z_t \mid x, z_{<t}).$$

Each proxy is trained with the same replay-derived listwise objective as the numeric ranker, learning to rank windows by predicted utility rather than directly sorting the raw feature. Mean token NLL is used only by the uncertainty proxy, not by the default deployed selector.

C.3 Window Ranker Variants

We evaluate three variants of the window ranker. The default variant is a numeric ranker, which uses only online-visible structural, positional, and surface features. It is the main paper setting because it keeps the selector lightweight and avoids making the method depend on a particular embedding backbone.

The semantic variants augment the numeric features with frozen text embeddings of the candidate window. They keep the same trace-conditioned ranking objective, but add a text tower and a gated fusion step before producing window scores. We consider a compact sentence-embedding variant based on all-MiniLM-L6-v2 and a larger variant based on Qwen3-Embedding-0.6B. These variants

Table 9: Information availability during online deployment and offline replay.

Field group	Visible at deployment	Offline only
Problem and base trace (x, τ)	✓	✗
Window text and position $(w_i, \mathrm{pos}(i))$	✓	✗
Window features	✓	✗
Offline replay branch text $\tilde{w}_i^{(k)}$	✗	✓
Offline replay-intervened trace $\tilde{\tau}_i^{(k)}$	✗	✓
Replay utility u_i	✗	✓
Answer change and divergence	✗	✓
Gold answer and correctness	✗	✓

Table 10: Online-safe numeric features used by the default selector.

Group	Features
Structural features	`window_position`, `num_windows`, `span_token_len`, `span_word_count`, `span_line_count`, `trace_token_len`, `relative_prefix_len`
Lightweight format cues	`span_has_equation`, `span_has_final_answer_marker`, `span_has_step_marker`, `span_has_code_block`, `span_digit_density`, `span_symbol_density`

test whether explicit semantic representations improve window selection; they are not required by the main method.

Empirically, the MiniLM semantic ranker performs best among the three variants, followed by the numeric ranker, while the Qwen3-Embedding variant does not yield further gains. We interpret this as a diagnostic result rather than the central contribution. Local reasoning windows are short and heterogeneous, and the selector is trained from sparse replay utility signals. Under this regime, a compact sentence encoder can provide a smoother inductive bias, while a larger frozen embedding model is not automatically better when paired with a shallow fusion head. For this reason, we keep the numeric ranker as the default paper setting and report semantic rankers as optional variants.

C.4 Training Hyperparameters

Table 11 summarizes the training settings. The activation gate is a lightweight window-level binary model, with trace activation obtained by max pooling over window scores. The rankers are trained with a listwise objective over windows in the same trace.

C.5 Compute Accounting

All compute comparisons use realized completion-token ratios rather than nominal controls such as sample count, activation threshold, or selected-window budget. For method m, we define

$$r_m = \frac{\sum_x c_m(x)}{\sum_x c_{\mathrm{base}}(x)},$$

where $c_{\mathrm{base}}(x)$ is the number of completion tokens generated by base decoding and $c_m(x)$ counts all completion tokens generated by method m on the same input. For full-trajectory baselines, this includes all sampled solutions or revision outputs. For local-intervention methods, this includes the base trace, local branch generations, and suffix rollouts; traces not activated by the selector incur no additional rollout cost. All points in the main compute–performance frontier are placed at this measured ratio.

D Additional Sensitivity Analyses

This section reports bootstrap confidence intervals and sensitivity analyses of four factors: the activation threshold, local rollout budget, window granularity, and window ranker. Sensitivity results use the primary model and are averaged over the four task families. Transfer and intervention-operator analyses are reported in Appendix E.3.

Table 11: Hyperparameters for selector training. The activation gate controls whether to spend compute, while the window-ranker family allocates compute across windows within activated traces.

param	**activation_gate**	**window_ranker variants**		
		numeric_ranker	minilm_ranker	qwen_emb_ranker
role	activate trace	rank windows	rank windows	rank windows
model	logistic	trace_context_mlp	gated_semantic_ranker	gated_semantic_ranker
input	online_features	online_features	online_features + text_emb	online_features + text_emb
text_model	–	–	all-MiniLM-L6-v2	Qwen3-Embedding-0.6B
target	binary_activation	rank_target	rank_target	rank_target
loss	log_loss	listwise_ce	listwise_ce	listwise_ce
hidden_dim	–	64	64	64
dropout	–	0.10	0.10	0.10
optimizer	L-BFGS	AdamW	AdamW	AdamW
lr	–	$1e{-3}$	$1e{-3}$	$1e{-3}$
regularization	$C = 1.0$	wd=$1e{-4}$	wd=$1e{-4}$	wd=$1e{-4}$
batch_size	full batch	16	16	16
max_iter/epochs	1000 iters	50 epochs	50 epochs	50 epochs
patience	–	10	10	10
listwise_temp	–	1.0	1.0	1.0

Table 12: Paired bootstrap 95% confidence intervals for representative performance and repair differences. Differences are reported in percentage points. Main-score intervals are macro-averaged over model–task pairs and resample examples; Repair@3 intervals are macro-averaged over tasks and resample originally incorrect base traces.

Setting	Comparison	Mean Δ	95% CI
Main performance	LEVER vs. Base	+4.2	[+3.5, +4.9]
Main performance	LEVER vs. strongest non-LEVER	+2.0	[+1.1, +2.8]
Controlled Repair@3	LEVER vs. Random	+28.9	[+25.1, +32.4]
Controlled Repair@3	LEVER vs. LLM-Distilled	+8.0	[+5.4, +10.7]

D.1 Bootstrap Confidence Intervals

Table 12 reports paired bootstrap 95% confidence intervals for representative performance and repair differences. For each model–task pair, we resample examples with replacement 10,000 times and recompute the paired score difference between LEVER and the comparison method. For controlled repair analyses, resampling is performed over originally incorrect base traces. Intervals are given by the 2.5th and 97.5th percentiles of the bootstrap distribution.

D.2 Activation Threshold Sensitivity

The activation threshold controls the trace-level spending decision. Here, trace activation rate denotes the fraction of base traces selected by the activation gate for additional local intervention; it is distinct from the number of windows selected within an activated trace. Table 13 reports the trade-off between trace activation rate, realized cost, and downstream performance. Lower thresholds activate more traces and increase cost, while more selective thresholds reduce cost but may miss repairable traces. The primary operating point is chosen to balance performance gain and realized cost.

D.3 Local Budget Sensitivity

Table 14 varies the number of selected windows B_w and the number of online branches per selected window K_{on}, while keeping the activation threshold fixed at the primary setting. The row with $(B_w, K_{\text{on}}) = (3, 1)$ corresponds to the primary operating point used in the main results. Repair generally benefits from a larger local budget, although the trend is not strictly monotonic and the gains are modest. This pattern is consistent with the concentration analysis in the main text: most useful intervention opportunities can be captured with a small local budget.

Table 13: Sensitivity to the activation threshold on the primary model. Trace act. rate denotes the fraction of base traces selected for additional local intervention. Avg. score is averaged over MATH-500, GPQA Diamond, HumanEval, and MMLU-Pro.

Gate setting	Trace act. rate	Cost	Avg. score	Δ over base
Very selective	9.7%	1.08×	82.0	+1.5
Selective	18.4%	1.17×	83.8	+3.2
Primary	29.1%	1.31×	84.9	+4.3
Broad	41.8%	1.46×	85.1	+4.5
Very broad	56.3%	1.63×	84.6	+4.0
All traces	100.0%	2.12×	84.1	+3.5

Table 14: Sensitivity to the local intervention budget on the primary model. The activation threshold is fixed at the primary setting; only B_w and K_{on} are varied. Repair is measured on originally incorrect traces under the specified online budget.

Setting	B_w	K_{on}	Cost	Repair
Top-1, one branch	1	1	1.15×	72.8
Top-2, one branch	2	1	1.23×	75.1
Primary	3	1	1.31×	74.6
Top-5, one branch	5	1	1.45×	77.0
Top-3, two branches	3	2	1.57×	77.3

D.4 Window Granularity

Window construction determines the granularity at which local intervention is applied. Table 15 varies the target number of windows while keeping the replay and evaluation protocol fixed. Very coarse windows reduce localization precision, while overly fine windows make the ranking problem noisier. The default setting provides a stable middle ground.

D.5 Ranker Variants

Table 16 compares three window-ranker variants under the same replay-derived listwise target. The numeric ranker is the default setting used in the main paper. The semantic variants add frozen text embeddings of the candidate window. The compact MiniLM variant slightly improves ranking quality, while the larger Qwen embedding variant does not yield further gains. This suggests that local window selection benefits from low-variance semantic features, but not necessarily from larger frozen embedding models.

E Additional Analyses

The Qwen3.5-9B analyses use the same candidate-selection policy as the main experiments. Cross-model transfer is evaluated separately on Phi-4-Reasoning. MMLU-Pro controls use a fixed 500-example subset of the test split (MMLU-Pro-500), while the 602 replay examples are drawn from the training split. The primary MMLU-Pro results use the full test split.

E.1 Matched Online Budget and Replay-Supervised Control

For the matched-budget comparison, each method is evaluated within $0.02\times$ of LEVER's realized task-level completion-token ratio. Baselines without a gate allocate additional inference to randomly selected queries at the corresponding rate, while gated variants tune their activation threshold. This matches online generation cost but not offline supervision. Table 17 reports the results.

The replay-supervised global-revision baseline shares LEVER's replay supervision and activation gate but revises the full trajectory without selecting local windows.

Table 15: Sensitivity to window granularity on the primary model. Mass@1 and Mass@3 measure oracle concentration under the corresponding segmentation; NDCG@3 evaluates the deployable ranker, and Repair is measured under Top-3 online selection.

Target windows	Avg. windows	Mass@1	Mass@3	NDCG@3	Repair (%)
4	4.2	0.67	0.94	0.49	74.5
6	5.6	0.58	0.90	0.53	75.8
8	7.2	0.41	0.87	0.55	74.6
10	9.2	0.36	0.84	0.54	74.9
12	10.5	0.29	0.81	0.52	71.1

Table 16: Ranker variant comparison on the primary model. The numeric ranker is the default paper setting; semantic rankers are optional diagnostic variants.

Ranker	NDCG@3	Repair@3	Avg. score
Numeric ranker	0.55	74.6	84.9
MiniLM semantic ranker	0.57	77.4	85.3
Qwen embedding ranker	0.49	75.6	82.6

E.2 One-Time Offline Replay Cost and Amortization

Table 18 reports the one-time cost of constructing the task-specific Qwen3.5-9B selector and the corresponding token break-even estimates. Offline completion tokens include base traces, replay branches, and regenerated suffixes. Base-query equivalents normalize this cost by the mean base-trace token count. GPU-hours are measured on one NVIDIA RTX PRO 6000, with selector-training time reported separately. Online ratios in the main text exclude these construction costs.

For Q future queries, total LEVER cost in base-query equivalents is $C_{\text{off}}^{\text{eq}} + r_{\text{LEVER}}Q$, compared with $r_{\text{BoN}}Q$ for Best-of-N. Hence

$$Q_{\text{BE}} = \frac{C_{\text{off}}^{\text{eq}}}{r_{\text{BoN}} - r_{\text{LEVER}}}.$$

The final column gives the resulting break-even query counts. This estimate assumes that the selector is reused without retraining or distribution shift. It compares completion tokens only and does not imply a wall-clock break-even point.

E.3 Transfer and Alternative Intervention Operators

We evaluate zero-shot transfer with the source selector and activation threshold unchanged, without target replay or validation labels. For cross-task transfer, the selector is trained on the other three task families. For cross-model transfer, a pooled Qwen3.5-9B selector is applied to Phi-4-Reasoning; the reported score is a four-task macro-average. Table 19 summarizes the transfer results.

To assess dependence on the intervention operator, pre-window suffix resampling retains the prefix before w_i and regenerates the remaining suffix without explicitly replacing the selected window. Across the four task families, local replacement produces stronger utility concentration than pre-window suffix resampling (Mass@3: 0.87 vs. 0.82). The same structural pattern also holds on Phi-4-Reasoning (Mass@3: 0.84).

E.4 Replay Stability and Deployment Variants

On GPQA Diamond, replay is repeated across five seeds, with up to five branches per window and $K_{\text{rep}} = 1, 3, 5$. Mass@3 remains between 0.85 and 0.90, while top-three window overlap increases from 61.2% to 78.6% as more branches are used. This indicates that the broad concentration pattern is stable despite variation in individual selected windows.

The default candidate selector uses generated token likelihoods. With generated candidates held fixed, a prompt-based text-only candidate selector achieves a comparable average task score (84.3 vs. 84.9) at a similar online cost (1.33× vs. 1.31×). This alternative requires access to the generated reasoning trace and an API that allows it to be supplied in a follow-up prompt; it is not an evaluation on proprietary closed-source models. On HumanEval, syntax and required-function checks filter

Table 17: Scores in the matched-budget comparison. The MMLU-Pro column uses the fixed 500-example test subset. Self-Consistency does not apply to HumanEval.

Method	MATH	GPQA	HumanEval	MMLU-Pro-500	Avg.
Base decoding	88.6	79.8	77.4	76.4	80.6
Self-Consistency	89.0	79.8	–	76.8	–
Best-of-N	89.0	80.3	77.4	76.8	80.9
Global Revision	88.8	78.3	76.8	77.0	80.2
Global Revision (replay-supervised)	90.4	81.8	79.3	78.8	82.6
LLM-Distilled Selector	89.8	80.8	78.7	78.4	81.9
LEVER	**92.0**	**84.8**	**82.3**	**80.6**	**84.9**

Table 18: Offline construction cost and token break-even versus Best-of-N. Online ratios are reported as LEVER/Best-of-N.

Task	Replay examples	Offline tokens	Base-query eq.	Online ratio	GPU-h	Train min	Break-even Q
MATH-500	500	4.17M	6.80k	1.34/3.07	19.5	1.5	3.93k
GPQA Diamond	198	2.66M	2.81k	1.36/3.11	12.4	1.0	1.61k
HumanEval	164	0.46M	2.06k	1.27/3.03	2.2	0.9	1.17k
MMLU-Pro	602	4.20M	7.82k	1.29/3.13	19.6	1.6	4.25k

malformed repair candidates, while retaining the original program among candidates helps limit regressions on initially correct programs.

F Qualitative Cases and Limitations

This section clarifies the scope of our claims and provides representative examples. We include short excerpts from replay artifacts to illustrate the failure categories analyzed in the main text.

Figure 6 presents representative local-intervention cases. Each panel provides minimal task context, the original and target outcome, the relevant local window, and the resulting intervention outcome. The examples are selected to illustrate four common patterns observed in our replay analysis.

Panel (a) shows a successful early repair. The original trace makes an early counting commitment by treating six circular units as $(5-1)! = 4!$. Replacing this local window with the correct circular count, $(6-1)! = 5!$, allows the regenerated suffix to recover the correct final answer. Panel (b) illustrates a weak repair. The selected region is relevant: the original code restricts prime candidates to $[2, 3, 5, 7]$ for a task that requires checking products of exactly three primes below 100. However, the sampled branch only partially corrects the prime-generation logic and still fails to produce a valid implementation. Panel (c) shows missed localization. The decisive window is the distinction between perishability and non-recoverability, but the assumed selected region focuses on earlier distractor elimination. This case reflects localization error rather than the absence of a corrective branch. Panel (d) illustrates residual failure. Some sampled branches touch useful ideas, such as $a \geq 3$ or the equality case $r_1 = r_2 = r_3 = -1$, but no sampled local replacement completes the global proof.

Together, these cases show that successful repair can arise from compact local corrections, while remaining errors can be attributed to weak branch generation, missed localization, or failures that require more than the current single-window local operator.

F.1 Limitations

Offline replay cost. LEVER separates discovery from deployment. Counterfactual replay is used to estimate recoverable utility and supervise the online selector, but it requires evaluating multiple local branches per window. This cost is not part of online inference cost and is excluded from the reported deployment ratios. The offline cost is quantified in Table 18; amortization depends on sufficient subsequent deployments or successful zero-shot transfer and does not follow from the $1.31\times$ online ratio alone.

Dependence on trace structure. The method assumes that generated traces contain enough visible structure for local windows to be meaningful. Very short traces, highly unstructured outputs, or

Table 19: Zero-shot transfer. Parentheses show absolute improvement over corresponding base decoding.

Transfer setting	Repair@3 (%)	Score (gain)	Online cost
Other tasks → MATH-500	72.8	91.2 (+2.6)	1.33×
Other tasks → GPQA Diamond	67.5	83.8 (+4.0)	1.36×
Other tasks → HumanEval	64.2	81.1 (+3.7)	1.29×
Other tasks → MMLU-Pro-500	58.0	79.4 (+3.0)	1.30×
Qwen3.5-9B → Phi-4-Reasoning	63.4	76.4 (+2.8)	1.34×

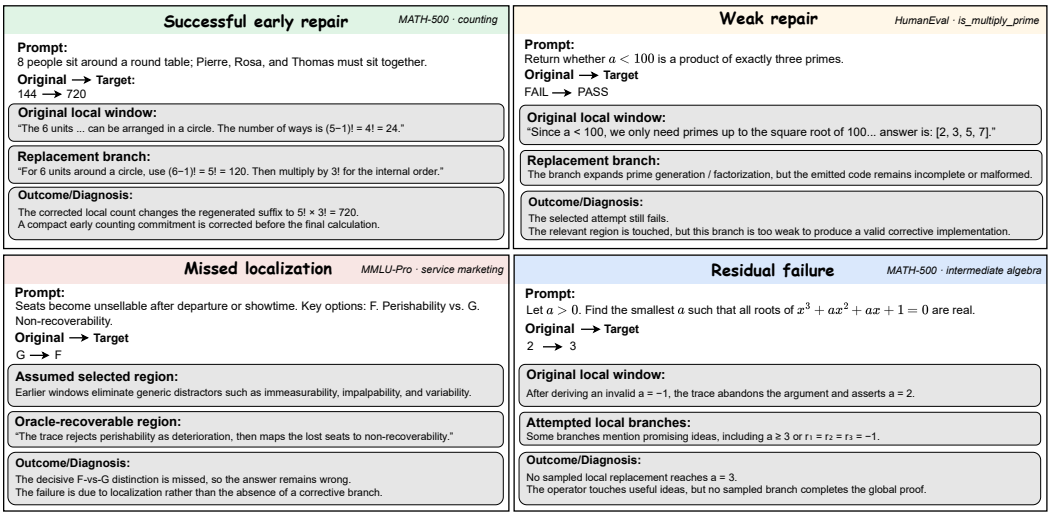

Figure 6: Representative local-intervention cases. Each panel gives minimal task context, the original and target outcome, and the local region responsible for success or failure. The cases illustrate four common patterns: early repair, weak branch generation, missed localization, and residual failure beyond the current local operator.

severe formatting errors can make segmentation noisy and reduce the reliability of window ranking. Proprietary APIs may limit access to visible reasoning traces or editable continuations; the present evaluation uses open-weight models.

Task verifiability. Recoverable utility is defined through an outcome metric. Our experiments use tasks with automatic evaluation, such as exact-answer checking or code execution. Extending the method to open-ended generation would require a reliable evaluator, preference signal, or alternative utility definition.

Locality of the intervention operator. The default operator replaces one window at a time and regenerates the suffix. Some failures may require coordinated changes across distant commitments, external knowledge, or broader search. These cases are reflected in the residual-failure category of the failure analysis. The second operator and additional model replication in Appendix E.3 support concentration beyond one configuration but do not establish that all reasoning errors are intrinsically sparse.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's main empirical and methodological claims: recoverable utility is concentrated in a small number of reasoning windows, this structure is not explained by simple cues, and LEVER improves the performance–cost trade-off under measured inference budgets. The claims are supported by the main results in Table 1, mechanistic analyses in Tables 2–3, and Figures 3–5. Additional matched-budget, replay-supervised, cross-model, and operator controls appear in Appendix E.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations through the residual-gap and failure-structure analysis in Section 4.4 and the scope notes in Appendix F. These include branch-quality failures, missed localization, residual cases requiring broader or multi-window changes, dependence on visible trace structure, reliance on automatically verifiable task outcomes, and the distinction between offline replay discovery and deployable test-time selection. Appendix E.2 reports the one-time replay cost and conditions for amortization, and Appendix E.3 clarifies the scope of the structural claim.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results or theorems requiring proofs. The mathematical definitions in Section 3 define the local intervention operator, recoverable utility, Mass@k, and selector training objectives for the proposed empirical method.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper specifies the benchmark suite, base models, evaluation metrics, prompt and answer-extraction protocol, data splits, local replay procedure, selector training setup, intervention budget, and realized-token accounting. The main text and Appendices A, C, and C.5 provide the information needed to reproduce the main experimental protocol and interpret the reported results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The appendix provides detailed experimental settings, including benchmark splits, prompt templates, window construction, replay configurations, selector features, generation settings, and evaluation protocols. The data and code will be publicly available.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 describes the benchmarks, models, baselines, evaluation metrics, split strategy, replay budget, deployment budget, activation rate, and realized-token accounting. Additional prompt templates, answer extraction rules, decoding settings, selector hyperparameters, feature boundaries, and compute accounting are provided in Appendices A and C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Appendix D.1 reports paired bootstrap 95% confidence intervals for representative main performance differences and controlled Repair@3 comparisons. Intervals are computed by example-level resampling with replacement; for Repair@3, resampling is over originally incorrect base traces.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix A.4 reports the generation and evaluation setup, including the serving backend, GPU hardware, model-level decoding settings, generated-completion-token accounting, and offline replay cost treatment. Table 1 reports realized deployment-time cost ratios for each method.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: The work uses public benchmarks and open-weight language models for reasoning evaluation. We do not collect private data, conduct human-subject experiments, or release high-risk models or datasets. The research is designed and reported in accordance with the NeurIPS Code of Ethics.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The paper studies foundational test-time reasoning methods. Potential positive impacts include improved compute efficiency and better allocation of inference resources. Potential negative impacts include enabling stronger reasoning capabilities in downstream applications, including harmful or unreliable uses if deployed without appropriate safeguards.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not introduce or release a new pre-trained model, image generator, scraped dataset, or other high-risk asset. Experiments are conducted using existing public benchmarks and open-weight models.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the original sources for the public benchmarks and open-weight models used in the experiments, including MATH-500, GPQA Diamond, HumanEval, MMLU-Pro, Qwen3.5-9B, Phi-4-Reasoning, and Mistral-Small-3.1-24B-Instruct-2503. The appendix documents the assets and evaluation protocol; their licenses and terms of use are respected.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not introduce a new dataset, benchmark, released model, or other standalone research asset. The contribution is a test-time intervention method and accompanying empirical analysis using existing benchmarks and models.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or human-subject research. All evaluations are conducted on existing public benchmarks.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments, collection of human-subject data, or experiments with human participants, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The paper's core method and baselines use language models for base reasoning traces, local branch generation, suffix regeneration, and full-trajectory baselines. The LLM-Distilled Selector baseline also uses offline LLM-generated window ratings. These uses are described in Sections 3–4 and the appendix.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.