

From Tokens to Topology: When Large Language Models Meet Graph-Structured Reasoning

Qihang Zhang
New York University
New York, NY, USA
qz2748@nyu.edu

Liang Zeng
Tsinghua University
Beijing, China
zengliangcs@gmail.com

Abstract

Reasoning benchmarks often collapse heterogeneous problem types into aggregate scores, obscuring which structural computations large language models (LLMs) can reliably perform. Existing graph-reasoning benchmarks cover graph traversal, broad graph computation, and graph-language interaction, but do not directly test whether graph-structured reasoning transfers across topological mechanisms under matched conditions. We introduce GraphReason-Bench (GR-Bench), comprising 1,800 programmatically generated and oracle-verified problems across three prompt-length regimes and six task families: local motif enumeration, exact-depth breadth-first search, bounded path enumeration, articulation-point identification, non-induced subgraph matching, and non-monotonic rule execution. All tasks share a text-only executable interface, exact oracles, canonical output normalization, and set-based scoring, enabling controlled cross-mechanism comparison. On the 600-instance main evaluation split across nine proprietary and open-source LLMs, we find mechanism-level dissociation: models that succeed on one structural operation often fail on others, and cross-task performance does not reduce to a stable graph-reasoning ability. Trace-level cases reveal mechanism-specific failures such as premature search termination and loss of intermediate state. An initial sparse-autoencoder probe, supported by cross-model robustness checks, shows that topology-sensitive representations become less separable as structural difficulty increases. Together, these results establish graph-structured reasoning as a measurable domain of LLM evaluation.

1 Introduction

Recent frontier LLMs achieve strong results on established reasoning evaluations (OpenAI, 2024, 2025b; Anthropic, 2024, 2025), including science,

knowledge, and multi-step reasoning benchmarks such as GPQA, MMLU-Pro, and FOLIO (Rein et al., 2023; Wang et al., 2024c; Han et al., 2024). However, it remains unclear whether these evaluations capture the ability to perform explicit multi-step computation over discrete relational structure. In many settings, strong benchmark performance can still rely on semantic priors, external knowledge, or shallow textual cues rather than precise manipulation of structured inputs. Graph-structured reasoning provides a clean setting for isolating this issue: problems are defined by explicit nodes, edges, and formally specified operations, so models must recover and execute structure-sensitive computation directly from the input graph (Veličković et al., 2022; OpenAI, 2025a).

Existing graph-reasoning benchmarks cover important settings such as long-context traversal, broad graph computation, and graph-language interaction (OpenAI, 2025a; Tang et al., 2024; Xu et al., 2025). Yet they do not directly answer a central diagnostic question: whether graph-structured reasoning transfers across qualitatively different topological mechanisms under matched evaluation conditions. A benchmark focused on this question must evaluate different graph operations under a shared executable interface, oracle construction, output normalization, and scoring protocol; otherwise, differences across tasks can be confounded with differences in task formulation, answer type, or evaluation metric.

This distinction matters because graph-structured reasoning is not a single primitive. Recognizing local graph motifs (Milo et al., 2002), maintaining an exact BFS frontier (Cormen et al., 2009), enumerating constrained paths (Yen, 1971), diagnosing articulation structure (Tarjan, 1972), matching a pattern graph to a host graph (Ullmann, 1976), and simulating graph-coupled rule execution (Green et al., 2013) require different

kinds of structural computation. Some emphasize local pattern recognition, while others require exhaustive search, global connectivity analysis, state tracking, or iterative execution. If these mechanisms are collapsed into a single coarse graph-reasoning score, it becomes difficult to tell whether a model has transferable graph-structured competence or only narrow task-specific success.

To address this gap, we introduce **GraphReason-Bench (GR-Bench)**, a mechanism-level diagnostic benchmark for graph-structured reasoning in LLMs. GR-Bench contains 1,800 programmatically generated and oracle-verified problems organized into three prompt-length regimes, with 600 instances in each regime and 100 instances per task family. The benchmark spans six tasks: local motif enumeration, exact-depth breadth-first search, bounded path enumeration, articulation-point identification, non-induced subgraph matching, and non-monotonic rule execution. All tasks use a common executable interface: the model receives a text-serialized graph and a precisely specified operation, and must return the resulting discrete structural object. This design fixes the interface, oracle construction, output normalization, and set-based scoring protocol across task families, enabling controlled comparison across graph-topological mechanisms.

Using the main evaluation split of GR-Bench, we find that even frontier reasoning-enabled LLMs remain far from saturating graph-structured reasoning. More importantly, we observe a mechanism-level dissociation: success on one structural operation does not reliably transfer to others, and model rankings vary substantially across mechanisms even under matched evaluation conditions. We then investigate these differences through targeted case studies, which reveal recurrent mechanism-specific failures including objective mis-specification, premature search termination, loss of intermediate state, and incomplete structural matching.

Finally, we complement behavioral evaluation with an initial representation-level probe based on sparse autoencoders (SAEs) (Templeton et al., 2024; Huben et al., 2024). Using final-layer activations from DeepSeek-R1-Qwen3-8B (DeepSeek-AI, 2025) in a controlled motif-based setting, together with robustness checks on SmolLM3-3B and Llama-3.2-3B-Instruct, we find that topology-sensitive separability weakens sys-

tematically as structural difficulty increases, suggesting a representation-level correlate of the behavioral difficulty observed in graph-structured reasoning.

Taken together, these results position graph-structured reasoning as a distinct and systematically measurable domain of LLM evaluation. GR-Bench fills a gap in current reasoning assessment by making mechanism-level transfer and dissociation directly measurable across multiple graph-topological operations. In summary, our main contributions are threefold:

- We introduce **GraphReason-Bench (GR-Bench)**, a mechanism-level diagnostic benchmark for graph-structured reasoning in LLMs, with 1,800 programmatically generated and oracle-verified problems spanning six representative graph-topological mechanisms under a common executable interface, exact oracle construction, canonical output normalization, and set-based scoring protocol.
- We identify **mechanism-level dissociation** as a stable empirical pattern across nine proprietary and open-source LLMs on the 600-instance main evaluation split: success on one graph-topological operation does not reliably transfer to others, and model rankings vary substantially across mechanisms under matched evaluation conditions.
- We combine trace-level case studies, sampled failure-mode analysis, and an initial sparse-autoencoder probe with cross-model robustness checks to connect behavioral dissociation with recurrent mechanism-specific failures and weakening topology-sensitive separability as structural difficulty increases.

2 Related Work

2.1 Graph-Structured Reasoning Benchmarks

Existing work evaluates graph-structured and algorithmic reasoning through long-range graph dependency modeling (Dwivedi et al., 2022), abstract graph transformation and rule induction (Peltonen et al., 2025), classical algorithm execution (Veličković et al., 2022), text-serialized algorithmic reasoning (Markeeva et al., 2024), and graph traversal in long-context LLM settings (OpenAI, 2025a). Broader reasoning benchmarks provide

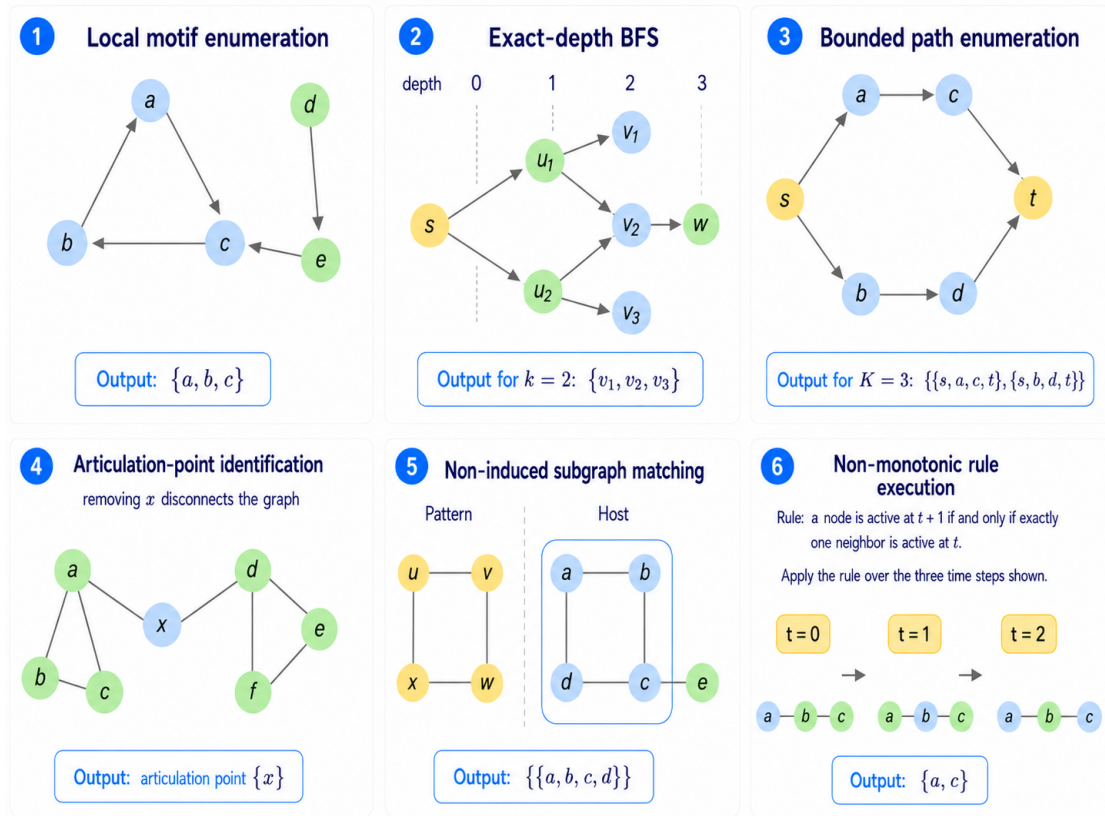


Figure 1: Representative examples from the six task families in GraphReason-Bench: local motif enumeration, exact-depth BFS, bounded path enumeration, articulation-point identification, non-induced subgraph matching, and non-monotonic rule execution. Each panel illustrates the graph structure, the task-specific objective, and the corresponding structured output.

non-graph baselines for formal logic, graduate-level science, and multitask reasoning (Han et al., 2024; Rein et al., 2023; Wang et al., 2024c), while Graph-LLM research studies structured-data reasoning, graph textualization, instruction tuning, graph-language architectures, representation alignment, and graph-oriented evaluation suites (Jiang et al., 2023; Fatemi et al., 2024; Tang et al., 2023; Tan et al., 2024; Chen et al., 2024; Luo et al., 2024b; Plenz and Frank, 2024; Luo et al., 2024a; Wang et al., 2024a; Li et al., 2024b,a; Yuan et al., 2025; Xu et al., 2025; Markowitz et al., 2025; Wang et al., 2024b; Zhang et al., 2024; Tang et al., 2024; Wu et al., 2025). These efforts substantially broaden graph-related evaluation, but they often differ in task family, input interface, supervision setup, graph formalism, or scoring protocol, making them less suitable for controlled comparison across multiple topological mechanisms under a unified benchmark design.

2.2 Behavioral and Representation-Level Analysis of Reasoning

Beyond aggregate scores, prior work studies why models succeed or fail on structured reasoning tasks. Behavioral analyses show that graph reasoning performance can depend on serialization, textualization, encoding choices, instruction design, and the alignment between graph structure and language tokens (OpenAI, 2025a; Jiang et al., 2023; Fatemi et al., 2024; Tang et al., 2023; Luo et al., 2024b; Plenz and Frank, 2024; Luo et al., 2024a; Wang et al., 2024a; Yuan et al., 2025; Xu et al., 2025; Markowitz et al., 2025; Tang et al., 2024), while case studies suggest that graph reasoning errors are often mechanism-specific rather than instances of a single generic reasoning deficit (Dai et al., 2024). At the representation level, sparse autoencoders (SAEs) and related dictionary-learning methods decompose activations into sparse, interpretable features (Bricken et al., 2023; Templeton et al., 2024; Huben et al., 2024; Gao et al., 2025), but existing SAE analyses have primarily focused

on semantic, lexical, or conceptual structure rather than explicit graph-topological computation. It therefore remains unclear how failure modes vary across graph mechanisms, and whether internal representations remain topology-sensitive as structural difficulty increases.

3 GraphReason-Bench

GraphReason-Bench (GR-Bench) is a unified benchmark for graph-structured reasoning in large language models. Each instance consists of a text-serialized graph, a natural-language specification of a graph operation, and an exact discrete target output. The model must execute the specified operation and return a structured object such as a node set, path set, motif list, or set of subgraph matches. All instances are programmatically generated under controlled task-specific regimes and paired with exact symbolic oracles. Tasks are presented through a shared text-only executable interface, and outputs are normalized into canonical forms before scoring.

GR-Bench spans six task families designed to probe distinct topological mechanisms: local motif enumeration (MOTIF), exact-depth breadth-first search (BFS), bounded path enumeration (PATH), articulation-point identification (ART), non-induced subgraph matching (SUBISO), and non-monotonic rule execution (RULE). Together, these tasks cover local versus global structure, static queries versus iterative execution, and within-graph versus cross-graph reasoning. Figure 1 shows one representative instance from each family.

3.1 Task Families

MOTIF asks the model to enumerate local three-node graph motifs, probing local motif recognition and exact relational consistency. In the main split, MOTIF is instantiated as a directed-cycle local motif; in the two scale-up splits, it is instantiated as connected triples satisfying specified incident-degree constraints. BFS asks for all nodes whose shortest-path distance from a source is exactly k , requiring precise frontier maintenance rather than generic reachability. PATH asks for all directed source–target paths of length at most K , stressing exhaustive constrained enumeration. ART asks for all articulation points in an undirected graph, requiring global connectivity reasoning. SUBISO asks for all host-node sets that realize

a non-induced topological copy of a pattern graph, testing cross-graph structural alignment. RULE asks the model to execute threshold-based synchronous updates over a directed graph, requiring intermediate-state tracking across multiple rounds.

3.2 Construction and Evaluation Split

GR-Bench contains 1,800 instances organized into three prompt-length regimes: below 256K characters, 256K–512K characters, and 512K–1M characters. Each regime contains 600 instances, evenly distributed across the six task families. We treat the below-256K regime as the main evaluation split because it supports controlled comparison across proprietary reasoning models, instruction-tuned models, and open-source models under a shared 128K-token context configuration.

The two longer regimes are future-oriented scale-up splits constructed with the same mechanism families, executable interface, and oracle-based scoring protocol. For the local motif family, the longer regimes use scale-appropriate three-node motif queries rather than the exact main-split motif template. They are designed to extend GR-Bench to substantially larger prompt and graph scales as long-context and long-thinking frontier models improve, rather than to define separate benchmark tasks. In this paper, we include these splits as part of the benchmark but do not report their systematic evaluation, since only a limited set of current systems can support them under comparable settings and their full evaluation would require substantially higher inference budgets.

All instances are generated programmatically from task-specific configurations and paired with exact symbolic oracles. Oracle answers are canonicalized into task-specific structured forms before scoring: motif lists for MOTIF, frontier sets for BFS, path sets for PATH, articulation-point sets for ART, host-node match sets for SUBISO, and final active sets for RULE. Across all regimes, oracle answers are non-empty, so models cannot obtain high scores by defaulting to empty outputs. Full benchmark statistics, generation and verification details, and a detailed comparison to existing graph-reasoning benchmarks are provided in Appendix A.

4 Evaluation Setup

4.1 Models

We evaluate GR-Bench on nine proprietary and open-source language models spanning multiple model families and inference configurations.

Proprietary reasoning models. This group includes GPT-5.4, Claude-Opus-4.6, and Gemini-3.1-Pro, evaluated in reasoning-enabled configurations when supported by the provider APIs (OpenAI, 2026; Anthropic, 2026; Google, 2026).

Proprietary instruction-tuned models. We additionally evaluate Claude-Opus-4.6 and Gemini-3.1-Pro in their non-reasoning configurations (Anthropic, 2026; Google, 2026).

Open-source LLMs. We further include DeepSeek-V3.2, MiniMax-M2.5, GLM-5, and Qwen3-Max (DeepSeek, 2025; MiniMax, 2026; Z.AI, 2026; Alibaba Cloud, 2025).

All models are evaluated with the same task inputs, answer-format requirements, parsing rules, and scoring procedure.

4.2 Prompting and Decoding

All reported GR-Bench experiments use the main below-256K split of GR-Bench. This split fits within a 128K-token context window and therefore enables controlled comparison across all evaluated models. The 256K–512K and 512K–1M scale-up splits are included in the benchmark release, but are not used for the main cross-model comparison for the reasons discussed in Section 3. All experiments use task-specific prompt templates rather than few-shot demonstrations sampled from the benchmark instances. For each task, the model receives a text-serialized graph, a natural-language description of the required graph operation, and a strict answer-format specification requiring the final output to appear in the last line as `Final Answer: ....`. Prompt templates are task-specific but structurally standardized. Each template includes short format exemplars covering both empty-set and non-empty outputs to specify the required output form.

We use deterministic decoding with temperature 0. The context window is set to 128K tokens for all reported experiments. For models that expose provider-side reasoning controls, we enable the corresponding reasoning configuration when evaluating reasoning-enabled variants (OpenAI, 2026;

Google, 2026; Alibaba Cloud, 2026). The reasoning budget is capped at approximately 16,384 tokens, which we found to be the highest setting that consistently avoids truncation across all evaluated reasoning-enabled models under the 128K-token context configuration.

Evaluation is performed through a unified batch pipeline with parallel execution, structured-output parsing, and resumable logging. All behavioral experiments are inference-only API evaluations, and we do not train or fine-tune any evaluated LLMs. The main evaluation comprises 600 instances per model configuration, for a total of 5,400 model-instance API evaluations across the nine evaluated configurations; exact parameter counts, GPU-hours, and serving infrastructure for API-served models are not publicly disclosed by the providers.

4.3 Metrics

Our primary metric is macro-averaged F1 over normalized structured outputs. After parsing model outputs, we normalize predictions into canonical forms and compare them against exact oracle answers. Precision, recall, and F1 are computed at the instance level, and macro-F1 is then averaged within each task.

Normalization is structure-aware and task-dependent. Unordered node sets are deduplicated and sorted, path outputs are treated as full node sequences, local motifs are canonicalized according to the task definition, and subgraph matches are represented as host-node sets. When a valid final structured answer cannot be recovered, the prediction is scored as incorrect.

5 Main Results on GraphReason-Bench

Table 1 reports overall macro-F1 and per-task performance on the six GR-Bench mechanisms. Three results stand out. First, GR-Bench is not saturated even for frontier reasoning models. Second, reasoning-enabled configurations yield substantial gains on most tasks over corresponding non-reasoning configurations, but these gains are not uniform across mechanisms. Third, and most importantly, we observe mechanism-level dissociation: models with similar overall scores can exhibit sharply different strengths across graph-topological operations, and success on one mechanism does not reliably transfer to others.

Bootstrap confidence intervals and selected comparison intervals are reported in Appendix B.

The main mechanism-level trends remain stable under resampling, although reasoning-mode gains are not uniform across all mechanisms.

5.1 Overall Results

GR-Bench separates model families clearly. The three strongest results are all obtained by proprietary reasoning models, with Claude-Opus-4.6 thinking achieving the highest overall macro-F1 (0.500), followed by Gemini-3.1-Pro thinking (0.462) and GPT-5.4 thinking (0.266). In contrast, the corresponding non-reasoning proprietary variants are substantially weaker overall, and all four open-source LLMs remain far below the top reasoning-enabled systems. These results show that GR-Bench remains challenging even for frontier models configured for multi-step inference.

5.2 Mechanism-Level Dissociation

The per-task breakdown shows that graph-structured reasoning is not well described by a single homogeneous ability, even under matched input, output, and scoring conditions. Among the strongest proprietary reasoning models, articulation-point identification and MOTIF are relatively strong overall, whereas exact-depth BFS and rule execution are consistently harder. This contrast suggests that BFS and RULE require not only structural recognition, but also exact intermediate-state maintenance across steps.

Bounded path enumeration is especially discriminative. Claude-Opus-4.6 thinking achieves a strong PATH score (0.718), whereas Gemini-3.1-Pro thinking (0.249) and GPT-5.4 thinking (0.317) are much weaker. Subgraph matching shows a similar split: Claude-Opus-4.6 thinking and Gemini-3.1-Pro thinking are both relatively strong on SUBISO, while GPT-5.4 thinking is much weaker. Together, these differences indicate that success on one graph-topological mechanism does not reliably predict success on another. They also suggest that task difficulty is not simply determined by output length or graph size: compact-output tasks such as BFS can remain difficult when correctness depends on maintaining exact intermediate state, while larger-output tasks such as PATH can be solved well when constrained search is preserved.

5.3 Effect of Reasoning Mode

The comparison between reasoning-enabled and non-reasoning proprietary variants shows that GR-

Bench is highly sensitive to explicit reasoning configuration. For Gemini-3.1-Pro, reasoning mode improves all six mechanisms, with especially large increases on MOTIF, ART, BFS, and SUBISO. For Claude-Opus-4.6, the gains are larger on most tasks, particularly MOTIF, BFS, PATH, ART, and SUBISO. The main exception is RULE, where the non-reasoning configuration is stronger than the current reasoning run.

This asymmetry is informative: reasoning mode does not act as a uniform multiplier across graph-structured tasks, but changes the per-task profile in mechanism-dependent ways. This selective improvement suggests that reasoning mode does not simply amplify generic chain-of-thought fluency, but interacts differently with different graph mechanisms.

Appendix C reports a controlled prompt-sensitivity experiment; the broad cross-task ordering is preserved across three prompt variants (mean Spearman’s $\rho = 0.924$).

6 Case Study: Mechanism-Specific Failure Patterns

To complement aggregate results, we analyze two representative trace-level failures. For each case, we anchor the diagnosis at the first sentence where the reasoning trace becomes misaligned with the formal task requirement. These case studies illustrate concrete computational breakdowns, while Appendix F reports a sampled failure-mode analysis over 540 incorrect outputs. A visual summary of the two failures is provided in Appendix E.

Case 1: Frontier-State Collapse in Exact-Depth BFS

Claude Opus 4.6 initially follows the traversal objective, but later replaces exact frontier expansion with a guessed global depth bound, producing an incorrect empty answer at depth 12.

Context (shortened): The task asks for all nodes first discovered at *exactly* depth 12 from a start node. Correctness requires explicit layer-by-layer BFS expansion.

Gold answer: The correct depth-12 frontier contains 26 nodes.

Trace snippet (misaligned): “Depth 6: Nodes like c1c8ce943b, 7ee0bd3b56, b55eaf32d0 ... Depth 7: The most distant nodes like 1219d9fc54 ... After

Model	Overall	MOTIF	BFS	PATH	ART	SUBISO	RULE
GPT-5.4 thinking*	0.266	0.447	0.019	0.317	0.643	0.062	0.108
Claude-Opus-4.6 thinking*	0.500	0.589	0.283	0.718	0.608	0.564	0.238
Gemini-3.1-Pro thinking*	0.462	0.545	0.352	0.249	0.776	0.526	0.326
Claude-Opus-4.6	0.097	0.069	0.000	0.041	0.016	0.000	0.453
Gemini-3.1-Pro	0.099	0.011	0.030	0.033	0.003	0.232	0.285
DeepSeek-V3.2*	0.019	0.031	0.010	0.054	0.008	0.000	0.011
MiniMax M2.5*	0.021	0.044	0.000	0.017	0.014	0.009	0.041
GLM-5*	0.124	0.085	0.022	0.309	0.212	0.025	0.092
Qwen3-Max thinking*	0.027	0.016	0.000	0.079	0.025	0.000	0.040

Table 1: Main results on GR-Bench in terms of macro-F1. In the main split, the MOTIF family is instantiated as local directed-cycle motif enumeration. BFS: exact-depth breadth-first search; PATH: bounded path enumeration; ART: articulation-point identification; SUBISO: non-induced subgraph isomorphism enumeration; RULE: non-monotonic rule execution. * indicates models evaluated with reasoning enabled. Rows are grouped into three blocks: proprietary reasoning models, instruction-tuned models, and open-source models. Bold indicates the best score in each column. Bootstrap confidence intervals and selected comparison intervals are reported in Appendix B.

thorough analysis, this extremely dense graph ... has all reachable nodes discovered by depth 7–8. By depth 12, no new nodes remain to be discovered.”

Failure diagnosis: The error begins at “**all reachable nodes discovered by depth 7–8**”. Instead of maintaining the BFS frontier as an explicit state, the model substitutes a guessed global depth bound. This is a **frontier-state collapse**: the model identifies the right procedure family, but fails to preserve the state invariant that defines exact-depth correctness.

Case 2: Premature Search Closure in Subgraph Isomorphism

Claude Opus 4.6 correctly verifies one valid embedding, but then incorrectly concludes that exhaustive search is complete, missing additional valid embeddings required by the task.

Context (shortened): The task requires enumerating *all* valid subgraph isomorphisms between a query graph and a host graph.

Gold answer: There are **3 valid embeddings**.

Trace snippet (misaligned): “Verification of all 15 edges: All pattern edges confirmed present in G with this mapping. All 9 nodes are distinct. No other valid configurations found after exhaustive search.”

Failure diagnosis: The error begins at “**No other valid configurations found after exhaustive search**”. The model does not fail at local matching; it fails when it turns one correct verification into an unjustified claim of complete-

ness. This is **premature search closure**: local structure is checked correctly, but exhaustive enumeration terminates too early.

These two cases illustrate recurring mechanism-specific failures in GR-Bench. In exact-depth BFS, the model loses a traversal-state invariant; in subgraph-isomorphism enumeration, it fails to maintain search completeness. Together with the sampled failure-mode analysis in Appendix F, this trace-level evidence helps explain why graph-structured reasoning on GR-Bench is not monolithic, but depends on the particular computational mechanism a task requires.

7 Representation-Level Probe of Graph-Structured Reasoning

Behavioral results on GR-Bench show clear differences across mechanisms, but they do not indicate whether increasing structural difficulty is also reflected in internal representations. As an initial representation-level probe, our primary analysis uses sparse autoencoder (SAE) features extracted from DeepSeek-R1-0528-Qwen3-8B, with additional robustness checks on SmoLLM3-3B and Llama-3.2-3B-Instruct. The goal is not to identify a complete causal mechanism, but to test whether topology-sensitive organization in latent space changes systematically as structural difficulty increases.

For the primary DeepSeek probe, we analyze activations at the `op_end` readout using a $64\times$ SAE aligned to layer 35 of the base model. The analysis uses a controlled motif-based shortest-path dataset with five topology classes: path, cycle,

Model / motif	Easy			Medium			Hard		
	Δmean	AUC	KS	Δmean	AUC	KS	Δmean	AUC	KS
<i>DeepSeek-R1-0528-Qwen3-8B: motif-level analysis</i>									
barbell	0.004832	0.8939	0.6364	0.001010	0.6661	0.2434	0.000379	0.5666	0.1072
cycle	0.002586	0.7229	0.3666	0.000935	0.6417	0.2329	0.000295	0.5505	0.0790
path	0.002893	0.7953	0.4692	0.001306	0.7115	0.3285	0.000282	0.5453	0.0650
star	0.002720	0.7181	0.3777	0.001006	0.6527	0.2239	0.000026	0.5064	0.0274
twocomm	0.004423	0.9350	0.7174	0.002088	0.8013	0.4870	0.000385	0.5657	0.0960
Mean	0.003491	0.8130	0.5135	0.001269	0.6947	0.3031	0.000273	0.5469	0.0750

Table 2: Motif-level topology separability across structural difficulty for the primary DeepSeek-R1-0528-Qwen3-8B probe at the op_end readout. For each motif and difficulty, we sample 50 instances and compare 50,000 within-motif pairs with 50,000 between-motif pairs using cosine distance. The Mean row averages the five motifs. Higher Δmean , AUC, and KS indicate stronger topology-sensitive separability.

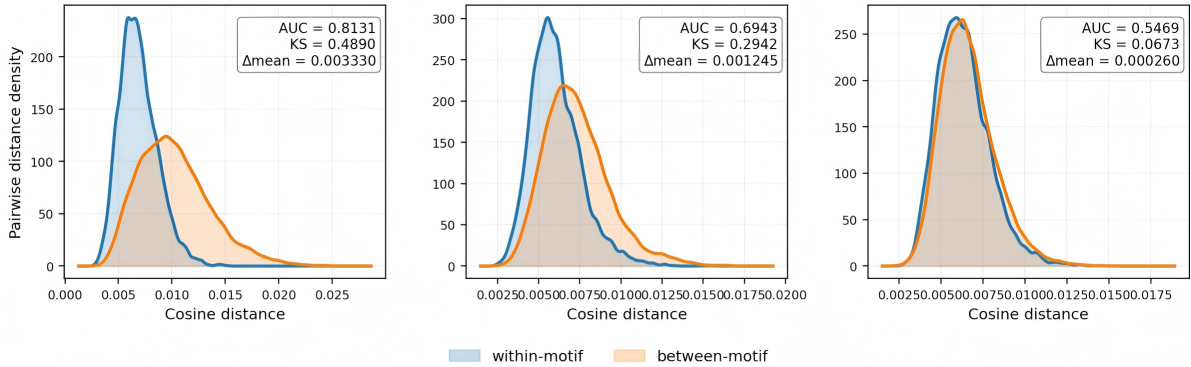


Figure 2: Within-motif and between-motif cosine-distance distributions in SAE latent space across structural difficulty for the primary DeepSeek-R1-0528-Qwen3-8B probe. The distributions increasingly overlap from easy to hard, matching the decline in Table 2.

star, barbell, and twocomm. The task interface is fixed across examples, while difficulty is varied through motif-specific graph construction. Full model, dataset, measurement, and projection details are provided in Appendix G.

We measure topology-sensitive organization by comparing pairwise cosine distances between SAE latent activations. For each difficulty regime, we compare within-motif pairs against between-motif pairs and summarize separation with Δmean , AUC, and the Kolmogorov–Smirnov (KS) statistic.

Table 2 and Figure 2 show a monotonic decline in topology-sensitive separability as structural difficulty increases. Averaged across motifs, AUC drops from 0.8130 in the easy regime to 0.6947 in the medium regime and 0.5469 in the hard regime. The same pattern appears in Δmean , which decreases from 0.003491 to 0.001269 and then 0.000273, and in KS, which falls from 0.5135 to 0.3031 and then 0.0750.

The decline is consistent across all five motifs, although its magnitude varies by class. In the easy regime, twocomm and barbell are the most

separable topologies. In the hard regime, even these motifs deteriorate substantially, and star nearly collapses to chance-level separability with an AUC of 0.5064. Because the task interface is fixed across difficulty regimes, this pattern is associated with increasing structural difficulty rather than changes in prompt wording.

The cross-model checks, summarized in Appendix Table 9, show the same easy-to-hard degradation. For SmolLM3-3B, AUC decreases from 0.721 to 0.703 and 0.513, while KS falls from 0.354 to 0.294 and 0.022. For Llama-3.2-3B-Instruct, AUC decreases from 0.743 to 0.671 and 0.505, and KS from 0.390 to 0.270 and 0.023. The corresponding hard-regime Δmean values also approach zero (0.004 and 0.002, respectively). This consistent pattern strengthens the robustness of the observed difficulty-linked decline.

Taken together, these results suggest that the behavioral difficulty observed in GR-Bench is accompanied by weaker topology-sensitive organization at the analyzed readout position. While this does not fully explain model failures and may omit in-

formation in other layers, positions, or nonlinear directions, the consistent degradation across models and across Δ_{mean} , AUC, and KS provides a concrete representation-level correlate of increasing graph-structural difficulty.

8 Conclusion

We introduced GraphReason-Bench (GR-Bench), a mechanism-level diagnostic benchmark for graph-structured reasoning in large language models. Across nine proprietary and open-source models, GR-Bench shows that graph-structured reasoning remains challenging and mechanism-dependent: success on one graph-topological operation does not reliably transfer to others under matched evaluation conditions. Trace-level case studies and representation analysis further suggest that this dissociation is tied to recurrent mechanism-specific failures and weakening topology-sensitive organization as structural difficulty increases. These findings show that graph-structured reasoning is not well captured by a single aggregate score, and requires evaluation at the level of distinct structural mechanisms.

Limitations

Our analysis is limited in failure-frequency coverage and representation-level breadth across models and tasks. Future work will broaden the benchmark and deepen mechanistic analysis of graph-structured reasoning.

References

- Alibaba Cloud. 2025. [Qwen3-max: Just scale it](#). Alibaba Cloud Community. Accessed 2026-03-17.
- Alibaba Cloud. 2026. [How to use deep thinking models](#). Alibaba Cloud Model Studio Documentation. Accessed 2026-03-17.
- Anthropic. 2024. Introducing claude 3.5 sonnet. <https://www.anthropic.com/news/claude-3-5-sonnet>.
- Anthropic. 2025. Introducing claude 4. <https://www.anthropic.com/news/claude-4>.
- Anthropic. 2026. [Introducing claude opus 4.6](#). Anthropic News. Accessed 2026-03-17.
- Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermy, Tom Conerly, Nicholas L. Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Alex Tamkin, Karina Nguyen, Brayden McLean, and 5 others. 2023. [Towards monosemanticity: Decomposing language models with dictionary learning](#). Transformer Circuits Thread. Accessed 2026-03-17.
- Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. 2024. [GraphWiz: An instruction-following language model for graph computational problems](#). In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 353–364. ACM.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms*, 3rd edition. MIT Press.
- Xinnan Dai, Qihao Wen, Yifei Shen, Hongzhi Wen, Dongsheng Li, Jiliang Tang, and Caihua Shan. 2024. [Revisiting the graph reasoning ability of large language models: Case studies in translation, connectivity and shortest path](#). *arXiv preprint arXiv:2408.09529*.
- DeepSeek. 2025. [Deepseek-v3.2 release](#). DeepSeek API Docs. Accessed 2026-03-17.
- DeepSeek-AI. 2025. [Deepseek-r1-0528-qwen3-8b](#). Hugging Face model card. Accessed 2026-04-17.
- Vijay Prakash Dwivedi, Ladislav Rampásek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. 2022. [Long range graph benchmark](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 25587–25602.
- Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2024. [Talk like a graph: Encoding graphs for large language models](#). In *The Twelfth International Conference on Learning Representations*.
- Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2025. [Scaling and evaluating sparse autoencoders](#). In *The Thirteenth International Conference on Learning Representations*.
- Google. 2026. [Gemini 3.1 pro: A smarter model for your most complex tasks](#). Google Blog. Accessed 2026-03-17.
- Todd J. Green, Shan Shan Huang, Boon Thau Loo, and Wenchao Zhou. 2013. [Datalog and recursive query processing](#). *Foundations and Trends in Databases*, 5(2):105–195.
- Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhen-tong Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alex Wardle-Solano, Hannah Szabo, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, and 16 others. 2024. [Folio: Natural language reasoning with first-order logic](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.
- Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. 2024. [Sparse autoencoders find highly interpretable features in language models](#). In *The Twelfth International Conference on Learning Representations*.
- Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. [Struct-GPT: A general framework for large language model to reason over structured data](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251.
- Xin Li, Weize Chen, Qizhi Chu, Haopeng Li, Zhaojun Sun, Ran Li, Chen Qian, Yiwei Wei, Zhiyuan Liu, Chuan Shi, Maosong Sun, and Cheng Yang. 2024a. [Can large language models analyze graphs like professionals? a benchmark, datasets and models](#). *arXiv preprint arXiv:2409.19667*.
- Yuhan Li, Peisong Wang, Xiao Zhu, Aochuan Chen, Haiyun Jiang, Deng Cai, Victor Wai Kin Chan, and Jia Li. 2024b. [GLBench: A comprehensive benchmark for graph with large language models](#). *Advances in Neural Information Processing Systems*, 37:42349–42368.
- Haitong Luo, Xuying Meng, Suhang Wang, Tianxiang Zhao, Fali Wang, Hanyun Cao, and Yujun Zhang. 2024a. [Enhance graph alignment for large language models](#). *arXiv preprint arXiv:2410.11370*.
- Zihan Luo, Xiran Song, Hong Huang, Jianxun Lian, Chenhao Zhang, Jinqi Jiang, Xing Xie, and Hai Jin. 2024b. [Graphinstruct: Empowering large language models with graph understanding and reasoning capability](#). *arXiv preprint arXiv:2403.04483*.
- Larisa Markeeva, Sean McLeish, Borja Ibarz, Wilfried Bounsi, Olga Kozlova, Alex Vitvitskiy, Charles Blundell, Tom Goldstein, Avi Schwarzschild, and

- Petar Veličković. 2024. [The CLRS-text algorithmic reasoning language benchmark](#). *arXiv preprint arXiv:2406.04229*.
- Elan Markowitz, Krupa Galiya, Greg Ver Steeg, and Aram Galstyan. 2025. [KG-LLM-Bench: A scalable benchmark for evaluating llm reasoning on textualized knowledge graphs](#). *arXiv preprint arXiv:2504.07087*.
- Ron Milo, Shalev Shen-Orr, Shai Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. [Network motifs: Simple building blocks of complex networks](#). *Science*, 298(5594):824–827.
- MiniMax. 2026. [Minimax m2.5: Built for real-world productivity](#). MiniMax News. Accessed 2026-03-17.
- OpenAI. 2024. [Learning to reason with llms](#). <https://openai.com/index/learning-to-reason-with-llms/>.
- OpenAI. 2025a. [Introducing GPT-4.1 in the API](#). OpenAI Blog. Includes GraphWalks benchmark results; accessed 2026-03-17.
- OpenAI. 2025b. [Introducing openai o3 and o4-mini](#). <https://openai.com/index/introducing-o3-and-o4-mini/>.
- OpenAI. 2026. [Introducing GPT-5.4](#). OpenAI Blog. Accessed 2026-03-17.
- Saku Peltonen, August Bøgh Rønberg, Andreas Plesner, and Roger Wattenhofer. 2025. [GraphARC: A comprehensive benchmark for graph-based abstract reasoning](#). In *Proceedings of the Foundations of Reasoning in Language Models Workshop (FoRLM)*.
- Moritz Plenz and Anette Frank. 2024. [Graph language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4477–4494, Bangkok, Thailand. Association for Computational Linguistics.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. [GPQA: A graduate-level google-proof Q&A benchmark](#). *arXiv preprint arXiv:2311.12022*.
- Yanchao Tan, Hang Lv, Xinyi Huang, Jiawei Zhang, Shiping Wang, and Carl Yang. 2024. [MuseGraph: Graph-oriented instruction tuning of large language models for generic graph mining](#). *arXiv preprint arXiv:2403.04780*.
- Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2023. [GraphGPT: Graph instruction tuning for large language models](#). *arXiv preprint arXiv:2310.13023*.
- Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. 2024. [GraphArena: Evaluating and exploring large language models on graph computation](#). *arXiv preprint arXiv:2407.00379*.
- Robert Endre Tarjan. 1972. [Depth-first search and linear graph algorithms](#). *SIAM Journal on Computing*, 1(2):146–160.
- Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L. Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermy, and 3 others. 2024. [Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet](#). Transformer Circuits Thread. Accessed 2026-03-17.
- Julian R. Ullmann. 1976. [An algorithm for subgraph isomorphism](#). *Journal of the ACM*, 23(1):31–42.
- Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dasheskiy, Raia Hadsell, and Charles Blundell. 2022. [The CLRS algorithmic reasoning benchmark](#). In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 22084–22102. PMLR.
- Duo Wang, Yuan Zuo, Fengzhi Li, and Junjie Wu. 2024a. [LLMs as zero-shot graph learners: Alignment of GNN representations with LLM token embeddings](#). *arXiv preprint arXiv:2408.14512*.
- Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2024b. [Can language models solve graph problems in natural language?](#) *Advances in Neural Information Processing Systems*, 37.
- Yubo Wang, Xueguang Ma, Ge Zhang, Yu Ni, Abhiram Chandrachud, Wenhui Guo, and Xiang Ren. 2024c. [MMLU-Pro: A more robust and challenging multi-task language understanding benchmark](#). *arXiv preprint arXiv:2406.01574*.
- Qiming Wu, Zichen Chen, Will Corcoran, Misha Sra, and Ambuj K. Singh. 2025. [GraphEval36K: Benchmarking coding and reasoning capabilities of large language models on graph datasets](#). In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 8095–8117.
- Hao Xu, Xiangru Jian, Xinjian Zhao, Wei Pang, Chao Zhang, Suyuchen Wang, Qixin Zhang, Zhengyuan Dong, Joao Monteiro, Bang Liu, Qiuzhuang Sun, and Tianshu Yu. 2025. [GraphOmni: A comprehensive and extensible benchmark framework for large language models on graph-theoretic tasks](#). *arXiv preprint arXiv:2504.12764*. ArXiv version; page notes publication at ICLR 2026.
- Jin Y. Yen. 1971. [Finding the \$K\$ shortest loopless paths in a network](#). *Management Science*, 17(11):712–716.
- Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. 2025. [GraCoRe: Benchmarking graph comprehension and](#)

complex reasoning in large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7925–7948.

Z.AI. 2026. [GLM-5: From vibe coding to agentic engineering](#). Z.AI Blog. Accessed 2026-03-17.

Zeyang Zhang, Xin Wang, Ziwei Zhang, Haoyang Li, Yijian Qin, Simin Wu, and Wenwu Zhu. 2024. [LLM4DyG: Can large language models solve problems on dynamic graphs?](#) In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Regime	Family	#Inst.	Prompt chars	Avg. edges	Output items	Answer chars
Main	MOTIF	100	19,545 / 19,545	730.0	51.4 / 60	2,106 / 2,461
Main	BFS	100	76,256 / 76,306	3,000.0	17.2 / 34	225 / 443
Main	PATH	100	20,112 / 24,452	743.4	621.9 / 757	84,954 / 109,776
Main	ART	100	22,903 / 29,892	856.4	22.3 / 29	291 / 378
Main	SUBISO	100	8,612 / 8,653	165.9	3.0 / 7	518 / 1,212
Main	RULE	100	70,982 / 88,419	2,741.0	73.2 / 113	953 / 1,470
256K–512K	MOTIF	100	317,995 / 319,084	12,647.4	32.0 / 32	1,313 / 1,313
256K–512K	BFS	100	360,752 / 409,767	14,334.4	24.8 / 28	323 / 365
256K–512K	PATH	100	488,859 / 511,788	19,468.8	99.6 / 150	16,668 / 25,625
256K–512K	ART	100	271,365 / 299,192	17,950.9	15.7 / 16	126 / 129
256K–512K	SUBISO	100	306,203 / 308,674	12,113.2	6.0 / 6	1,261 / 1,261
256K–512K	RULE	100	320,487 / 388,167	12,636.2	288.7 / 525	3,754 / 6,826
512K–1M	MOTIF	100	690,069 / 691,884	27,530.4	64.0 / 64	2,625 / 2,625
512K–1M	BFS	100	774,243 / 879,371	30,873.9	37.8 / 42	492 / 547
512K–1M	PATH	100	966,793 / 999,413	38,586.2	98.4 / 150	16,442 / 25,131
512K–1M	ART	100	653,762 / 734,237	43,444.0	23.6 / 26	190 / 209
512K–1M	SUBISO	100	747,607 / 754,924	29,769.3	6.0 / 6	1,261 / 1,261
512K–1M	RULE	100	653,362 / 746,353	25,838.6	598.3 / 1,035	7,779 / 13,456

Table 3: Benchmark statistics for GR-Bench across the main evaluation split and two future-oriented scale-up splits. Prompt chars, output items, and answer chars are reported as mean / max. Avg. edges reports the average number of serialized graph edges; for SUBISO, this refers to the host graph.

A Additional Benchmark Details

A.1 Benchmark Statistics

Table 3 summarizes the benchmark statistics for the main evaluation split and the two future-oriented scale-up splits. All three splits use the same six mechanism families, executable interface, and oracle-based scoring protocol, but instantiate the mechanisms at different prompt and graph scales. The main split is used for the controlled cross-model experiments reported in this paper, while the 256K–512K and 512K–1M splits extend the same benchmark design to larger structural scales.

GR-Bench scales along multiple axes rather than a single notion of difficulty. Prompt length increases from tens of thousands of characters in the main split to hundreds of thousands or nearly one million characters in the larger-scale splits. Graph scale also grows substantially: average edge counts increase from hundreds or a few thousand in the main split to tens of thousands in the 512K–1M split. Output scale varies independently of prompt length. PATH is the largest structured-output task in the main split, averaging 621.9 paths and 84,954

answer characters per instance, while RULE becomes the largest output task in the 512K–1M split, averaging 598.3 active nodes. SUBISO has fewer output items but requires cross-graph alignment; its host graphs scale from 68 nodes and 165.9 edges in the main split to 16,000 nodes and 29,769.3 edges in the 512K–1M split, while the pattern graph grows from 9 nodes and 15 edges to 16 nodes and 41 edges. For the MOTIF family, the main split uses directed-cycle local motifs, whereas the scale-up splits use connected degree-constrained triples.

These statistics reflect the intended design of GR-Bench: the main split supports controlled comparison across current model families, while the two larger-scale splits preserve the same mechanisms at larger prompt and graph scales for future frontier-model evaluation.

A.2 Generation, Verification, and Reproducibility

GR-Bench instances are generated programmatically from task-specific generation scripts rather than manually authored. A master generation

script coordinates six family-specific generators, one for each task family, using a fixed default seed. The main split contains a frozen below-256K benchmark, while the 256K–512K and 512K–1M splits are independently generated from larger-scale configurations. The scale-up splits are not produced by padding or duplicating main-split instances; their length comes from native graph and task structure, such as larger host graphs, additional decoys, deeper BFS constructions, larger rule-execution graphs, and more near-miss paths.

Generation is controlled by task-specific parameters. MOTIF controls the number and type of local three-node structures, decoy degree patterns, hubs, and noise edges. BFS controls target depths, frontier sizes, distractors, noise nodes, and auxiliary edges. PATH controls graph size, invalid-chain prefixes, path budget, and length constraint. ART controls the number of components, lobes, lobe sizes, detours, and target articulation counts. SUBISO controls host-graph size, decoy blocks, pattern size, and pattern-edge count. RULE controls graph size, listed-edge count, and the number of synchronous update rounds. These parameters allow the same task mechanisms to be instantiated at different prompt and graph scales.

Each generated instance is paired with an exact symbolic oracle. MOTIF uses local structured motif enumeration, BFS computes exact-depth frontier sets, PATH enumerates bounded source–target paths under the task-specific filters, ART computes articulation points under the specified constraints, SUBISO constructs and verifies non-induced subgraph matches, and RULE simulates the synchronous non-monotone threshold dynamics for the specified number of rounds. Gold answers are computed from the generated graph by these oracles, not from model outputs or manual annotation.

We apply validity and consistency checks during construction. Retained instances must satisfy task-specific structural constraints, fall within the target prompt-length bucket, and have non-empty oracle answers. The prompt graph and oracle graph are checked for consistency, and gold answers must have the expected structured type: motif triples, node sets, path sets, articulation-point sets, subgraph match sets, or active-node sets. Across the current splits, the empty-output rate is zero, so models cannot obtain high scores by defaulting to an empty answer.

Outputs are canonicalized before release and

scoring. Duplicate answer objects are removed, unordered sets are sorted, paths are represented as full node sequences, and subgraph matches are normalized into canonical host-node-set representations. The same normalization logic is applied to oracle answers and parsed model predictions, which supports precision, recall, macro-F1, and exact-match evaluation. Unparseable or structurally invalid model outputs are assigned zero score for the corresponding instance.

The benchmark is designed to reduce data-leakage risk. All graphs are synthetic, node identifiers are random-looking strings without semantic content, and answers depend on the exact generated edge list and task operation. This does not eliminate all leakage risk after release, but it reduces dependence on memorized natural-language facts, named entities, or public real-world graph datasets.

The GR-Bench data release¹ contains the 1,800 benchmark instances, split assignments, prompts, reference answers, and per-instance metadata, including task family, prompt length, graph statistics, answer size, and generation configuration. The accompanying code release² provides the evaluation parser and the canonicalization, scoring, resumption, and offline re-scoring utilities. The data and code are released under CC BY 4.0 and MIT, respectively.

A.3 Overlap Analysis with Closely Related Benchmarks

GR-Bench is closest to recent LLM graph-reasoning benchmarks that evaluate text-serialized graph traversal, broad graph computation, or general graph-theoretic reasoning. Table 4 focuses on the three most direct points of overlap: GraphWalks, GraphArena, and GraphOmni. Other graph and structured-reasoning benchmarks cover important but less directly comparable settings, including graph representation learning, algorithm-trace prediction, graph transformation, graph coding, and GraphLLM method evaluation (Dwivedi et al., 2022; Veličković et al., 2022; Peltonen et al., 2025; Markeeva et al., 2024; Li et al., 2024b,a; Yuan et al., 2025; Markowitz et al., 2025; Wang et al., 2024b; Zhang et al., 2024; Wu et al., 2025).

The purpose of this comparison is not to claim that GR-Bench subsumes prior benchmarks. In-

¹<https://huggingface.co/datasets/topostruct/GR-Bench>

²<https://github.com/Ryou-code/GR-Bench>

Benchmark	Closest overlap with GR-Bench	Task / mechanism coverage	Interface and output	Scoring / evaluation target
GraphWalks (OpenAI, 2025a)	Text-serialized graph inputs and traversal-style graph reasoning.	Closest to BFS-like traversal, but does not compare a broad set of distinct topological mechanisms.	Text edge-list interface with traversal-oriented answers.	Oracle-checkable traversal performance; mainly tests long-context graph following rather than cross-mechanism transfer.
GraphArena (Tang et al., 2024)	Broad graph-computation problems for LLMs.	Covers polynomial-time and NP-complete graph tasks, but with heterogeneous problem formulations and solver objectives.	Natural-language problem-solving interface; answer format varies by task.	Solver-oriented evaluation, including correctness, feasibility, optimality/suboptimality, and hallucination behavior.
GraphOmni (Xu et al., 2025)	General graph-theoretic LLM evaluation.	Covers diverse graph tasks and explicitly studies graph types, serialization formats, and prompting schemes.	Interface varies with serialization and prompt design.	Task-specific metrics; useful for broad graph evaluation and interface sensitivity, but not designed to hold interface and scoring fixed while varying only the mechanism.
GR-Bench	Controlled graph-structured reasoning over explicit graph inputs.	Six mechanisms: local motif enumeration, exact-depth BFS, bounded path enumeration, articulation-point identification, non-induced sub-graph matching, and non-monotonic rule execution.	Shared text-only executable interface; structured set-valued outputs normalized into canonical forms.	Exact symbolic oracles and macro-F1 over normalized outputs, enabling mechanism-level comparison under matched input, output, and scoring conditions.

Table 4: Overlap analysis between GR-Bench and the closest existing LLM graph-reasoning benchmarks. GR-Bench overlaps with prior work in graph input modality, graph-computation tasks, and graph-theoretic evaluation, but differs by fixing the interface, output normalization, and scoring protocol while varying the graph-topological mechanism itself.

stead, GR-Bench isolates a different evaluation variable: it fixes the executable text interface, oracle construction, output normalization, and set-based scoring protocol, while varying the graph-topological mechanism being tested.

GR-Bench therefore targets a narrower but more controlled question than broad graph benchmark suites: whether the same LLM can execute distinct graph-topological mechanisms under matched input, output, and scoring conditions. This design reduces confounds from heterogeneous serialization schemes, prompt formats, answer types, and task-specific metrics, making cross-task differences more directly interpretable as mechanism-level variation.

B Uncertainty Analysis

We estimate uncertainty over instance-level variation using nonparametric bootstrap resampling. For each model and task family, we resample the 100 instances with replacement and recompute the mean instance-level F1. For overall macro-F1, we use stratified bootstrap resampling within each task

family and average the six bootstrapped task means. Table 5 reports 95% confidence intervals for all model–mechanism scores.

We also report selected bootstrap comparisons in Table 6. Reasoning-mode comparisons are paired over shared benchmark instances. Cross-mechanism contrasts are computed using independent nonparametric resampling within each task family, since instances are not paired across mechanisms.

Model	Overall	MOTIF	BFS	PATH	ART	SUBISO	RULE
GPT-5.4 thinking	0.266 [0.252, 0.280]	0.447 [0.422, 0.472]	0.019 [0.011, 0.028]	0.317 [0.271, 0.366]	0.643 [0.607, 0.677]	0.062 [0.026, 0.103]	0.108 [0.074, 0.145]
Claude-Opus-4.6 thinking	0.500 [0.475, 0.525]	0.589 [0.567, 0.610]	0.283 [0.202, 0.369]	0.718 [0.663, 0.771]	0.608 [0.550, 0.665]	0.564 [0.483, 0.640]	0.238 [0.194, 0.282]
Gemini-3.1-Pro thinking	0.462 [0.437, 0.488]	0.545 [0.486, 0.601]	0.352 [0.264, 0.441]	0.249 [0.190, 0.314]	0.776 [0.728, 0.822]	0.526 [0.458, 0.592]	0.326 [0.284, 0.368]
Claude-Opus-4.6	0.097 [0.092, 0.101]	0.069 [0.059, 0.079]	0.000 [0.000, 0.000]	0.041 [0.037, 0.045]	0.016 [0.003, 0.032]	0.000 [0.000, 0.000]	0.453 [0.431, 0.474]
Gemini-3.1-Pro	0.099 [0.085, 0.114]	0.011 [0.000, 0.029]	0.030 [0.000, 0.069]	0.033 [0.026, 0.040]	0.003 [0.000, 0.010]	0.233 [0.167, 0.300]	0.285 [0.238, 0.331]
DeepSeek-V3.2	0.019 [0.015, 0.024]	0.031 [0.023, 0.040]	0.010 [0.000, 0.026]	0.054 [0.039, 0.070]	0.008 [0.001, 0.018]	0.000 [0.000, 0.000]	0.011 [0.002, 0.023]
MiniMax M2.5	0.021 [0.016, 0.027]	0.044 [0.027, 0.064]	0.000 [0.000, 0.000]	0.017 [0.010, 0.025]	0.014 [0.005, 0.026]	0.009 [0.000, 0.025]	0.041 [0.022, 0.062]
GLM-5	0.124 [0.109, 0.140]	0.085 [0.051, 0.124]	0.022 [0.006, 0.043]	0.309 [0.254, 0.365]	0.212 [0.162, 0.264]	0.025 [0.004, 0.052]	0.092 [0.064, 0.121]
Qwen3-Max thinking	0.027 [0.021, 0.032]	0.016 [0.011, 0.021]	0.000 [0.000, 0.000]	0.079 [0.057, 0.103]	0.025 [0.015, 0.037]	0.000 [0.000, 0.000]	0.040 [0.021, 0.061]

Table 5: Bootstrap 95% confidence intervals for GR-Bench macro-F1. Task-level intervals are computed by non-parametric resampling over the 100 instances within each task family. Overall intervals are computed by stratified bootstrap resampling within task families and averaging the six resulting task means.

Comparison type	Contrast	Task(s)	Δ F1 with 95% CI
Reasoning mode	Claude-Opus-4.6 thinking — Claude-Opus-4.6	MOTIF	0.520 [0.496, 0.543]
Reasoning mode	Claude-Opus-4.6 thinking — Claude-Opus-4.6	BFS	0.283 [0.204, 0.369]
Reasoning mode	Claude-Opus-4.6 thinking — Claude-Opus-4.6	PATH	0.677 [0.620, 0.729]
Reasoning mode	Claude-Opus-4.6 thinking — Claude-Opus-4.6	ART	0.592 [0.530, 0.653]
Reasoning mode	Claude-Opus-4.6 thinking — Claude-Opus-4.6	SUBISO	0.564 [0.485, 0.642]
Reasoning mode	Claude-Opus-4.6 thinking — Claude-Opus-4.6	RULE	-0.215 [-0.261, -0.170]
Reasoning mode	Claude-Opus-4.6 thinking — Claude-Opus-4.6	Overall	0.403 [0.379, 0.428]
Reasoning mode	Gemini-3.1-Pro thinking — Gemini-3.1-Pro	MOTIF	0.534 [0.473, 0.594]
Reasoning mode	Gemini-3.1-Pro thinking — Gemini-3.1-Pro	BFS	0.322 [0.228, 0.416]
Reasoning mode	Gemini-3.1-Pro thinking — Gemini-3.1-Pro	PATH	0.216 [0.155, 0.280]
Reasoning mode	Gemini-3.1-Pro thinking — Gemini-3.1-Pro	ART	0.773 [0.721, 0.821]
Reasoning mode	Gemini-3.1-Pro thinking — Gemini-3.1-Pro	SUBISO	0.293 [0.212, 0.374]
Reasoning mode	Gemini-3.1-Pro thinking — Gemini-3.1-Pro	RULE	0.042 [-0.008, 0.090]
Reasoning mode	Gemini-3.1-Pro thinking — Gemini-3.1-Pro	Overall	0.363 [0.336, 0.391]
Mechanism contrast	Claude-Opus-4.6 thinking: PATH — BFS	PATH-BFS	0.434 [0.330, 0.533]
Mechanism contrast	Claude-Opus-4.6 thinking: SUBISO — BFS	SUBISO-BFS	0.280 [0.160, 0.395]
Mechanism contrast	Gemini-3.1-Pro thinking: ART — PATH	ART-PATH	0.527 [0.449, 0.602]
Mechanism contrast	Gemini-3.1-Pro thinking: ART — RULE	ART-RULE	0.450 [0.384, 0.515]
Mechanism contrast	GPT-5.4 thinking: ART — BFS	ART-BFS	0.624 [0.587, 0.659]

Table 6: Selected bootstrap comparisons for reasoning-mode effects and mechanism-level contrasts. Reasoning-mode comparisons are paired over shared instances. Mechanism contrasts use independent resampling within each task family because instances are not paired across different mechanisms.

C Prompt Sensitivity Analysis

single prompt template.

To test whether prompt-template quality could confound the observed cross-task pattern, we evaluated GPT-5.4 on the same 20 fixed instances from each of the six task families under three prompt variants: instruction paraphrases, removal of format examples, and structured edge (source, target) serialization. Instances, gold answers, decoding settings, reasoning budget, and output constraints were otherwise fixed. Table 7 reports macro-F1 in percent.

Prompt variants affect absolute scores, with the largest shift occurring for RULE, but preserve a broadly similar cross-task ordering (mean $\rho = 0.924$). ART and MOTIF remain the strongest families, while exact-depth BFS remains difficult across all variants. This stability indicates that the observed cross-task pattern is not an artifact of a

Family	Orig.	Paraph.	No ex.	Struct.
MOTIF	52.0	47.6	47.6	48.2
BFS	1.8	3.5	2.2	1.9
PATH	18.7	26.8	21.1	24.1
ART	68.3	67.5	74.1	63.9
SUBISO	14.5	5.8	9.2	14.2
RULE	13.9	1.7	7.3	26.1
Mean	28.2	25.5	26.9	29.7
ρ vs. original	—	0.943	1.000	0.829

Table 7: Prompt-sensitivity results on 20 fixed instances per task family. Columns correspond to the original, paraphrased, no-example, and structured-edge prompts; values are GPT-5.4 macro-F1 (%). The final row gives Spearman’s ρ across the six family-level scores relative to the original prompts.

D Additional Result Visualization

Figure 3 provides a grouped view of the main GR-Bench results by mechanism and model family. This visualization complements Table 1 by making the cross-mechanism heterogeneity more explicit: models with similar overall scores can still show substantially different strengths across MOTIF, BFS, PATH, ART, SUBISO, and RULE.

E Additional Failure Visualization

Figure 4 visualizes the two failure modes analyzed in Section 6: frontier-state collapse in exact-depth BFS and premature search closure in subgraph matching.

F Failure-Mode Frequency Analysis

To assess whether the qualitative case studies reflect broader patterns, we conduct a sampled failure-mode analysis over incorrect outputs from the main evaluation split. We sample 30 incorrect outputs for each task family and each of the three proprietary reasoning models, yielding 540 analyzed errors in total. For each item, we inspect the final answer and, when available, the reasoning trace, and assign one dominant failure mode according to the first identifiable mismatch between the model output or trace and the oracle task requirement.

Table 8 shows that failures are concentrated in structural execution errors rather than malformed outputs. Across all sampled errors, incomplete enumeration is the most common category, followed by premature search termination and intermediate-state loss. The distribution also varies strongly by task family. ART and PATH

failures are dominated by incomplete enumeration, BFS and RULE show many premature-termination and state-loss errors, and SUBISO is split between premature search termination and incomplete enumeration. This pattern supports the interpretation that different GR-Bench mechanisms induce different failure profiles, rather than reflecting a single generic inability to follow graph prompts.

G Representation-Level Analysis Details

This appendix provides the full setup and additional visualization details for the representation-level analysis summarized in Section 7.

G.1 Model Setup

Our primary analysis uses last-layer hidden representations from DeepSeek-R1-0528-Qwen3-8B with a $64\times$ sparse autoencoder (SAE) trained on the openai/graphwalks training split. The SAE is aligned to layer 35 of the base model, whose hidden size is 4096; the corresponding latent dimension is 262,144. Following the SAE architecture used in our checkpoint, hidden states are mapped into latent space by an encoder, passed through a ReLU nonlinearity, and sparsified with TopK selection ($K = 512$). We additionally evaluate the same difficulty-linked separability trend with SmoLLM3-3B and Llama-3.2-3B-Instruct as cross-model robustness checks.

G.2 Dataset Setup

The analysis dataset is a controlled motif-based shortest-path benchmark in which each instance belongs to one of five topology classes: path, cycle, star, barbell, and twocomm. The task interface is fixed across conditions: the model is always asked to predict the shortest-path length between a designated source node and target node. Difficulty is not varied by changing the question format, but by motif-specific graph construction that increases the effective source–target distance. In the multi-motif generator, easy, medium, and hard correspond to increasing depth settings ($d \in \{5, 15, 25\}$); this yields shortest-path targets of d for path, cycle, and star, $d + 1$ for barbell, and $d + 3$ for twocomm.

G.3 Measurement Setup

For each motif–difficulty cell, we sample 50 instances and encode the corresponding op_end activations with the SAE. We then compare within-motif and between-motif pairwise cosine distances

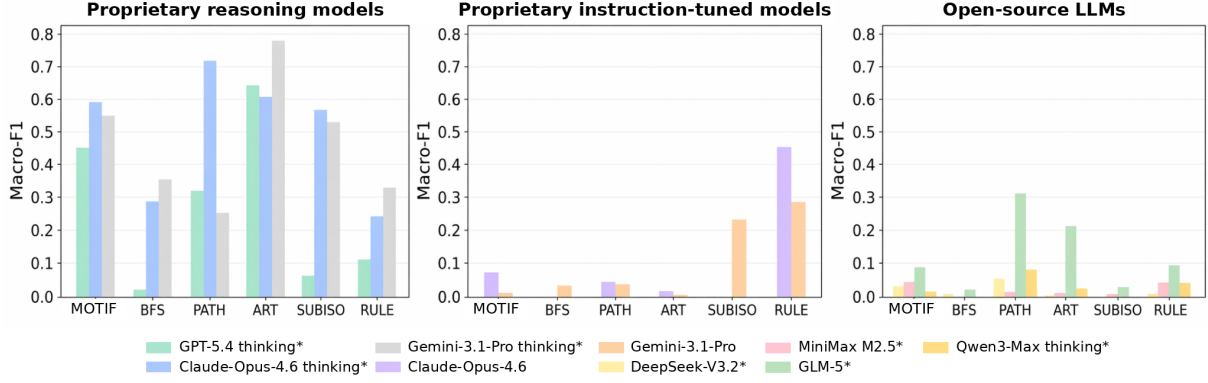


Figure 3: Grouped visualization of GR-Bench performance by mechanism and model family. The chart highlights substantial mechanism-level variation even among models with relatively similar overall scores, reinforcing that graph reasoning performance is not well captured by a single aggregate number alone.

Failure mode	MOTIF	BFS	PATH	ART	SUBISO	RULE	Total
Objective mis-specification	1	11	0	0	3	0	15
Premature search termination	0	58	13	2	46	50	169
Intermediate-state loss	33	13	9	5	2	34	96
Incomplete enumeration	56	5	64	83	38	3	249
Spurious / hallucinated outputs	0	3	0	0	1	2	6
Malformed or unparsable output	0	0	4	0	0	1	5
Other / unclear	0	0	0	0	0	0	0

Table 8: Dominant failure modes over 540 sampled incorrect outputs from the main evaluation split. We sample 30 incorrect outputs for each task family and each of the three proprietary reasoning models, yielding 90 errors per task family. Each output is assigned one dominant failure mode based on the first identifiable mismatch between the model output or trace and the oracle task requirement.

in SAE latent space. For each difficulty regime, we sample 50,000 within-motif pairs and 50,000 between-motif pairs, and summarize separability with three statistics: (i) Δmean , the gap between the mean between-motif distance and the mean within-motif distance; (ii) AUC, interpreted as the probability that a randomly chosen between-motif pair is farther apart than a randomly chosen within-motif pair; and (iii) the Kolmogorov–Smirnov (KS) statistic between the two distance distributions. Higher values indicate stronger topology-sensitive separability.

G.4 Cross-Model Robustness

Model	Difficulty	Δmean	AUC	KS
DeepSeek-8B	Easy	0.003491	0.8130	0.5135
	Medium	0.001269	0.6947	0.3031
	Hard	0.000273	0.5469	0.0750
SmolLM3-3B	Easy	0.068	0.721	0.354
	Medium	0.054	0.703	0.294
	Hard	0.004	0.513	0.022
Llama-3.2-3B	Easy	0.127	0.743	0.390
	Medium	0.096	0.671	0.270
	Hard	0.002	0.505	0.023

Table 9: Cross-model topology-separability summaries. DeepSeek-8B denotes DeepSeek-R1-0528-Qwen3-8B and is the mean across the five motifs in Table 2. Because SAE latent scales are model-specific, Δmean is interpreted within model; AUC and KS are scale-free.

For visualization, we additionally project the full 262,144-dimensional SAE latent activations into two dimensions using a global TruncatedSVD projection fit over all analyzed instances. This projection is used as a qualitative view of motif-level organization; the quantitative separability results are computed from pairwise cosine distances in SAE latent space rather than from the two-dimensional projection.

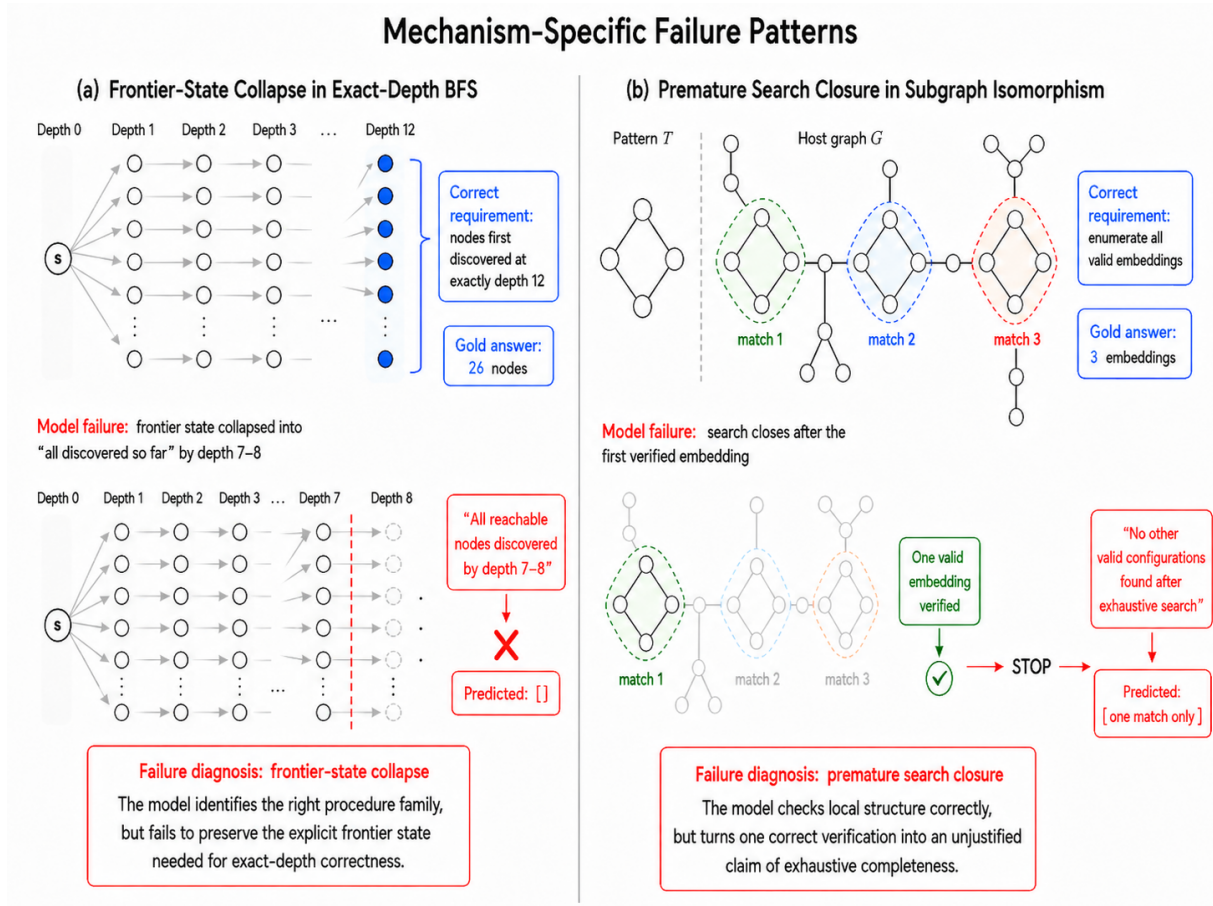


Figure 4: Visual summary of the two mechanism-specific failure patterns analyzed in Section 6. **Left:** In exact-depth BFS, the model initially follows traversal expansion but fails to preserve the explicit frontier state required for exact-depth correctness, and instead substitutes a guessed global depth bound. **Right:** In subgraph isomorphism, the model correctly verifies one valid embedding but terminates enumeration prematurely, confusing local correctness with exhaustive completeness.

G.5 SVD Projection Visualization

Figure 5 shows the global two-dimensional TruncatedSVD projection of the SAE latent activations. The projection provides an illustrative view of motif-level organization across difficulty regimes, complementing the quantitative separability scores and distance-distribution analysis in Section 7.

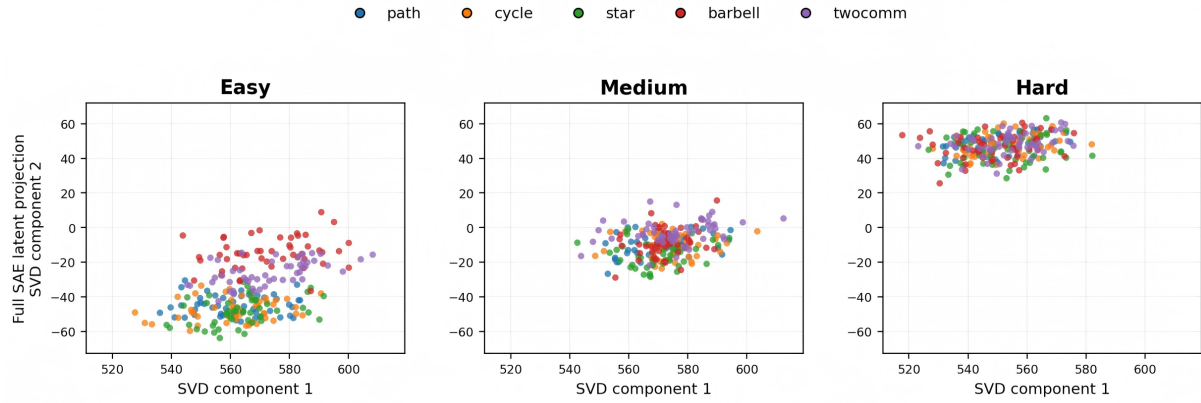


Figure 5: Global two-dimensional TruncatedSVD projection of the full 262,144-dimensional SAE latent activations at the op_end readout. Each point is one instance and colors denote motif families. This projection provides a qualitative view of motif-level organization across difficulty regimes; the quantitative separability results are computed from pairwise cosine distances in the original SAE latent space.

H Prompt Templates

This appendix reports the prompt templates used for all six task families in GraphReason-Bench. Placeholders such as [DIRECTED EDGE LIST], [START_NODE], and [K] are instantiated with example-specific values during data generation.

H.1 Articulation Point Identification

You will be given a graph as a list of
 ↳ undirected edges. All nodes are at least
 ↳ degree 1. You will also get a description
 ↳ of an operation to perform on the graph.
 Your job is to execute the operation on the
 ↳ graph and return the set of nodes that the
 ↳ operation results in. If asked to find
 ↳ articulation points, only return the nodes
 ↳ whose removal (removing the node and all
 ↳ its incident edges) increases the number
 ↳ of connected components of the graph.
 Treat the edge list as a simple undirected
 ↳ graph: ignore self-loops and duplicate
 ↳ edges.

Here is an example:
 <begin example>

The graph has the following edges:
 aaaa -- bbbb
 bbbb -- cccc
 cccc -- aaaa
 cccc -- dddd
 dddd -- eeee

Example 1:
 Operation:
 Find all articulation points.
 Final Answer: [cccc, dddd]

The graph has the following edges:
 p111 -- p222
 p222 -- p333
 p333 -- p111

Example 2:

Operation:
 Find all articulation points.
 Final Answer: []

<end example>

Here is the graph to operate on:
 The graph has the following edges:
 [UNDIRECTED EDGE LIST]

Operation:
 Find all articulation points.

You should immediately return the set of nodes
 ↳ that the operation results in, with no
 ↳ additional text. Return your final answer
 ↳ as a list of nodes in the very last line
 ↳ of your response. The order does not
 ↳ matter. Duplicates will be ignored. For
 ↳ example, if the operation returns the set
 ↳ of nodes [node1, node2, node3], your
 ↳ response should be:
 Final Answer: [node1, node2, node3]
 If the operation returns the empty set, your
 ↳ response should be:
 Final Answer: []

H.2 Breadth-First Search

You will be given a graph as a list of
 ↳ directed edges. All nodes are at least
 ↳ degree 1. You will also get a description
 ↳ of an operation to perform on the graph.
 Your job is to execute the operation on the
 ↳ graph and return the set of nodes that the
 ↳ operation results in. If asked for a
 ↳ breadth-first search (BFS), only return
 ↳ the nodes that are reachable at that
 ↳ depth, do not return the starting node.

Here is an example:
 <begin example>

The graph has the following edges:
 uvwx -> alke
 abcd -> uvwx
 abcd -> efgh

efgh -> uvwx

Example 1:

Operation:

Perform a BFS from node abcd with depth 1.

Final Answer: [uvwx, efgh]

Example 2:

Operation:

Perform a BFS from node alke with depth 1.

Final Answer: []

<end example>

Here is the graph to operate on:

The graph has the following edges:

[DIRECTED EDGE LIST]

Operation:

Perform a BFS from node [START_NODE] with

↪ depth [K].

You should immediately return the set of nodes

↪ that the operation results in, with no

↪ additional text. Return your final answer

↪ as a list of nodes in the very last line

↪ of your response. For example, if the

↪ operation returns the set of nodes [node1,

↪ node2, node3], your response should be:

Final Answer: [node1, node2, node3]

If the operation returns the empty set, your

↪ response should be:

Final Answer: []

H.3 Bounded Path Enumeration

You will be given a graph as a list of

↪ directed edges. All nodes are at least

↪ degree 1. You will also get a description

↪ of an operation to perform on the graph.

Your job is to execute the operation on the

↪ graph and return the set of paths that the

↪ operation results in. If asked to

↪ enumerate paths from node u to node v with

↪ length $\leq K$, only return directed paths

↪ that start at u and end at v and use at

↪ most K edges.

Each path should be represented as a list of

↪ node strings [u, ..., v].

Do not output duplicate paths (treat each path

↪ as a full node sequence).

Deduplicate and sort the list of paths

↪ lexicographically (compare paths as tuples

↪ of node strings).

Here is an example:

<begin example>

The graph has the following edges:

aaaa -> bbbb

aaaa -> cccc

bbbb -> dddd

cccc -> dddd

bbbb -> eeee

eeee -> dddd

Example 1:

Operation:

Enumerate all directed paths from node aaaa to

↪ node dddd with length ≤ 3 .

Final Answer: [[aaaa, bbbb, dddd], [aaaa,
↪ bbbb, eeee, dddd], [aaaa, cccc, dddd]]

Example 2:

Operation:

Enumerate all directed paths from node dddd to

↪ node aaaa with length ≤ 3 .

Final Answer: []

<end example>

Here is the graph to operate on:

The graph has the following edges:

[DIRECTED EDGE LIST]

Operation:

Enumerate all directed paths from node [U] to

↪ node [V] with length $\leq [K]$.

You should immediately return the set of paths

↪ that the operation results in, with no

↪ additional text. Return your final answer

↪ as a list of paths in the very last line

↪ of your response. For example, if the

↪ operation returns the paths [[u, x, v],

↪ [u, y, z, v]], your response should be:

Final Answer: [[u, x, v], [u, y, z, v]]

If the operation returns the empty set, your

↪ response should be:

Final Answer: []

H.4 Rule Execution

You will be given a graph as a list of

↪ directed edges. All nodes are at least

↪ degree 1. You will also get a description

↪ of an operation to perform on the graph.

Your job is to execute the operation on the

↪ graph and return the set of nodes that the

↪ operation results in. If asked for rule

↪ execution, apply the update rule

↪ synchronously for the given number of

↪ steps K: an INACTIVE node becomes ACTIVE

↪ at t+1 if it has at least TH_ON ACTIVE

↪ IN-neighbors at time t; an ACTIVE node

↪ stays ACTIVE at t+1 only if it has at

↪ least TH_KEEP ACTIVE IN-neighbors at time

↪ t, otherwise it becomes INACTIVE.

Here is an example:

<begin example>

The graph has the following edges:

uvwx -> alke

abcd -> uvwx

abcd -> efgh

efgh -> uvwx

Example 1:

Operation:

Execute rule execution with K = 1, TH_ON = 1,

↪ TH_KEEP = 0, and initial ACTIVE set

↪ [abcd].

Final Answer: [abcd, efgh, uvwx]

Example 2:

Operation:

Execute rule execution with K = 2, TH_ON = 1,

↪ TH_KEEP = 0, and initial ACTIVE set

↪ [abcd].

Final Answer: [abcd, alke, efgh, uvwx]

Example 3:
 Operation:
 Execute rule execution with $K = 1$, $TH_ON = 2$,
 $\hookrightarrow TH_KEEP = 0$, and initial ACTIVE set [abcd,
 $\hookrightarrow$ efgh].
 Final Answer: [abcd, efgh, uvwx]

Example 4:
 Operation:
 Execute rule execution with $K = 1$, $TH_ON = 2$,
 $\hookrightarrow TH_KEEP = 1$, and initial ACTIVE set
 $\hookrightarrow$ [alke].
 Final Answer: []

<end example>

Here is the graph to operate on:
 The graph has the following edges:
 [DIRECTED EDGE LIST]

Operation:
 Execute rule execution with $K = [K]$, $TH_ON =$
 $\hookrightarrow [TH_ON]$, $TH_KEEP = [TH_KEEP]$, and initial
 $\hookrightarrow$ ACTIVE set [[NODE_1], [NODE_2], ...].

You should immediately return the set of nodes
 $\hookrightarrow$ that the operation results in, with no
 $\hookrightarrow$ additional text. Return your final answer
 $\hookrightarrow$ as a list of nodes in the very last line
 $\hookrightarrow$ of your response. For example, if the
 $\hookrightarrow$ operation returns the set of nodes [node1,
 $\hookrightarrow$ node2, node3], your response should be:
 Final Answer: [node1, node2, node3]
 If the operation returns the empty set, your
 $\hookrightarrow$ response should be:
 Final Answer: []

H.5 Subgraph Matching

You will be given a host graph G and a pattern
 $\hookrightarrow$ graph T as lists of undirected edges. All
 $\hookrightarrow$ nodes are at least degree 1.
 You will also get a description of an
 $\hookrightarrow$ operation to perform on the graphs.
 Your job is to execute the operation and
 $\hookrightarrow$ return the set of matches that the
 $\hookrightarrow$ operation results in.

A match is represented ONLY by the set of
 $\hookrightarrow$ nodes in G that are used to match all
 $\hookrightarrow$ nodes of T (so each match is a list of
 $\hookrightarrow$ |T_nodes| node strings).
 The order of nodes inside a match does not
 $\hookrightarrow$ matter, and the order of matches does not
 $\hookrightarrow$ matter.
 Do not output duplicate matches (treat each
 $\hookrightarrow$ match as a node set).

Here is an example:
 <begin example>

The host graph G has the following edges:
 aaaa -- bbbb
 bbbb -- cccc
 cccc -- aaaa
 cccc -- dddd

The pattern graph T has the following edges:
 t111 -- t222
 t222 -- t333
 t333 -- t111

Example 1:
 Operation:
 Enumerate all subgraph-isomorphic matches of T
 $\hookrightarrow$ in G (non-induced).
 Final Answer: [[aaaa, bbbb, cccc]]

The host graph G has the following edges:
 aaaa -- bbbb
 bbbb -- cccc
 cccc -- dddd

The pattern graph T has the following edges:
 t111 -- t222
 t222 -- t333
 t333 -- t111

Example 2:
 Operation:
 Enumerate all subgraph-isomorphic matches of T
 $\hookrightarrow$ in G (non-induced).
 Final Answer: []

<end example>

Here is the host graph G to operate on:
 The host graph G has the following edges:
 [HOST UNDIRECTED EDGE LIST]

Here is the pattern graph T:
 The pattern graph T has the following edges:
 [PATTERN UNDIRECTED EDGE LIST]

Operation:
 Enumerate all subgraph-isomorphic matches of T
 $\hookrightarrow$ in G (non-induced).

You should immediately return the set of
 $\hookrightarrow$ matches that the operation results in,
 $\hookrightarrow$ with no additional text.
 Return your final answer as a list of matches
 $\hookrightarrow$ in the very last line of your response.
 $\hookrightarrow$ For example, if the operation returns the
 $\hookrightarrow$ matches [[node1, node2, node3], [node4,
 $\hookrightarrow$ node5, node6]], your response should be:
 Final Answer: [[node1, node2, node3], [node4,
 $\hookrightarrow$ node5, node6]]
 If the operation returns the empty set, your
 $\hookrightarrow$ response should be:
 Final Answer: []

H.6 Main-Split Local Motif Enumeration

You will be given a graph as a list of
 $\hookrightarrow$ directed edges. All nodes are at least
 $\hookrightarrow$ degree 1. Edges may contain duplicates and
 $\hookrightarrow$ self-loops.

Your job is to find all directed triangles in
 $\hookrightarrow$ the graph.

Definition:
 - A directed triangle is a triple of three
 $\hookrightarrow$ distinct nodes [a,b,c] such that $a \rightarrow b$,
 $\hookrightarrow$ $b \rightarrow c$, and $c \rightarrow a$ all appear.

Output format (canonicalization):
 1) Write each triangle in cycle order as
 $\hookrightarrow$ [a,b,c].
 2) Rotate each triangle so that the
 $\hookrightarrow$ lexicographically smallest node string is
 $\hookrightarrow$ first.

3) Deduplicate and sort all triangles
↪ lexicographically.

Here is an example:
<begin example>

The graph has the following edges:

aaaa111111 -> bbbb222222
bbbb222222 -> cccc333333
cccc333333 -> aaaa111111
zzzz999999 -> zzzz999999
aaaa111111 -> dddd444444
dddd444444 -> eeee555555

Operation:
Find all directed triangles.

Final Answer: [[aaaa111111, bbbb222222,
↪ cccc333333]]

<end example>

Here is the graph to operate on:
The graph has the following edges:
[DIRECTED EDGE LIST; DUPLICATES/SELF-LOOPS MAY
↪ APPEAR]

Operation:
Find all directed triangles.

You should immediately return the complete
↪ result with no additional text. Return
↪ your final answer as a list of triangles
↪ in the very last line of your response.
If there are no triangles, your response
↪ should be:
Final Answer: []
Otherwise, your response should be:
Final Answer: [[nodeA, nodeB, nodeC], ...]