GRACER: GRAPH REGULATION AND ADAPTIVE COMPRESSION FOR EFFICIENT LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Chain-of-thought reasoning improves large language models by making intermediate reasoning steps explicit. Recent methods organize intermediate reasoning into structured forms such as trees, graphs, and forests. However, these methods often produce verbose or inconsistent structures. To address this issue, we introduce GRACER, Graph-Regularized Adaptive Compression for Efficient Reasoning. GRACER first regularizes noisy dependency structures through role-constrained repair. A critic then estimates structural credit and downstream uncertainty over the dependency graph. These estimates guide greedy compression of redundant nodes under the information-bottleneck principle. Across six benchmarks and two backbone models, GRACER produces more compact reasoning graphs and improves accuracy. GRACER consistently outperforms AoT, the strongest baseline in our comparison. Average performance rises from 82.42% to 86.05% on DeepSeek-V3 and from 79.13% to 81.33% on GPT-4o-mini. Compared with AoT, GRACER reduces average token usage per instance from 26.48K to 23.21K and average LLM calls by 21.5%. Role-constrained repair also reduces structural error rates from 32% to 12%, supporting the effectiveness of graph regularization.

1 INTRODUCTION

Large language models (LLMs) have achieved substantial improvements when intermediate reasoning is made explicit during inference. Chain-of-thought (CoT) prompting shows that eliciting intermediate rationales can improve performance on complex reasoning tasks (Wei et al., 2022), while zero-shot CoT demonstrates that such reasoning behavior can be induced with lightweight instructions (Kojima et al., 2022). Beyond eliciting a single reasoning trace, subsequent methods have made inference-time reasoning more structured. Least-to-most prompting (Zhou et al., 2023) decomposes difficult problems into easier subproblems, and self-consistency (Wang et al., 2023) improves robustness by aggregating multiple sampled reasoning paths. More recent structured frameworks further organize intermediate computation into explicit non-linear forms: Tree-of-Thoughts (Yao et al., 2023a) explores tree-structured candidate thoughts, Graph-of-Thoughts (Besta et al., 2024) represents reasoning as graph-structured states, and Atom of Thoughts (Teng et al., 2025) formulates reasoning through decomposition–contraction states. These methods indicate a shift in test-time reasoning from producing isolated textual rationales to constructing structured intermediate states during inference.

However, this structural shift leaves a critical gap in current test-time reasoning pipelines. Once reasoning is decomposed into sub-questions, intermediate claims, or local reasoning units, the intermediate state is no longer a linear rationale that can be shortened uniformly. It becomes a dependency structure whose regions differ in their relevance to the final answer and in their sensitivity to compression. Existing methods mainly use such structures to support search, aggregation, or contraction, while offering limited analysis of the generated structure itself. Consequently, post-decomposition management is often reduced to flat trace summarization, local pruning, or heuristic confidence estimation. These operations lack a structural criterion for deciding which prerequisite substructures must be preserved and which redundant regions can be abstracted. This mismatch suggests that efficient structured reasoning is not simply a matter of shortening traces, but of deciding how much fidelity each part of a dependency graph should retain before contraction.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

This gap motivates GRACER, Graph-Regularized Adaptive Compression for Efficient Reasoning. Rather than treating intermediate reasoning units as isolated text spans, GRACER maps them into a dependency directed acyclic graph (DAG). Given the raw dependency graph, GRACER first repairs malformed edges and refreshes node roles to improve structural consistency. It then applies a no-reference evaluator, which estimates structural credit and downstream uncertainty without relying on gold answers, step-level annotations, or reference reasoning graphs. These estimates subsequently guide the selective compression of low-value regions before contraction. Figure 1 illustrates the overall pipeline. In this view, the central question is not simply how to produce longer or shorter reasoning traces, but how to allocate fidelity across a structured reasoning graph.

Our framework is composed of three technical components:

- **No-reference graph evaluation.** We introduce a graph evaluator that evaluates reasoning units by jointly modeling structural credit and downstream uncertainty. Concretely, it combines reverse personalized PageRank with a weighted maximum-entropy path model, providing an interpretable signal for deciding which regions deserve higher fidelity. The signal is derived from the generated graph itself, without external reference answers or human annotations.
- **Graph compression via the information bottleneck principle.** We formulate reasoning compression as a fidelity-allocation problem over local graph structures rather than over flat traces. This yields a policy that can keep, summarize, or drop specific reasoning regions according to an explicit risk–length trade-off.
- **Role-constrained repair for graph regularization.** We use repair as a pre-processing step that regularizes noisy dependency structures. Before evaluation, the framework repairs malformed dependencies and refreshes node roles so that later efficiency decisions are made on a coherent reasoning structure.

2 Related Work

Structured Reasoning Frameworks. A broad line of work improves LLM reasoning by making intermediate computation explicit. Chain-of-thought prompting (Wei et al., 2022) elicits step-by-step rationales, while scratchpads (Nye et al., 2022) treat intermediate computation as an external workspace. Program-of-Thoughts (Chen et al., 2023) delegates subtasks to executable programs, and ReAct (Yao et al., 2023b) interleaves reasoning with external actions. Beyond linear traces, recent methods adopt non-linear structures: Tree-of-Thoughts (Yao et al., 2023a) explores tree-structured candidate thoughts, Graph-of-Thoughts (Besta et al., 2024) represents reasoning as a graph of transformable thought states, and Forest-of-Thoughts (Bi et al., 2025) scales reasoning across multiple trees. Related planning-based methods, such as Language Agent Tree Search (Zhou et al., 2024), further combine reasoning, acting, and planning within tree search. System-level frameworks such as AFlow (Zhang et al., 2025) coordinate multi-step agent reasoning across stages. These approaches enable explicit reasoning structures for search, aggregation, planning, and tool use. Their focus is mainly on constructing or coordinating such structures, leaving open how to assess and manage the intermediate dependency structure itself.

Reasoning Efficiency. Another line of work improves the reliability and efficiency of inference-time reasoning once intermediate reasoning is available. Self-refinement methods (Madaan et al., 2023) treat intermediate reasoning as a revisable draft and use model feedback to improve outputs. Process-supervision methods (Uesato et al., 2022) assess intermediate reasoning steps, while verifier-based approaches (Lightman et al., 2024) evaluate reasoning trajectories to improve final-answer reliability. Faithfulness-oriented work (Lyu et al., 2023) further asks whether generated rationales genuinely support model predictions rather than merely accompany them. Beyond reasoning-step and trajectory assessment, system-level approaches allocate computation across inference stages. Language-model cascades (Dohan et al., 2022) route queries across models or stages, and compound AI systems (Chen et al., 2024) study how multiple calls and modules interact under inference-time scaling. These approaches provide useful mechanisms for revision, verification, faithfulness analysis, and compute allocation. However, their efficiency decisions are usually made at the level of complete trajectories, local reasoning steps, final answers, or workflow configurations. When reasoning is represented as a dependency graph, shortening one unit can change the information available to

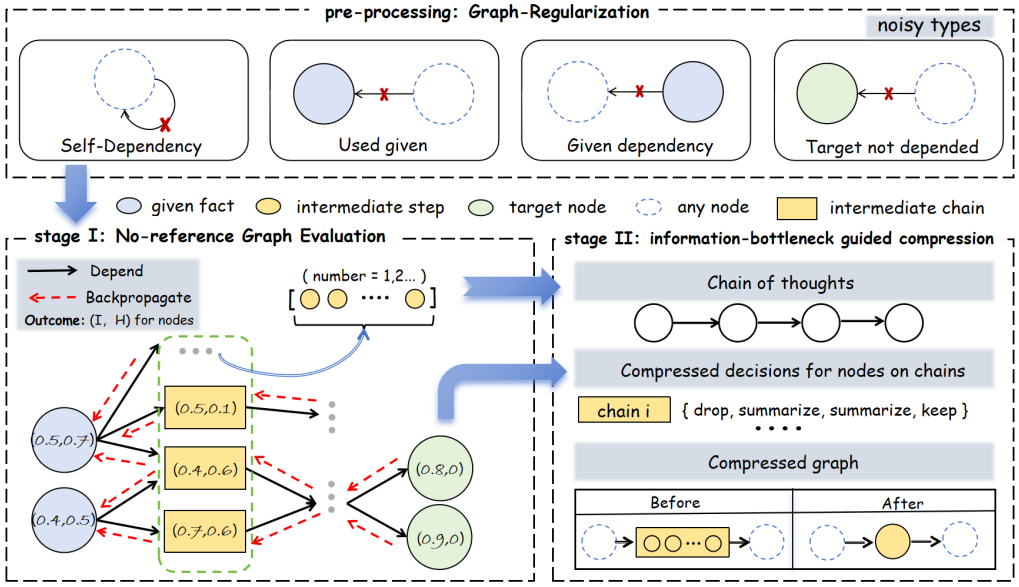


Figure 1: Overview of GRACER. Given decomposed reasoning as input, GRACER constructs and regularizes a dependency graph, estimates structural credit and downstream uncertainty with a no-reference evaluator, and uses these signals to selectively compress low-risk local chains before contraction.

downstream units. This creates a distinct problem of compression and fidelity allocation under explicit dependency structure.

3 METHODOLOGY

Design rationale. This is a pragmatic rather than theoretically necessary combination. Drawing on graph analysis, statistical modeling, and information theory, it adapts tools that are not commonly used together to manage intermediate LLM reasoning graphs. The pipeline poses three practical problems: estimating answer-facing support, assessing downstream compression risk, and balancing fidelity against length. We address these problems with reverse PageRank, a maximum-entropy path model, and an IB-inspired rule, respectively. The main results and ablations show that this division of labor consistently improves accuracy and efficiency. Figure 1 illustrates the overall workflow.

Preliminaries. We consider structured test-time reasoning under the decomposition–contraction paradigm (Teng et al., 2025). In this setting, decomposition breaks a problem into explicit intermediate reasoning units and exposes dependencies that are implicit in a single textual rationale. Contraction then condenses the retained reasoning content into a shorter self-contained form for downstream answer synthesis. We focus on the intermediate reasoning state between these two operations. Rather than treating this state as a flat collection of textual units, we represent it as an explicit dependency graph.

We represent this intermediate state as a directed acyclic graph:

$$G = (V, E), \qquad V = \{v_1, \ldots, v_n\}, \qquad E \subseteq V \times V,$$

where each node $v_i \in V$ corresponds to a reasoning unit and each edge $(v_i, v_j) \in E$ records that v_j depends on v_i for subsequent inference. The acyclicity assumption reflects the forward dependency order of decomposed reasoning: later units may depend on earlier premises or derivations, but not vice versa. When the raw decomposition violates these structural constraints or produces inconsistent dependencies, we regularize the graph through the repair procedure described next.

3.1 GRAPH EVALUATION

Fine-grained role assignment and structural repair. Our graph analysis begins with a basic observation: nodes in a structured reasoning representation contribute unevenly to the final answer, and this contribution cannot be reliably inferred from local text alone. Raw decomposition outputs are often too coarse and noisy for direct graph evaluation, as units that provide premises, perform derivations, or directly support the target may be ambiguous or misaligned with the dependency structure. We therefore first assign each node one of three roles: GIVEN nodes encode premises, retrieved facts, or directly available conditions; INTERMEDIATE nodes capture nontrivial derivations, transformations, and bridging steps; and TARGET nodes represent terminal reasoning units that directly support answering the original question.

Let x denote the input question and let c denote optional external context. Before subsequent analysis, we further regularize the graph via a bounded repair operator:

$$G^{(t+1)} = \Phi\Big(x, c, G^{(t)}, \mathcal{R}(G^{(t)})\Big), \tag{1}$$

where $\mathcal{R}(G^{(t)})$ denotes a defect report that summarizes structural errors detected in the current graph, and Φ produces a revised graph conditioned on the original problem, optional context, and the defect report. In practice, we observe four recurring structural violations in decomposed reasoning graphs (see Figure 1): self-dependencies, GIVEN nodes with nonempty dependencies, isolated GIVEN nodes, and TARGET nodes that are incorrectly used as prerequisites for later reasoning units. The repair process terminates when no violations remain or when a preset retry budget is reached. After repair, node roles are updated according to the revised topology to ensure consistency between node function and dependency structure. This step turns a noisy decomposition into a coherent reasoning graph suitable for the subsequent credit and uncertainty analysis.

Structural credit propagation. Given the resulting reasoning graph, we estimate a structural credit score I_i for each node to measure its support for target-facing reasoning. Since prerequisite nodes can influence the final answer through downstream reasoning units, credit should propagate backward along dependency edges. Inspired by the propagation principle of PageRank (Brin & Page, 1998), we define node credit as the stationary solution of a reverse dependency update:

$$\mathbf{I} = (1-d)\mathbf{S} + dM^{\top}\mathbf{I}, \tag{2}$$

where $\mathbf{I} = [I_1, \ldots, I_{|V|}]^{\top}$ is the node credit vector, $\mathbf{S} = [S_1, \ldots, S_{|V|}]^{\top}$ is the structural-prior vector, $d \in (0,1)$ is the damping coefficient, and $M^{\top}$ is the reverse dependency transition matrix defined by

$$(M^{\top})_{ij} = \frac{\mathbf{1}(v_i \to v_j)}{|\mathrm{Pa}(j)|}. \tag{3}$$

Here, $\mathbf{1}(v_i \to v_j)$ equals 1 when v_i is a parent of v_j, and 0 otherwise. The set $\mathrm{Pa}(j)$ denotes the parent nodes of v_j, so the credit of a child node is distributed among its direct prerequisites. Equivalently, Eq. (2) can be written in node-wise form as

$$I_i = (1-d)S_i + d\sum_{j \in \mathrm{Ch}(i)} \frac{I_j}{|\mathrm{Pa}(j)|}. \tag{4}$$

Here $\mathrm{Ch}(i)$ denotes the set of child nodes of v_i. This update assigns credit to a node both from its own structural prior and from the downstream reasoning units that depend on it.

To instantiate this propagation model, we use the following structural prior:

$$S_i = \frac{\alpha\,\mathrm{outd}(v_i) + \beta\,\mathrm{dep}(v_i) + \gamma\,\log\big(l(v_i)\big)}{\sum_{j \in V}\left[\alpha\,\mathrm{outd}(v_j) + \beta\,\mathrm{dep}(v_j) + \gamma\,\log\big(l(v_j)\big)\right]}, \tag{5}$$

where $\mathrm{outd}(v_i)$ is the out-degree of node v_i, $\mathrm{dep}(v_i)$ is its depth-related structural score, $l(v_i)$ denotes its description length, and α, β, γ are weighting coefficients. This prior assigns larger base mass to nodes that are more topologically central, closer to the target region, or carry more textual content. Under Eq. (2), a node can receive high credit either because it is structurally salient by itself or because it supports descendants that are closely tied to the final answer. Thus, reverse propagation

emphasizes prerequisite regions that sustain answer synthesis, rather than treating all intermediate units as equally important.

Structural credit measures how strongly a node supports the final answer, but it does not indicate whether this support is concentrated along a stable continuation or dispersed across multiple downstream routes. Some high-credit nodes lie on a narrow set of paths to the target region, whereas others branch into several plausible continuations, making them more sensitive to compression. We therefore complement structural credit with a downstream uncertainty estimate.

Maximum-entropy downstream uncertainty. For each node i, let $\mathcal{P}_i$ denote the set of directed paths from v_i to a TARGET node. We characterize the uncertainty over these paths using a maximum-entropy distribution (Berger et al., 1996), which incorporates feature-based constraints while avoiding unnecessary assumptions elsewhere. Concretely, let $\phi(p) = [\phi_1(p), \ldots, \phi_K(p)]^\top$ be a vector of structural features for a path p. For each feature k, we enforce that its model expectation $\mu_{i,k}$ matches a credit-weighted empirical target $\bar{\phi}_{i,k}$:

$$\mu_{i,k} = \bar{\phi}_{i,k}, \ \forall k, \tag{6}$$

where

$$\mu_{i,k} = \sum_{p \in \mathcal{P}_i} P_i(p)\, \phi_k(p) \tag{7}$$

is the expected value of feature k under the model distribution P_i. Under these moment-matching constraints, the downstream path distribution takes the exponential-family form:

$$P_i(p) = \frac{\exp\left(\sum_{k=1}^{K} \lambda_{i,k} \phi_k(p)\right)}{Z_i}, \tag{8}$$

where Z_i is the normalization factor for the path distribution.

To adapt this principle to dependency-graph analysis, we instantiate the empirical feature statistics using a credit-conditioned path weighting scheme. Specifically, we define a credit-induced weight for each path p:

$$w_p = \exp\left(\sum_{j \in p} \log r_j\right), \ r_j := I_j, \tag{9}$$

which is equivalent to the product form $w_p = \prod_{j \in p} I_j$ and assigns larger weights to paths passing through high-credit nodes. The corresponding empirical targets are then computed as:

$$\bar{\phi}_{i,k} = \frac{\sum_{p \in \mathcal{P}_i} w_p\, \phi_k(p)}{\sum_{p \in \mathcal{P}_i} w_p}. \tag{10}$$

We estimate the multipliers $\{\lambda_{i,k}\}_{k=1}^{K}$ using Improved Iterative Scaling (IIS) (Berger et al., 1996) with the update rule:

$$\lambda_{i,k} \leftarrow \lambda_{i,k} + \frac{1}{C} \log \frac{\bar{\phi}_{i,k}}{\mu_{i,k}}, \tag{11}$$

where C is the IIS normalization constant, chosen as an upper bound on the total feature sum over all paths. The iteration terminates when a convergence criterion is satisfied:

$$\max_k \left| \lambda_{i,k}^{(t+1)} - \lambda_{i,k}^{(t)} \right| < \epsilon. \tag{12}$$

Finally, we define the uncertainty of a node as the entropy of its downstream path distribution:

$$H_i = -\sum_{p \in \mathcal{P}_i} P_i(p) \log P_i(p). \tag{13}$$

Salience–risk profile. The resulting pair (I_i, H_i) defines the *salience–risk profile* of node i: I_i measures answer-facing structural credit, while H_i measures how diffuse its downstream continuation is. A large H_i suggests that node i participates in multiple non-trivial downstream continuations, so aggressive compression may remove distinctions needed by later reasoning steps. Conversely, a small H_i indicates a relatively stable continuation pattern and therefore lower compression risk. This profile provides the quantitative foundation for the information-bottleneck-guided compression policy described in the next subsection.

3.2 INFORMATION-BOTTLENECK-GUIDED GRAPH COMPRESSION

We frame reasoning compression as selective fidelity allocation over the dependency graph: local substructures critical for the final answer should be preserved, whereas less important regions can be shortened or abstracted. The dependency structure specifies *where* compression may be applied without altering the overall reasoning topology, and the information bottleneck (IB) principle offers a compact formulation of the trade-off between compression and fidelity.

In its canonical form, the IB objective seeks a compressed representation Z of an input G that retains minimal information about G while preserving information relevant to the target Y (Tishby et al., 1999):

$$\min_{q(z|G)} I(G;Z) - \beta I(Z;Y), \tag{14}$$

where $I(\cdot;\cdot)$ denotes mutual information and $\beta > 0$ controls the trade-off between compression and target relevance. The first term penalizes retaining excessive information about G, whereas the second term rewards preserving information useful for predicting or supporting Y.

At test time, however, the true target Y is unavailable, and our method does not directly optimize Eq. (14). Instead, we adopt an IB-guided surrogate for local graph compression. Concretely, we extract contiguous INTERMEDIATE segments in which each node has exactly one parent and one child. These segments form compressible local chains that can be rewritten without altering the branching structure of the graph. For each candidate chain g and action $a \in \{\texttt{keep}, \texttt{summarize}, \texttt{drop}\}$, we define a risk-adjusted surrogate score:

$$R(g,a) = \widetilde{I}(g)\left(b_a + \kappa_a \widetilde{H}(g)\right) - \lambda_{\text{len}} \frac{\ell(g)}{L_{\max}}, \tag{15}$$

where $\widetilde{I}(g)$ and $\widetilde{H}(g)$ are the normalized chain-level structural credit and downstream uncertainty, $\ell(g)$ denotes the length of chain g, $L_{\max}$ is the maximum length among candidate chains, and (b_a, κ_a) are action-specific fidelity penalties. The first term estimates the fidelity cost of applying action a, while the second term encourages compression on longer candidate chains. Consequently, chains with higher structural credit or downstream uncertainty are more resistant to aggressive rewriting, whereas longer chains with lower surrogate scores are more amenable to compression.

Actions are ordered by increasing aggressiveness, $\texttt{keep} \prec \texttt{summarize} \prec \texttt{drop}$, and the policy selects the most aggressive action whose surrogate score satisfies the rewrite condition:

$$a^\star(g) = \max_{\preceq} \{a : \; R(g,a) \le \tau_a\}. \tag{16}$$

Here τ_a denotes the threshold for action a: an action is allowed only when its surrogate score falls below this threshold. When no compression action satisfies the threshold constraint, the policy defaults to `keep` and leaves the chain unchanged. Selected chains are either rewritten into shorter summaries or removed as permitted by the action, and the graph is re-validated before contraction. Thus, Eq. (15) serves as a task-specific, IB-guided surrogate for fidelity allocation, rather than a direct optimization of Eq. (14). We evaluate its effect on graph size in Section 4.

4 EXPERIMENTS

Our experiments evaluate whether GRACER improves structured test-time reasoning by optimizing the intermediate reasoning graph after decomposition. We examine final-answer performance, accuracy–cost trade-offs, component-level contributions, and the structural effects of repair and compression.

4.1 EXPERIMENTAL SETUP

Benchmarks and Metrics. We evaluate GRACER on six benchmarks spanning three representative reasoning settings: mathematical reasoning with MATH (Hendrycks et al., 2021) and GSM8K (Cobbe et al., 2021), multiple-choice reasoning with MMLU-CF (Zhao et al., 2025) and BBH (Suzgun et al., 2023), and multi-hop question answering with HotpotQA (Yang et al., 2018) and LongBench (Bai et al., 2024). We use task-appropriate metrics: exact-match accuracy for MATH and

GSM8K, option accuracy for MMLU-CF and BBH, and answer F1 for HotpotQA and LongBench. All methods are evaluated on the same instances under the same answer extraction rules. Further dataset-specific details are provided in Appendix A.9.

Backbones and Baselines. We report results under two backbone models: GPT-4o-mini-2024-07-18[1] and DeepSeek-V3 (DeepSeek-AI et al., 2024). We compare with CoT (Wei et al., 2022) and AoT (Teng et al., 2025), with AoT providing the closest comparison under the decomposition–contraction paradigm. Additional baselines include CoT-SC (Wang et al., 2023), Self-Refine (Madaan et al., 2023), ToT (Yao et al., 2023a), AFlow (Zhang et al., 2025), and FoT (Bi et al., 2025). Implementation details and supplementary diagnostics appear in Appendices A and B.

Evaluation Protocol and Budget Control. All methods share evaluation instances, backbones, answer-extraction rules, and metrics. Baselines follow their standard inference procedures, with token usage, runtime, and calls reported. GRACER's retry/repair budgets, decomposition depth, prompts, and compression policy remain fixed across benchmarks and backbones. Appendix A provides implementation details and settings.

4.2 Main Results

Table 1 shows that GRACER achieves the highest score across all six benchmarks under both backbones. Compared with AoT, the average improves from 82.42% to 86.05% on DeepSeek-V3 and from 79.13% to 81.33% on GPT-4o-mini, gains of 3.63 and 2.20 percentage points. The largest DeepSeek-V3 gains occur on MMLU-CF (6.40 points) and HotpotQA (5.20 points), suggesting benefits when reasoning must coordinate multiple constraints or evidence pieces.

Inference efficiency. On DeepSeek-V3, GRACER uses 12.3% fewer tokens and 4.8% less runtime than AoT while achieving higher accuracy (Figure 2). Compared with search-heavy ToT and FoT, it improves accuracy by 5.66 and 6.48 percentage points, respectively, with substantially lower token usage and runtime. These results indicate a better accuracy–efficiency trade-off without increasing test-time computation.

Table 1: Main results on six benchmarks under two LLM backbones. Results are reported as accuracy for MATH, GSM8K, BBH, and MMLU-CF, and F1 for HotpotQA and LongBench. Best results are bolded, and second-best results are underlined.

Backbone	Method	MATH	GSM8K	BBH	MMLU-CF	HotpotQA	LongBench	Avg.
DeepSeek-V3	CoT	88.40	92.20	79.60	72.90	68.10	58.80	76.67
	CoT-SC (n=5)	94.80	93.60	84.70	76.20	69.00	58.00	79.38
	Self-Refine	90.00	92.90	81.00	73.20	66.00	59.00	77.02
	AFlow	95.50	94.60	80.50	<u>76.40</u>	72.50	53.50	78.83
	ToT	<u>96.40</u>	94.80	83.00	75.60	72.00	60.57	80.39
	FoT (n=8)	95.20	<u>95.20</u>	82.80	74.70	69.00	60.50	79.57
	AoT	95.90	95.00	<u>86.00</u>	76.20	<u>76.40</u>	<u>65.00</u>	<u>82.42</u>
	GRACER	**97.60**	**97.70**	**88.80**	**82.60**	**81.60**	**68.00**	**86.05**
GPT-4o-mini	CoT	78.30	90.90	78.30	69.60	67.20	57.60	73.65
	CoT-SC (n=5)	81.80	92.00	<u>83.40</u>	71.10	66.20	58.60	75.52
	Self-Refine	78.70	91.70	80.00	69.70	68.30	58.20	74.43
	AFlow	<u>83.00</u>	93.50	76.00	69.50	73.50	61.00	76.08
	ToT	82.50	93.80	74.50	<u>74.50</u>	72.00	64.80	77.02
	FoT (n=8)	82.50	<u>94.00</u>	82.40	70.60	66.70	59.10	75.88
	AoT	81.20	93.20	<u>83.40</u>	72.50	<u>79.00</u>	<u>65.50</u>	<u>79.13</u>
	GRACER	**83.20**	**94.80**	**86.20**	**75.60**	**80.80**	**67.40**	**81.33**

Table 2: Call-matched comparison on DeepSeek-V3. Calls and total tokens are averaged per instance. Metrics follow Table 1; token usage is measured, not forced to be identical.

Method	Calls	MATH	GSM8K	BBH	MMLU-CF	HotpotQA	LongBench	Avg.	Tokens (K)
AoT	11.11	95.90	95.00	86.00	76.20	76.40	65.00	82.42	26.50
AoT-CallMatch	8.72	96.30	95.50	86.80	77.40	76.60	66.00	83.10	23.15
GRACER	8.72	97.60	97.70	88.80	82.60	81.60	68.00	**86.05**	23.20

[1]GPT-4o mini model information is available from the OpenAI blog.

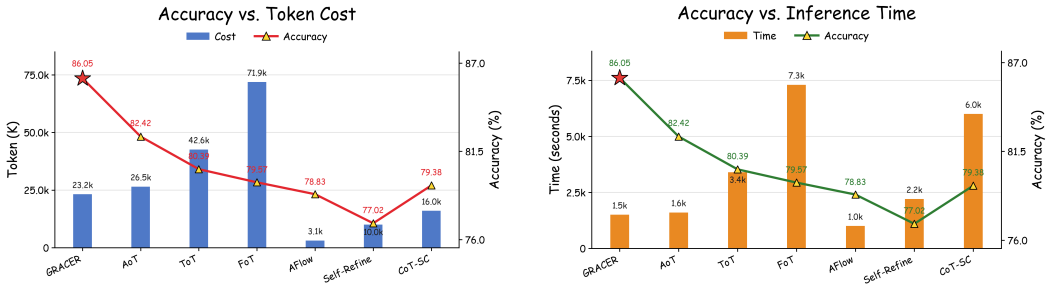


Figure 2: Accuracy–efficiency trade-off across reasoning methods. The two panels plot average accuracy against token usage and runtime, respectively. The horizontal axis lists different methods, and the vertical axis uses evenly spaced reference lines.

Call-matched control. AoT-CallMatch receives the same per-instance call allocation as GRACER, with generic repair and rewriting but no credit, entropy, selected-chain, node-selection, or action-label guidance. Under DeepSeek-V3, GRACER outperforms this control on all six benchmarks by 2.95 points on average, using identical calls (8.72) and nearly equal tokens (23.20K versus 23.15K per instance; Table 2). Generic revision improves AoT by only 0.68 points. This supports a benefit from graph-derived guidance beyond the revision budget; token usage is measured rather than forced to match exactly.

Stage-wise resource accounting. We separately profile 400 sampled instances, recording every model call, input/output token count, and stage latency. GRACER uses 8.72 calls per instance versus AoT's 11.11. Repair and compression account for 1.48 calls, while contraction uses 2.16 calls versus 6.70 for AoT; deterministic graph evaluation requires no LLM calls. Appendix F.4 gives the full breakdown. This 400-instance sample differs from the evaluation sets used for Table 2 and Figure 2; resource comparisons therefore use each experiment's own set of instances.

4.3 Ablation Study

Graph-level components. Table 3(a) shows that all three graph-level components contribute to final-answer performance. Removing repair causes the largest drop (1.18 percentage points), supporting the importance of structural validity before graph evaluation and compression. The losses without compression or critic-guided rewriting suggest that removing redundant structure and using graph-derived guidance to select local rewrites both benefit final contraction.

Critic variants. Table 3(b) further evaluates the internal design of the critic. Replacing the explicit credit–uncertainty critic with an LLM-based node scorer reduces the average score, suggesting that direct prompt-based node scoring is less reliable than using explicit graph-derived signals. Removing structural credit or downstream entropy leads to larger drops, showing that both terms are needed for effective fidelity allocation. Structural credit helps identify answer-supporting regions, while downstream entropy captures branch-sensitive regions where aggressive compression may be risky.

Additional controlled interventions support the assigned roles of both signals: removing high-credit nodes causes larger performance losses, while compressing high-entropy chains is more damaging under length-matched conditions. Full protocols and results are reported in Appendix F.5 (Tables 10 and 11).

4.4 Analysis

Graph regularization and repair behavior. Figure 3 further examines the role-constrained repair stage used as a pre-processing step for graph regularization. The structural-error diagnostics are computed on the first 200 examples from each dataset and then averaged across datasets. The left panel shows that repair reduces the average structural error rate from 32% to 12%, indicating that many malformed dependencies can be corrected before graph evaluation. The right panel breaks down the remaining errors by type, showing that unresolved errors are concentrated in final-linked

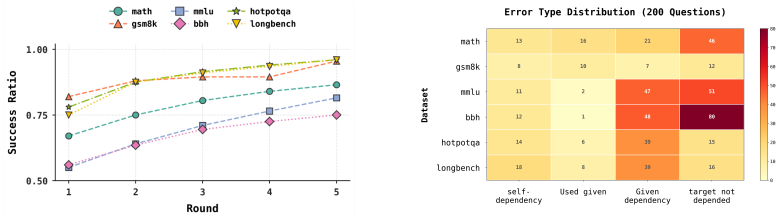


Figure 3: Structural behavior of role-constrained repair. The left panel reports the average error-rate reduction from 32% to 12%, and the right panel breaks down the remaining errors by type, averaged over the first 200 examples from each dataset.

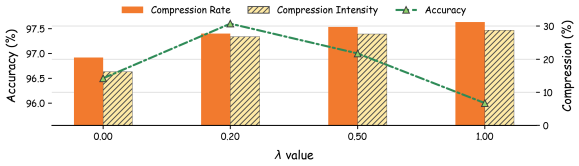


Figure 4: Sensitivity of GRACER to the IB compression coefficient λ. Accuracy reaches its strongest value at $\lambda = 0.20$, while compression rate and compression intensity increase as λ grows. Larger λ values therefore induce a stronger compression–fidelity trade-off.

Table 3: Ablation study on DeepSeek-V3. Panel (a) reports graph-level component ablations; panel (b) reports critic variants averaged across the six benchmarks.

(a) w/o three components

Method	Avg.
Full GRACER	86.05%
w/o Critic-Guided Rewrite	85.09%
w/o Compression	85.27%
w/o Repair	84.87%

(b) critic

Method	Avg.
Full GRACER	86.05%
Model Evaluator	85.40%
Critic w/o Importance	83.83%
Critic w/o Entropy	83.50%

and unused-given violations. These diagnostics support the role of repair as a structural regularizer: it improves graph validity before the critic estimates structural credit and downstream uncertainty.

Hyperparameter behavior. Reverse credit propagation improves over the prior-only setting, with $d = 0.85$ achieving the best average score (Appendix B.2.1). Increasing λ from 0.00 to 0.20 improves accuracy and compression, while larger values increase compression at the cost of accuracy (Figure 4). We therefore use $d = 0.85$ and $\lambda = 0.20$ as defaults.

5 Conclusion

GRACER presents a graph-aware framework for efficient test-time reasoning under the decomposition–contraction paradigm. Instead of directly contracting a generated reasoning graph, GRACER first repairs malformed dependencies, evaluates structural credit and downstream uncertainty, and selectively compresses low-risk local chains before final answer synthesis. Across mathematical reasoning, multiple-choice reasoning, and multi-hop question answering, GRACER achieves consistent gains over a broad set of inference-time reasoning baselines, including direct prompting, sampling-based reasoning, refinement-based reasoning, search-based reasoning, workflow-based reasoning, and the original AoT decomposition–contraction pipeline. These results suggest that test-time reasoning can benefit from optimizing the structure of intermediate reasoning, rather than only increasing the amount of generated reasoning.

AI USE STATEMENT

Generative AI tools, including OpenAI ChatGPT and Codex, Anthropic Claude, and DeepSeek, were used to check and refine mathematical derivations, provide feedback on hypotheses and author-designed experiments, suggest experimental parameters, and assist with initial code development. They were also used for English translation and editing, drafting from author-provided material, manuscript organization, literature search and summarization, figure-code generation, and LaTeX formatting. The GRACER framework and experimental protocols were proposed by the authors; generative AI was not used to generate synthetic data, modify benchmark datasets, or interpret the experimental results. AI-assisted text was cross-checked by the first two authors. AI-assisted code was reviewed by the first three authors through code inspection, end-to-end execution, and consistency checks against the method specification and reported outputs. All references were manually verified. The authors take full responsibility for the final text, claims, code, figures, and results.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 describe the method and evaluation protocol. Appendix A documents the algorithmic workflow, generation and answer-extraction settings, baseline implementations, reported hyperparameters, and dataset-specific evaluation details. Appendix B reports the ablation design and sensitivity analyses, while Appendix F documents the additional controls and resource accounting. We will release the executable pipeline upon acceptance to provide the remaining implementation details and support full reproduction.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 3119–3137, 2024.

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

Maciej Besta, Nils Blach, Aleš Kubíček, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 17682–17690, 2024.

Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. Forest-of-thought: Scaling test-time compute for enhancing LLM reasoning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 4253–4267, 2025.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are more LLM calls all you need? towards the scaling properties of compound AI systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

David Dohan, Aitor Lewkowycz, Jacob Austin, Winnie Xu, Yuhuai Wu, David Bieber, Raphael Gontijo-Lopes, Henryk Michalewski, Rif A. Saurous, Jascha Sohl-Dickstein, Kevin Patrick Murphy, and Charles Sutton. Language model cascades. In *ICML Workshop on Beyond Bayes: Paths Towards Universal Reasoning Systems*, 2022.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Aviral Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning. In *Proceedings of the IJCNLP-AACL*, pp. 305–329, 2023.

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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models. In *ICLR Workshop on Deep Learning for Code*, 2022.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL*, pp. 13003–13051, 2023.

Fengwei Teng, Quan Shi, Zhaoyang Yu, Jiayi Zhang, Yuyu Luo, Chenglin Wu, and Zhijiang Guo. Atom of thoughts for Markov LLM test-time scaling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing*, pp. 368–377, 1999.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

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 *International Conference on Learning Representations (ICLR)*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2369–2380, 2018.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, and Karthik Narasimhan. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)*, 2025.

Qihao Zhao, Yangyu Huang, Tengchao Lv, Lei Cui, Qinzheng Sun, Shaoguang Mao, Xin Zhang, Ying Xin, Qiufeng Yin, Scarlett Li, and Furu Wei. MMLU-CF: A contamination-free multi-task language understanding benchmark. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 13371–13391, 2025.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 62138–62160, 2024.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, and Quoc V. Le. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations (ICLR)*, 2023.

APPENDIX OVERVIEW

This appendix complements the main text by providing additional technical depth, empirical diagnostics, and implementation-level insights that are not included in the main paper. Rather than restating the core method, we focus on clarifying design choices, analyzing model behavior under different conditions, and documenting reproducibility details. The appendix is organized as follows. Appendix A details the full pipeline and experimental setup, including algorithmic structure, implementation considerations, and hyperparameter configurations. Appendix B reports supplementary analyses such as sensitivity and structural behavior. Appendix C provides a detailed case study illustrating how the method operates in practice. Appendices D and E discuss limitations and broader implications. Appendix F provides further methodological clarifications, controlled interventions, and stage-wise resource accounting.

A REPRODUCIBILITY DETAILS

This section documents the algorithmic and experimental details available in the manuscript. While the core design is outlined in Section 3, several aspects of the pipeline require additional clarification to ensure consistent implementation, including how graph operations are instantiated, how decisions are coordinated across modules, and how stochastic components are controlled during evaluation. We therefore present the algorithmic structure, implementation considerations, and experimental configurations in a way that exposes both the procedural flow and the practical constraints that affect reproducibility.

A.1 ALGORITHMIC OVERVIEW

We will release the full executable pipeline upon acceptance. In this appendix, we report the algorithmic structure at the level of operations and decision rules, rather than listing implementation code. Algorithm 1 summarizes the complete test-time procedure.

The subroutines in Algorithm 1 interact through shared graph structure rather than operating independently. For example, the evaluation module relies on a repaired graph to produce stable signals, while the compression module feeds its modifications back into the repair stage to ensure structural consistency. This interdependence means that the pipeline should be viewed as a coordinated process over a single evolving graph representation, rather than as a sequence of isolated transformations.

A.2 EXPERIMENTAL PROTOCOL

This subsection complements the high-level description in Section 4 (Experimental Setup) by documenting the execution details used in the reported experiments.

Evaluation Setup. All experiments are conducted on the standard evaluation splits of each benchmark. To ensure reproducibility, the same random seed is used for all sampling-based methods, and the random number generator state is reset before each run. Generation hyperparameters are fixed across all methods: temperature 0.7, top-p sampling with $p = 0.9$, and a maximum of 1024 tokens per response (except where noted otherwise for specific baselines). Deterministic (greedy) decoding is used for final answer extraction to eliminate any randomness at the evaluation stage. Transient API failures are handled with an exponential backoff retry mechanism: up to 3 attempts, starting with a 1-second delay and doubling each time, capped at 10 seconds.

Answer Extraction. For mathematical reasoning (MATH, GSM8K), answers are extracted by matching patterns that commonly appear in model outputs: content following `####` or inside `\boxed{}`. Extracted strings are normalized to numeric form by removing units and converting to a standard decimal representation. For multichoice tasks (BBH, MMLU-CF), the model output is mapped to the closest valid option letter (A, B, C, D). If multiple letters appear, the first occurrence is taken. For multihop QA (HotpotQA, LongBench), we use the official evaluation scripts (version 1.0 for HotpotQA, the LongBench repository's default for LongBench) and report exact match (EM) or F1 scores as appropriate. The same extraction logic is applied to every method uniformly.

Algorithm 1 GRACER: Full Test-Time Reasoning Pipeline

Require: Question x, optional context c, LLM p_θ, hyperparameters, thresholds τ_a for each action $a \in \{\text{keep}, \text{summarize}, \text{drop}\}$
Ensure: Final answer a^*

function GRACER(x, c)
  $(G, \text{roles}) \leftarrow \textsc{Decompose}(p_\theta, x, c)$
  $G \leftarrow \textsc{Repair}(p_\theta, x, c, G)$ ▷ role-constrained iterative repair
  $(I_v, H_v)_{v \in V} \leftarrow \textsc{Evaluate}(G)$ ▷ reverse credit and path entropy
  $\mathcal{C} \leftarrow \emptyset$
  for each maximal intermediate chain $g = v_1 \to \cdots \to v_k$ with $\deg^+(v_i) = \deg^-(v_i) = 1$ **do**
    $\widetilde{I}(g) \leftarrow \textsc{NormalizeCredit}(I_{v_1}, \ldots, I_{v_k})$
    $\widetilde{H}(g) \leftarrow \textsc{NormalizeEntropy}(H_{v_1}, \ldots, H_{v_k})$
    $\text{action}(g) \leftarrow \text{keep}$
    for $a \in \{\text{drop}, \text{summarize}, \text{keep}\}$ **do**
      $R(g, a) \leftarrow \widetilde{I}(g)(b_a + \kappa_a \widetilde{H}(g)) - \lambda_{\text{len}} \ell(g)/L_{\max}$
      if $R(g, a) \le \tau_a$ **then**
        $\text{action}(g) \leftarrow a$
        break
      end if
    end for
    if $\text{action}(g) \neq \text{keep}$ **then**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{(g, \text{action}(g))\}$
    end if
  end for
  if $\mathcal{C} \neq \emptyset$ **then**
    $G \leftarrow \textsc{Rewrite}(p_\theta, x, c, G, \mathcal{C})$ ▷ local chain rewriting
    $G \leftarrow \textsc{Repair}(p_\theta, x, c, G)$ ▷ re-validate graph structure
  end if
  $q \leftarrow \textsc{Contract}(p_\theta, x, c, G)$
  $a^* \leftarrow p_\theta(q)$
  return a^*
end function

Fairness Control. All methods share identical backbone models (GPT-4o-mini-2024-07-18, DeepSeek-V3), the same set of input instances, and the same answer post-processing routines. No method receives task-specific hyperparameter tuning or additional supervision signals beyond what is documented in its original publication. This strict control is essential because structured reasoning methods could otherwise benefit from hidden advantages, such as larger sampling budgets, access to external verifiers, or hand-crafted task-specific prompts, which would make performance comparisons misleading. For example, a method that internally samples ten reasoning chains would naturally incur higher computational cost; if another method is allowed only a single chain, the comparison would not isolate the algorithmic contribution. By fixing the generation budget (token limits, retries, concurrency) and disabling any method-specific optimizations, our protocol ensures that observed differences in accuracy and efficiency directly reflect the core reasoning and graph-optimization mechanisms.

API and Runtime Considerations. All methods are executed under identical API conditions: the same endpoint (OpenAI or DeepSeek), the same rate limits as provided by each service, and the same request retry logic. Concurrency is not artificially increased for any method, as that could introduce unfair runtime advantages. GRACER uses asynchronous execution with adaptive backoff to respect API constraints while maintaining stability; its effective concurrency is determined dynamically by the number of pending requests and the configured rate limits. Other baselines follow their default execution strategies, typically sequential processing or the concurrency settings recommended in their original implementations. No additional parallelization is added beyond documented defaults. This uniform environment guarantees that token usage, API call counts, and elapsed time are comparable

across methods, and that any observed efficiency gain can be attributed to the reasoning mechanism rather than to differences in request scheduling or parallelization.

A.3 Complexity and Runtime Characteristics

The computational cost of GRACER is dominated by interactions with the language model during decomposition, repair, rewriting, and contraction. On the small, sparse reasoning graphs studied here (typically fewer than ten nodes), reverse credit propagation, path enumeration, and chain identification introduce only lightweight measured overhead. Consequently, the observed runtime is governed primarily by the number of LLM calls and the length of the generated responses.

Cost distribution. The evaluation stage (credit and entropy computation) remains negligible compared with generation on the evaluated graphs. Its path-enumeration cost can nevertheless grow combinatorially with the number of branches, so this empirical observation should not be interpreted as a polynomial worst-case guarantee. The primary source of runtime variability in our experiments comes from iterative repair and rewriting, which depend on model responses and may involve retries. Both processes are bounded by fixed budgets (`MAX_RETRIES` and `LABEL_RETRIES`), preventing unbounded refinement cycles even when the model produces low-quality outputs.

Efficiency gains are preserved. A notable property is that the total cost does not scale linearly with the length of the generated reasoning trace. Instead, it depends on how much of the structure is modified during refinement. When large portions of the reasoning exhibit stable continuation patterns, the compression policy removes redundant content without requiring additional exploration or generation. This selective reduction in information flow means that efficiency improves not by decreasing the number of reasoning steps, but by reducing the amount of information that must be explicitly generated and maintained.

Scaling behavior. As problem complexity increases, the reasoning graph may grow in node count, depth, and branching factor. Reverse credit propagation and chain identification remain inexpensive on sparse graphs, whereas complete target-reaching path enumeration can grow combinatorially in highly branching DAGs. In the evaluated setting, graphs remain small and sparse, and the bounded repair and rewrite budgets keep LLM generation as the dominant cost. Larger graphs may require approximate or truncated path enumeration.

Comparison with search-based baselines. This cost profile distinguishes GRACER from methods such as ToT, FoT, or AFlow, which often allocate additional compute by sampling multiple paths, expanding trees, or executing complex multi-step workflows. In those approaches, efficiency gains typically require limiting the exploration budget, whereas GRACER reduces cost by eliminating redundant structure without sacrificing answer quality. The empirical results in Section 4 confirm that GRACER achieves higher accuracy with fewer tokens and lower runtime than these baselines, supporting the practical significance of its complexity characteristics.

A.4 Module Behavior and Failure Modes

The pipeline exhibits several characteristic behaviors that arise from how structure is generated, revised, and selectively simplified during inference. These behaviors are not explicit in the high-level formulation, but become important for understanding when the method succeeds or fails.

Granularity and structural bias. The initial reasoning structure tends to favor coverage over precision, leading to nodes that combine multiple semantic roles or contain partially specified dependencies. This bias increases recall of intermediate signals while introducing redundancy. Rather than enforcing a strict decomposition format, the pipeline allows such irregularities to persist, with the expectation that later stages will either correct or exploit them. In practice, this design makes the method robust to variable decomposition quality, but it also means that downstream modules must tolerate a wide range of structural noise. Consequently, tasks with extremely noisy or underspecified decompositions remain challenging.

Stability under structural correction. Structural inconsistencies are resolved through localized modifications that prioritize validity over optimality. The resulting graphs are typically coherent but not fully corrected, as missing or ambiguous dependencies are seldom reconstructed. This conservative behavior avoids injecting artificial connections, yet leaves residual imperfections that propagate to later stages. For instance, a 'GIVEN' node that incorrectly depends on another node can be fixed by removing the dependency, but if the correct dependency is entirely absent, the repair module has no basis to add it. Hence, the pipeline can eliminate obvious errors but does not perform deep structural inference.

Competing signals in structural assessment. Nodes that appear central to the reasoning process do not always exhibit stable downstream behavior. High structural influence can coincide with multiple plausible continuations, making such regions sensitive to modification. Conversely, regions with modest influence often follow more predictable patterns. The interaction between these effects introduces a trade-off that shapes how aggressively different parts of the graph can be simplified. When credit and entropy point in opposite directions, for instance a high-credit node with low entropy versus a low-credit node with high entropy, the compression policy defaults to preserving the high-credit node. This approach is generally safe but may leave some low-impact, high-uncertainty regions unoptimized.

Selective simplification and its limits. Local rewriting tends to concentrate on regions with stable continuation patterns, where reducing detail does not disrupt downstream inference. In contrast, regions connected to branching or uncertain structures are less frequently modified, even when they are long or redundant. Failures typically occur when sensitivity is underestimated, leading to the removal of distinctions that later steps rely on. In such cases, the resulting graph remains structurally valid but omits information necessary for correct answer synthesis. A common failure scenario on long-context benchmarks (e.g., LongBench) is when a low-entropy chain is compressed, but the removed intermediate node contained a subtle qualification that later turns out to be critical for disambiguating evidence.

A.5 Implementation Notes

We highlight practical considerations that arise when executing GRACER under real-world constraints, which are not captured by the abstract formulation.

Iteration control and termination. The refinement process involves multiple stages that may be invoked repeatedly. Without explicit limits, certain inputs can trigger prolonged update cycles. Fixed iteration budgets and early stopping criteria are therefore used to guarantee predictable runtime and prevent degenerate loops. These budgets are set conservatively and remain constant across all experiments.

Coupling between pipeline stages. Although each stage operates on a shared graph, their effects are not independent. Updates introduced during rewriting can invalidate structural assumptions made by the evaluation stage, leading to additional correction passes. This interaction creates implicit feedback loops that must be stabilized through periodic validation of the entire graph. A full graph validation is performed after each rewrite, followed by a bounded repair pass, ensuring that downstream modules always receive a coherent structure.

Runtime variability. Execution time depends more on the number of refinement steps than on the initial input size. Inputs that induce repeated corrections or structural adjustments can incur higher latency, while simpler cases complete with minimal overhead. This variability is controlled through bounded retries, constrained scheduling, and asynchronous execution that respects API rate limits without artificially inflating concurrency for any particular method.

Decoding configuration. The randomness of generation affects the structural quality of the initial reasoning graph. To balance diversity and stability, the decoding configuration is chosen such that the model can explore alternative decompositions without introducing excessive noise. The specific temperature and top-p values are kept consistent across all generation steps and are not tuned per task.

Implementation simplicity. Despite the multi-stage design, the method does not require complex coordination mechanisms. Each stage operates using standard model calls and lightweight graph updates, making the overall system straightforward to implement and integrate with existing LLM pipelines. Core graph operations are implemented with efficient numerical libraries, and the pipeline uses asynchronous scheduling to manage API calls without introducing artificial dependencies between modules.

A.6 Interpretation and Rationale

Several components of GRACER draw inspiration from established ideas, but are not used in their canonical form. Instead, they are re-interpreted as operational signals within a unified decision process tailored to test-time reasoning.

Structural credit reflects an intuition similar to influence propagation: importance is determined by how information flows backward through the reasoning graph from answer nodes to their prerequisites. However, the goal is not to produce a globally consistent ranking, but to supply a local, relative signal that guides compression decisions.

Downstream uncertainty captures the observation that not all reasoning steps contribute equally to stability. Some nodes branch into multiple plausible continuations, while others follow nearly deterministic patterns. Distinguishing these cases is more important than precisely estimating the uncertainty values; the entropy of the path distribution serves as a practical risk indicator.

Compression is treated as selective information retention. Rather than optimizing a strict information-theoretic objective, the method implicitly prioritizes regions that are both influential and sensitive to variation, while simplifying those that exhibit predictable behavior. The risk-adjusted surrogate score (Eq. 15) operationalizes this trade-off in a form that does not require access to the true target.

Taken together, these components form a loosely coupled system that approximates a balance between preserving critical structure and reducing unnecessary detail, without relying on explicit global optimization or gold-standard reasoning graphs.

A.7 Baseline Implementation Details

We use prompts only to instantiate the graph operations required by the pipeline, rather than as separate algorithmic contributions. The decomposition prompt asks the model to produce reasoning units with descriptions, answers, dependencies, and roles. The repair prompt receives the current graph and a diagnostic report of structural violations, then returns a corrected graph with preserved semantics. The rewriting prompt is constrained to operate only on candidate chains selected by the IB policy. The contraction prompt converts the repaired and optionally compressed graph into a shorter self-contained query for final answer generation. The LLM-based evaluator used in the ablation receives the repaired graph and returns node-level importance and entropy scores without modifying the graph.

All baselines share the same evaluation instances, answer extraction rules, and task metrics as GRACER. Unless otherwise stated, the backbone models are GPT-4o-mini-2024-07-18 and DeepSeek-V3.

Chain-of-Thought (CoT). We follow standard few-shot CoT prompting (Wei et al., 2022) with eight exemplars per dataset. Temperature is set to 0.7, the maximum generation length is 1024 tokens, and one reasoning path is sampled per instance.

CoT with Self-Consistency (CoT-SC). We implement CoT-SC (Wang et al., 2023) with five independent samples and majority voting. Temperature is set to 0.7, the maximum generation length is 1024 tokens, and sampling is parallelized when applicable.

Self-Refine. We follow the iterative refinement procedure (Madaan et al., 2023) with one refinement step. Hyperparameters follow the default settings in the original implementation, the maximum generation length is set to 4096 tokens, and refinement is applied sequentially.

Tree of Thoughts (ToT). We implement ToT (Yao et al., 2023a) using breadth-first search with depth 2. At each step, three candidate thoughts are generated and evaluated, and the top two states are retained. Temperature is set to 0.7, and the maximum generation length per call is 8192 tokens.

Forest of Thoughts (FoT). We adopt the FoT framework (Bi et al., 2025) with eight independent trees. Each tree follows the same BFS configuration as ToT. We remove early stopping criteria that rely on rule-based validation, because such task-specific signals are not available uniformly in our setting. Temperature is set to 0.7, the maximum generation length is 8192 tokens, and final answers are obtained by majority voting across trees.

AFlow. For AFlow (Zhang et al., 2025), we use the task-specific workflows reported in the original paper. For MATH and GSM8K, we use the workflow optimized for mathematical reasoning. For HotpotQA and LongBench, we use the workflow optimized for multi-hop question answering. Temperature is set to 0.7, the maximum generation length is 8192 tokens, and each workflow is executed once per instance.

Atom of Thoughts (AoT). We implement the original AoT pipeline (Teng et al., 2025) as the closest structured baseline, following its decomposition–contraction paradigm. Hyperparameters follow the official implementation, including a maximum chain length of 3 and LLM-based state selection. GRACER uses the same high-level paradigm but adds role-constrained repair, explicit graph evaluation, and IB-guided graph compression.

A.8 Hyperparameter Configuration

Table 4 summarizes the default hyperparameter configuration used for the reported experiments unless otherwise stated.

Parameter rationale. The damping factor $d = 0.85$ and IB coefficient $\lambda = 0.20$ follow the sensitivity analysis reported in the main text (Section 4) and remain fixed across all experiments. The action penalties (b_a, κ_a) for $\{\texttt{keep}, \texttt{summarize}, \texttt{drop}\}$ were selected via grid search on a validation set, with disjoint search intervals: $b_{\text{keep}}, \kappa_{\text{keep}} \in [0.0, 0.3]$, $b_{\text{summarize}}, \kappa_{\text{summarize}} \in [0.1, 1.0]$, and $b_{\text{drop}}, \kappa_{\text{drop}} \in [0.5, 1.0]$, each with step size 0.01. The risk thresholds are set to $\tau_{\text{drop}} = 0.10$ and $\tau_{\text{keep}} = 0.18$, creating a safety margin where removal is allowed only for chains with negligible risk-adjusted surrogate scores. The same configuration yields consistent gains across DeepSeek-V3 and GPT-4o-mini (Table 1), indicating robustness across model scales.

A.9 Dataset-Specific Evaluation Details

For the MATH dataset, we filter out questions with non-integer or non-decimal answers to ensure consistent evaluation. We evaluate the first 1,000 cases from MATH for efficiency, while assessing the remaining benchmarks in their entirety.

For the cost–accuracy and efficiency analysis in Section 4, baseline methods are evaluated within the same first-1,000-example range when a benchmark contains more than 1,000 examples. LongBench uses its complete local test set. This matched scope ensures that the reported efficiency statistics are comparable across methods.

The hyperparameter sensitivity experiments use the same six benchmarks and the same capped evaluation scope as the main efficiency analysis: up to 1,000 examples per benchmark when the available split contains more than 1,000 examples, and the complete available split otherwise. The reported averages in Appendix B.2 are computed over this matched benchmark set under the DeepSeek-V3 backbone.

B Supplementary Experiments

This section reports supplementary empirical analyses that support the main results, including hyperparameter sensitivity, compression behavior, and structural repair diagnostics.

Table 4: Current hyperparameter configuration used by GRACER.

Category	Hyperparameter	Value
Retry / repair	`MAX_RETRIES`	5
Retry / repair	`LABEL_RETRIES`	5
Chain depth	`ATOM_DEPTH`	3
IB policy	`tau_H`	`auto`
IB policy	`lambda_len`	0.20
IB policy	`eps`	10^{-7}
IB policy	`h_floor`	0.05
IB actions	action set	{`keep`, `summarize`, `drop`}
IB action params	`keep`: (b, κ)	$(0.18,\ 0.18)$
IB action params	`summarize`: (b, κ)	$(0.36,\ 0.42)$
IB action params	`drop`: (b, κ)	$(0.90,\ 0.75)$
Rewrite gate	min chain length	2
Rewrite gate	max average objective	0.20
Action thresholds	drop max score	0.10
Action thresholds	keep min score	0.18

B.1 Ablation Design Rationale

This section clarifies the design and implementation of the ablation variants reported in the main text (Table 3), focusing on what each variant removes and how the remaining pipeline is adjusted.

Module-Level Ablations. Each ablation removes one core component while keeping the rest of the pipeline identical to the full GRACER.

- **w/o Repair**: The role-constrained repair stage is skipped entirely. The raw decomposition graph is fed directly into the evaluation and compression modules, without any structural correction or role refreshing.
- **w/o Compression**: The IB-guided compression policy is disabled. After evaluation, all chains are kept as is (i.e., the `keep` action is forced), and the graph proceeds directly to contraction.
- **w/o Critic-Guided Rewrite**: The evaluation module (credit and entropy) is removed. Instead, a simple LLM-based node scorer (prompted to rate each node's importance on a 1–5 scale) is used to decide rewriting. The same compression policy and thresholds are applied, but the risk score $R(g, a)$ is computed using the LLM scores in place of $\widetilde{I}(g)$ and $\widetilde{H}(g)$.

Evaluator Decomposition. For the critic variants, we isolate the two signals that constitute the evaluator:

- **Critic w/o Importance**: Structural credit I_i is set to a constant (equal for all nodes), so the risk-adjusted surrogate score $R(g, a)$ depends only on downstream entropy $\widetilde{H}(g)$ and length reward.
- **Critic w/o Entropy**: Downstream entropy H_i is set to zero for all nodes, so the risk-adjusted surrogate score depends only on structural credit.

These variants use the same reverse PageRank and IIS implementations as the full critic, but mask the respective signal.

Design Principles. All ablations follow the isolation principle: only one factor is altered at a time, while all other settings (prompts, backbone, hyperparameters, decoding parameters) remain strictly identical to the full system. This ensures that any performance difference can be attributed solely to the removed component. The quantitative results are presented and discussed in the main text (Section 4 and Table 3); we do not repeat them here.

B.2 Hyperparameter Sensitivity

B.2.1 Sensitivity to the Propagation Coefficient

Table 5 reports the effect of the propagation coefficient d in the reverse structural-credit update. When $d = 0.00$, node credit is determined entirely by the structural prior, which removes downstream dependency propagation and leads to lower accuracy. Increasing d improves performance because answer-facing credit can flow back to prerequisite reasoning units. The best result is obtained at $d = 0.85$, while $d = 1.00$ remains strong but is slightly worse, indicating that retaining a small prior component stabilizes the propagated credit.

B.2.2 Sensitivity to the IB Compression Coefficient

We further evaluate the effect of the IB compression coefficient λ on the compression–fidelity trade-off. This sweep is conducted on the same six-benchmark validation scope described in Appendix A.9, using DeepSeek-V3 as the backbone. Figure 4 reports accuracy, compression rate, and compression intensity under $\lambda \in \{0.00, 0.20, 0.50, 1.00\}$. Increasing λ from 0.00 to 0.20 improves accuracy while sharply increasing both compression metrics, indicating that mild compression removes redundant reasoning content without harming answer synthesis. Larger values continue to increase compression rate and intensity, but the accuracy curve declines, suggesting that overly aggressive compression begins to remove information needed for final contraction. Based on this trade-off, we set `lambda_len` to 0.20 in the main experiments.

B.3 Structural Diagnostics

This section provides the structural diagnostics referenced in the main experiments. We report compression behavior to show how graph-aware rewriting changes the generated reasoning structures. Repair diagnostics are reported in Figure 3 in the main text.

B.3.1 Compression Behavior

We first examine how graph compression behaves across datasets. Rather than applying a fixed contraction rate, GRACER activates rewriting only when the dependency structure admits lower-risk local compression. Figure 5 reports the relationship between compression intensity and compression ratio, showing that graph-size reduction varies with dataset-specific structure.

C Mechanism Analysis and Case Study

C.1 Worked Case Study: Graph Evaluation and Active Compression

We analyze a concrete reasoning graph from the following problem:

> *A store sells notebooks for 2 EUR each, pencils for $0.5 each, rulers for $1 each, and pens for $3 each. Alice buys 4 notebooks and 5 pens. What is the total cost in dollars?*

Table 5: Sensitivity of GRACER to the propagation coefficient d. The average score is computed on the same validation setting used for the hyperparameter analysis.

Setting	Avg.
$d = 0.85$	**86.05%**
$d = 0.00$	85.18%
$d = 0.50$	85.62%
$d = 1.00$	85.95%
AoT	82.42%

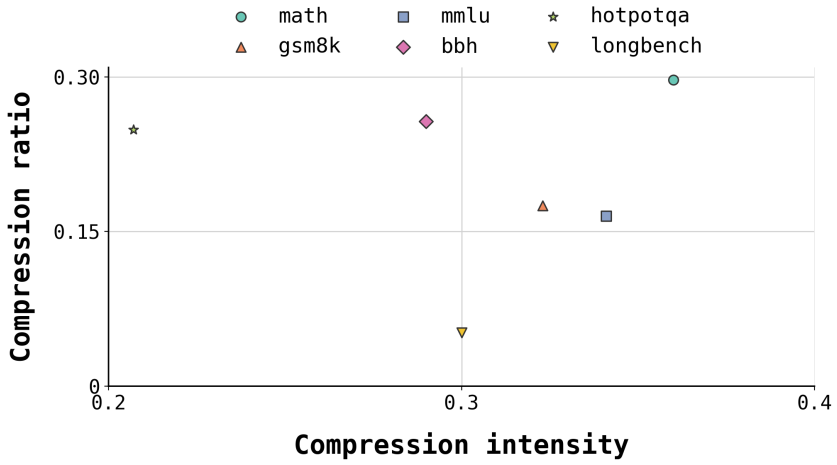


Figure 5: Relationship between compression intensity and compression ratio. The figure shows that graph-size reduction varies with dataset-specific dependency structure rather than following a fixed contraction rate.

This case contains a branching structure with two independent cost calculations and a linear chain of auxiliary unit checks that is compressible. It therefore illustrates both the credit–entropy evaluator and the information-bottleneck compression policy in a single graph instance.

Case graph and node semantics. The case-study graph is shown in Figure 6. The red dashed region marks the local three-node chain selected before compression, and the black solid edges show the dependency structure after compression. The graph contains three GIVEN nodes (v_0, v_1, v_2), two branching intermediate nodes (v_3, v_4), a three-node linear unit-check chain (v_5, v_6, v_7), and one TARGET node (v_8). The selected chain is compressed into a single node $\tilde{v}_0$ that preserves the currency-conversion constraint before the final answer. Node semantics are listed in Table 6.

Importance analysis. For the repaired graph, the critic first computes the structural prior

$$S_i = \frac{\alpha\,\mathrm{outd}(N_i) + \beta\,\mathrm{dep}(N_i) + \gamma\,\log\big(l(N_i)\big)}{\sum_{j\in V}\big[\alpha\,\mathrm{outd}(N_j) + \beta\,\mathrm{dep}(N_j) + \gamma\,\log\big(l(N_j)\big)\big]},$$

and then iterates the reverse personalized PageRank update

$$\mathbf{I}^{(t+1)} = (1-d)\mathbf{S} + dM^{\top}\mathbf{I}^{(t)}, \qquad d = 0.85.$$

In this case, the largest prior masses are assigned to the notebook-price node and the unit-check chain,

$$S_0 = 0.0950, \qquad S_5 = 0.0950,$$

followed by the quantity node and the two cost-calculation nodes,

$$S_2 = 0.0804, \qquad S_3 = S_4 = 0.0804,$$

while the final total-cost node receives the smallest prior,

$$S_8 = 0.0574.$$

The reverse transition matrix is defined by

$$(M^{\top})_{ij} = \frac{\mathbb{I}(v_i \to v_j)}{|\mathrm{Pa}(j)|},$$

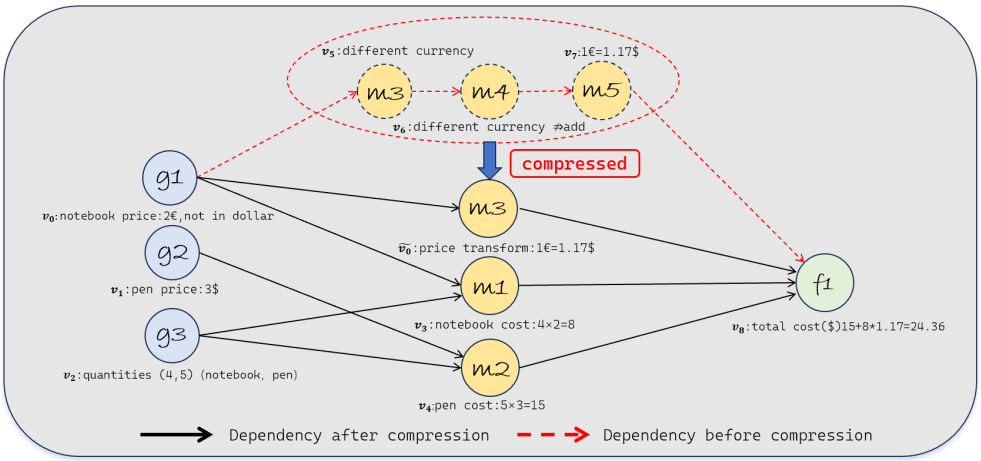


Figure 6: Case-study graph before and after active compression. Red dashed arrows and the dashed oval indicate the pre-compression dependency chain $v_5 \to v_6 \to v_7$; black solid arrows indicate the post-compression graph, where the chain is replaced by the merged node $\tilde{v}_0$.

Table 6: Node semantics for the case-study graph in Figure 6.

Node	Meaning	Node	Meaning	Node	Meaning	Node	Meaning
v_0	notebook price (2 EUR)	v_1	pen price ($3)	v_2	quantities (4, 5)	v_3	notebook cost $4 \times 2 = 8$
v_4	pen cost $5 \times 3 = 15$	v_5	notebook cost in EUR, pen cost in dollars	v_6	different currencies $\not\Rightarrow$ direct addition	v_7	unit check with $1\ \text{EUR} = \$1.17$
v_8	total cost $\$15 + 8 \times 1.17 = \$24.36$						

where $\mathbb{I}(v_i \to v_j) = 1$ if there is an edge from v_i to v_j, and $|\mathrm{Pa}(j)|$ is the in-degree of node v_j. For this example, the dependencies yield the following 9×9 matrix:

$$M^\top = \begin{bmatrix} 0 & 0 & 0 & 0.5 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3333 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3333 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3333 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Starting from $\mathbf{I}^{(0)} = \mathbf{S}$, the iteration converges to

$$\mathbf{I}^\star = [0.2211,\quad 0.0790,\quad 0.1296,\quad 0.0832,\quad 0.0832,\quad 0.1543,\quad 0.1207,\quad 0.0701,\quad 0.0589]^\top.$$

The fixed point shows that the notebook-price node v_0 and the unit-check chain v_5, v_6, v_7 receive nontrivial credit, while the final node v_8 obtains the lowest raw credit due to having no outgoing edges.

Maximum-entropy path uncertainty. For each node i, the same graph is analyzed through the credit-conditioned maximum-entropy path model

$$w_p = \prod_{u \in p} I_u, \qquad P_i(p) = \frac{\exp\left(\sum_{k=1}^{K} \lambda_{i,k} \phi_k(p)\right)}{Z_i}, \qquad H_i = -\sum_{p \in \mathcal{P}_i} P_i(p) \log P_i(p).$$

Consistent with Eqs. (10) and (8), w_p enters through the credit-weighted feature targets used to fit the multipliers; it is not multiplied into the fitted probability a second time. In this graph, node v_2

(quantities) has two distinct descendant paths: one through v_3 and one through v_4. Because the two paths are symmetric in their credit-weighted support, the IIS procedure yields equal probabilities:

$$P_2(p_1) = P_2(p_2) = \frac{1}{2}, \qquad H_2 = \log 2 \approx 0.6931.$$

Node v_0 also has multiple paths, one through v_3 and another through the chain v_5–v_7, but the credits are not balanced; the resulting entropy is $H_0 = 0.0798$. All other nodes lie on unique downstream paths leading to the final answer, so $H_i = 0$ for $i \in \{1, 3, 4, 5, 6, 7, 8\}$. The computed entropy values are:

i	0	1	2	3	4	5	6	7	8
H_i	0.0798	0.0000	0.6931	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Extracted chains and rewrite decision. Under the local-chain rule, the graph yields one raw compressible chain: $[v_5, v_6, v_7]$. After evaluator normalization, the normalized importance values are

$$\widetilde{I}_5 = 0.1543, \quad \widetilde{I}_6 = 0.1207, \quad \widetilde{I}_7 = 0.0701,$$

and $\widetilde{H} = 0$ for all three nodes. Applying the chain-level policy in Eq. (16) with $\lambda_{\text{len}} = 0.20$ selects the `summarize` action for this low-entropy chain. The implementation then checks the rewrite gate,

$$\texttt{best_chain_len} = 3, \qquad \texttt{best_avg_obj} = -0.00174,$$

whose objective is below the safety threshold $\texttt{max_avg_obj} = 0.20$. Hence rewriting is enabled and the three nodes are merged into a single summarized node. This account uses the selected `summarize` action throughout; the drop threshold is not used to justify the merge.

Merged graph and merged node semantics. After compression, nodes v_5, v_6, v_7 are replaced by a new node $\tilde{v}_0$ with description "Unit consistency is satisfied after converting notebook cost into dollars." The post-compression graph is the black-solid-edge structure in Figure 6, and its node meanings are listed in Table 7. The total node count reduces from 9 to 7, while answer equivalence is preserved.

The corresponding merged question, i.e., the compressed state after contraction, is:

> ***A store sells notebooks for $2.34 each and pens for $3 each. Alice buys 4 notebooks and 5 pens. What is the total cost in dollars?***

Discussion. This case illustrates how the graph evaluator assigns low entropy to linear chains, enabling safe compression, while preserving branching structures that are essential for the final answer. The IB-guided policy merges the low-risk unit-check chain without harming correctness, demonstrating structure-aware fidelity allocation.

D Additional Limitations

Although GRACER improves the efficiency of structured reasoning at test time, it remains subject to several limitations. Its performance is inherently sensitive to the quality of the initial decomposition: errors in dependencies or missing reasoning steps may not be fully recoverable through repair and can propagate through evaluation and compression. The evaluator operates without external supervision and relies solely on structural signals, leaving semantically incorrect yet structurally consistent reasoning undetected. Final-answer accuracy does not fully capture semantic drift introduced by graph repair or compression, and a dedicated semantic-faithfulness audit remains future work. Although the policy is designed to preserve high-salience and high-uncertainty components, LLM-based rewriting may still omit or alter answer-critical information, particularly in long or complex reasoning chains. The locality of compression improves stability but can overlook long-range dependencies or fail to coordinate adjustments across distant parts of the graph. The multi-stage pipeline also incurs additional latency due to multiple LLM calls, which may limit applicability in real-time or resource-constrained settings. The repair module does not generate missing nodes or edges from scratch, so severely malformed decompositions may remain partially invalid after repair. Moreover, the refinement process operates under fixed iteration budgets to ensure tractable runtime; consequently, the system may terminate before reaching a fully stable structure when competing structural variants persist.

Table 7: Node semantics for the merged graph in Figure 6.

Node	Meaning
$\tilde{v}_0$	summarized unit-check chain
v_0	notebook price
v_1	pen price
v_2	quantities
v_3	notebook cost
v_4	pen cost
v_8	total cost

E BROADER IMPACTS

GRACER improves both the efficiency and interpretability of LLM reasoning by introducing explicit graph-structured representations of intermediate steps. This structured formulation enhances transparency and auditability, as dependencies between reasoning units can be directly inspected, marking a clear departure from standard chain-based prompting. It also enables finer-grained intervention and control, allowing selective modification or validation of reasoning components, which may benefit high-stakes or safety-critical applications. Compression reduces token usage and computational cost, but overly aggressive rewriting may remove implicitly important reasoning steps and affect correctness or interpretability. The method focuses on structural optimization and does not provide guarantees of factual accuracy or alignment, and should therefore be viewed as complementary to external safety mechanisms such as verification systems or human oversight in sensitive deployment settings. Potential misuse could involve applying compression to hide undesirable reasoning steps, though the explicit graph representation makes such tampering easier to detect than with opaque chain-of-thought traces. We recommend deploying GRACER with logging of compression decisions for auditability.

F ADDITIONAL EVALUATION AND IMPLEMENTATION DETAILS

F.1 REVERSE PAGERANK ON SMALL REASONING GRAPHS

The purpose of reverse PageRank in GRACER is recursive prerequisite-credit propagation. Its relevant property is the dependency update in Eq. (4), rather than an assumption that the graph is large. The prior S_i summarizes local degree, depth, and description information. The propagated term additionally depends on the credit of downstream units. Nodes with the same degree or depth can therefore receive different scores when their descendants differ in target-facing support.

This distinction is explicit on a DAG. Let $A = M^\top$ be the reverse transition matrix in Eq. (3), with zero columns for nodes without parents. For a graph with n nodes, $A^n = 0$, and the stated update has the finite expansion

$$\mathbf{I} = (1-d)\sum_{k=0}^{n-1} d^k A^k \mathbf{S}, \qquad 0 \le d < 1. \tag{17}$$

The $k = 0$ term is the local prior; the higher-order terms carry support through prerequisite paths of increasing length. Such paths remain meaningful in graphs with fewer than ten nodes. This is a structural credit signal, not a probability that a node's reasoning is correct.

On the evaluated small graphs, removing propagation ($d = 0$) yields an average score of 85.18%, compared with 85.62% at $d = 0.50$ and 86.05% at $d = 0.85$. The controlled deletion study in Section 4 further finds larger losses when high-credit nodes are removed. Together, these results support the assigned role of recursive credit. They do not establish that PageRank is the only suitable estimator or that it dominates every simpler graph statistic.

F.2 INTERPRETATION OF THE IB-INSPIRED SURROGATE

Equation (14) states the canonical information-bottleneck principle as motivation. GRACER does not numerically estimate $I(G; Z)$ or $I(Z; Y)$, and Eq. (15) is neither an estimator nor a variational

bound of that objective. At inference time, the true target is unavailable. The implemented decision instead uses graph-derived quantities in an IB-inspired risk–length surrogate: normalized chain length represents compression benefit, while credit and credit-conditioned entropy estimate fidelity-loss risk.

The evaluator signals are coupled in sequence:

$$I_j \;\longrightarrow\; w_p = \prod_{j \in p} I_j \;\longrightarrow\; \bar{\phi}_{i,k} \;\longrightarrow\; P_i(p) \;\longrightarrow\; H_i.$$

Credit measures how much target-facing support is at stake. Entropy measures whether that support is concentrated in a stable continuation or dispersed over several paths. The surrogate combines these quantities with chain length to select a local action. The ablations and interventions evaluate these operational roles; they do not provide a guarantee of information preservation or global optimality.

The actions specify how standalone representations change. `Keep` preserves a step. `Summarize` retains its premises, core operation, and downstream-required result, allowing safe adjacent merges. `Drop` removes only the standalone representation and absorbs necessary content into a retained neighbor; when safe absorption is unavailable, it falls back to summarization. The rewriter is restricted to selected linear intermediate chains, preserves unselected content and external interfaces, and must not introduce new premises. Dependencies, acyclicity, roles, and target placement are revalidated afterward. These constraints support structural consistency but do not certify semantic faithfulness.

F.3 Path Features and IIS Fitting

Each target-reaching path uses four fixed, task-independent features: average normalized description length, average out-degree, average target-related depth, and proportion of intermediate nodes. Equivalently,

$$\phi(p) = \frac{1}{|p|} \sum_{v_j \in p} \Big[\widehat{l}_j,\ \mathrm{outd}(v_j),\ \mathrm{dep}(v_j),\ \mathbf{1}_{\mathrm{int}}(v_j)\Big]^{\top}, \tag{18}$$

where $\widehat{l}_j$ is normalized description length, dep is the target-related depth score, and $\mathbf{1}_{\mathrm{int}}$ indicates an intermediate node. These features require no reference answers or step-level labels.

For each node, fitting follows four operations: enumerate target-reaching paths; compute their features and the credit-induced weights w_p; form the weighted feature targets in Eq. (10); and adjust the multipliers using the IIS update in Eq. (11) until the convergence criterion in Eq. (12) is met. The fitted distribution in Eq. (8) matches the target feature expectations, and Eq. (13) gives its entropy. In the main formulation, w_p enters through the feature targets rather than being multiplied into the fitted probability a second time.

The scaling constant C bounds the total feature sum over paths. In the implementation, path features are normalized before IIS fitting, zero moments are protected by a small numerical floor in the log-ratio update, and optimization terminates when the convergence criterion is met or a fixed iteration limit is reached. A node with a single downstream path has zero entropy, while a node with no target-reaching path is excluded from entropy-based compression decisions. Path enumeration can grow combinatorially in large branching DAGs, so the small overhead observed on the evaluated graphs should not be generalized into a polynomial worst-case guarantee for full enumeration.

F.4 Stage-Wise Resource Accounting

We audit 400 sampled instances across the six benchmarks under DeepSeek-V3, tracing all model calls and input/output tokens. Values below are per-instance averages; tokens are in thousands and latency is in seconds. This audit is distinct from the call-matched comparison in Table 2, and token totals must be compared within the same evaluation subset.

Reverse credit propagation and entropy estimation are deterministic and require no LLM calls. Repair and compression together account for 1.48 calls per instance. Contraction uses 2.16 calls in GRACER versus 6.70 in AoT, consistent with reduced retry work on inconsistent dependencies. This stage association complements the call-matched control rather than independently isolating the cause of the reduction.

Within this audit, decomposition has the largest measured stage latency (0.66 s), followed by contraction (0.44 s) and structural repair (0.43 s). Repair, graph evaluation, and compression together

account for approximately 0.51 s. End-to-end latency is 2.18 s for GRACER and 3.57 s for AoT. Totals and reduction ratios are computed from unrounded logs. Because latency also depends on serving load, scheduling, and retries, calls and input/output tokens provide the more portable resource measures. We do not report monetary cost because no fixed pricing snapshot was recorded for the audit.

F.5 Robustness and Controlled Interventions

Credit intervention: protocol and full results. The additional controls use 1,000 instances each from MATH, GSM8K, BBH, MMLU-CF, and HotpotQA, and all 400 examples of the local LongBench set. For each graph containing an eligible intermediate node, exactly one node is removed using the highest-credit, lowest-credit, or uniform-random rule. Dependency rewiring, prompts, model calls, and decoding remain identical across conditions. Table 10 shows the full benchmark scores: removing high-credit nodes is consistently more damaging than random or low-credit removal. This tests the impact of the selected removal, rather than only an observational correlation between scores and graph statistics.

Entropy intervention: matched compression sensitivity. For each continuous intermediate chain g, define $H(g) = |g|^{-1} \sum_{v_i \in g} H_i$. Within graphs with at least two candidate chains, those at or above the graph-level mean chain entropy form the High-H group; those below form the Low-H group. Group sizes are balanced and chain lengths are matched. Starting from the same graph, we summarize or drop only selected chains while holding other nodes, external dependency interfaces, prompts, model-call budgets, and decoding fixed. Keep is the unmodified control. The selected subset's keep score is 85.58, rather than the full evaluation's 86.05.

High-entropy chains incur larger losses under both actions, with a larger difference for drop. Credit thus predicts removal impact, whereas entropy predicts compression sensitivity in these controls.

Table 8: Stage-wise GRACER resource audit on 400 instances. Latency is measured in the audited serving environment. Graph evaluation requires no LLM calls or tokens; its time rounds to 0.00 s at the displayed precision.

Stage	Calls	Input tokens (K)	Output tokens (K)	Latency (s)
Direct reasoning	1.27	3.63	0.40	0.30
Decomposition	2.81	8.05	0.86	0.66
Structural repair	0.73	0.64	0.46	0.43
Graph evaluation	0.00	0.00	0.00	0.00
Compression	0.75	0.45	0.20	0.08
Contraction	2.16	5.32	0.61	0.44
Final synthesis	1.00	3.53	0.34	0.26
Pipeline total	8.72	21.70	2.90	2.18

Stage-level allocations are rounded profiling values, whereas the pipeline totals are computed directly from the complete request logs. Aggregate comparisons therefore use the pipeline totals. The displayed latency values sum to 2.17 s, while the corresponding unrounded-log total is 2.18 s.

Table 9: Pipeline totals and call allocation under the same 400-instance audit. A dash denotes an absent stage. Label/repair follows the trace categories and does not imply identical repair operators in both methods.

Method	Direct	Decomp.	Label/repair	Compress	Contract	Synthesis	Total calls
AoT	1.11	1.01	1.29	—	6.70	1.00	11.11
GRACER	1.27	2.81	0.73	0.75	2.16	1.00	8.72

Method	Calls	Input tokens (K)	Output tokens (K)	Latency (s)
AoT	11.11	25.60	4.10	3.57
GRACER	8.72	21.70	2.90	2.18
Reported reduction	21.5%	15.3%	29.3%	38.9%

Table 10: Controlled removal of one intermediate node on DeepSeek-V3. Metrics follow Table 1.

Selection	MATH	GSM8K	BBH	MMLU-CF	HotpotQA	LongBench	Avg.
Base	97.60	97.70	88.80	82.60	81.60	68.00	86.05
High-I	95.70	96.20	86.50	80.50	78.50	65.50	83.82
Low-I	97.40	97.40	88.80	82.50	81.00	67.50	85.77
Random	96.70	96.90	88.50	81.60	79.80	67.00	85.08

Table 11: Entropy-conditioned intervention on DeepSeek-V3. Parentheses give the score change from the keep control.

Action	Keep	Low-H	High-H
Summarize	85.58	85.33 (-0.25)	84.18 (-1.40)
Drop	85.58	85.41 (-0.17)	83.25 (-2.33)

Length matching does not establish independence from all other graph properties, and the reported aggregates do not supply confidence intervals or the retained matched-pair count.

Imperfect initial decompositions. Among the 1,200 graphs inspected for structural diagnostics, 68% contain no detected violation. The remaining defects are typically localized role or dependency inconsistencies. Bounded repair rechecks unresolved defects for at most five rounds and reduces the structural-error rate from 32% to 12%. This supports repair of broadly relevant decompositions with localized structural defects. It does not establish robustness to mostly incorrect node contents or missing essential reasoning steps. Such semantic failures remain outside the repair operator's scope.