GRAPHTRACEBENCH: DIAGNOSING LONG-RANGE REASONING IN GRAPH TRANSFORMERS AND LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-range reasoning over graphs requires models to recover latent computation traces and execute multi-step operations beyond local neighborhoods. We introduce GraphTraceBench, a synthetic benchmark suite for controlled evaluation of long-range graph reasoning. GraphTraceBench contains two complementary tasks: GraphLR-Path, where the target is obtained by executing arithmetic operators along the unique directed path from a source node to a destination node, and GraphLR-BFS, where the target is defined by an arithmetic program executed over the breadth-first discovery order of a directed graph. The benchmark allows independent control over graph size, reasoning depth, operator vocabulary, value range, and distractor density, and provides explicit path and traversal annotations for fine-grained error analysis at different execution horizons. We evaluate graph-native transformers and foundation models under matched graph-execution tasks, using both structured and text-serialized graph inputs. Across models, performance degrades sharply as reasoning depth increases, and cross-length generalization remains poor, suggesting that current models often fail to learn length-invariant graph execution procedures. Additional analyses reveal that operator composition, distractor structure, and input modality substantially affect model behavior. GraphTraceBench provides a diagnostic testbed for studying whether graph-native and large language models can perform controlled long-range execution over directed graphs.

1 INTRODUCTION

Long-range reasoning over graphs is central to many structured prediction and relational reasoning problems (Gilmer et al., 2017; Battaglia et al., 2018; Alon & Yahav, 2020; Topping et al., 2021; Dwivedi et al., 2022), where the correct output may depend on a multi-hop chain, an ordered traversal process, or a latent program executed over distant parts of the graph (Gilmer et al., 2017; Battaglia et al., 2018; Loukas, 2019; Velickovic et al., 2019; Velivckovi'c et al., 2022). Message-passing graph neural networks provide a natural inductive bias for such problems, but their ability to capture distant dependencies is limited by depth, under-reaching, and over-squashing (Alon & Yahav, 2020; Topping et al., 2021; Giovanni et al., 2023). This limitation has motivated Graph Transformers and hybrid architectures that combine structural encodings, global attention, and local propagation to improve information flow beyond local neighborhoods (Ying et al., 2021; Chen et al., 2022a; Rampášek et al., 2022). In parallel, large language models and vision-language models have been applied to graph reasoning from serialized text, tabular descriptions, program-like representations, and visual diagrams, raising the question of whether general-purpose foundation models can execute graph-structured computations without explicit message-passing mechanisms (Wang et al., 2023; Fatemi et al., 2024; Chen et al., 2024; Luo et al., 2024; Tang et al., 2025). However, despite dedicated benchmarks such as LRGB (Dwivedi et al., 2022), recent work has noted that measuring long-range graph reasoning remains nontrivial: existing evaluations often entangle long-range dependency, graph size, structural noise, label distribution, input representation, and task-specific heuristics, making it difficult to determine whether a model has learned to execute a graph-structured computation across different graph execution horizons (Tönshoff et al., 2024; Bamberger et al., 2025).

A related but more controlled perspective is offered by neural algorithmic reasoning, which asks whether models can execute procedures such as breadth-first search or shortest-path style computation

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.

rather than merely fit final labels (Velickovic et al., 2019; Velickovic & Blundell, 2021; Velivckovi'c et al., 2022). For foundation models receiving text-serialized graphs, strong performance may reflect textual patterns, arithmetic shortcuts, or serialization-specific correlations rather than faithful execution. Existing LLM-oriented graph benchmarks typically emphasize broad graph QA, instruction-tuned problem solving, or classical algorithms, leaving room for targeted evaluation of *controlled long-range execution*. Such evaluation should combine algorithmic controllability with LLM-compatible inputs while varying reasoning depth, graph size, operator composition, and distractor structure under clearly specified experimental conditions.

We introduce **GraphTraceBench**, a synthetic benchmark family with two complementary tasks. **GraphLR-Path** defines the target by executing arithmetic operators along the unique directed source-to-destination path. **GraphLR-BFS** defines it through an arithmetic program over the graph's breadth-first discovery order. These tasks support controlled evaluation of path-based and traversal-ordered execution across graph sizes, reasoning horizons, operator compositions, and distractor structures. Rather than approximating a real-world graph domain, GraphTraceBench provides an executable testbed for manipulating graph structure, program length, and potential shortcuts.

Using GraphTraceBench, we systematically evaluate graph Transformer baselines alongside large language models on graph reasoning from serialized graph text. (Ying et al., 2021; Rampášek et al., 2022; Shirzad et al., 2023; Ma et al., 2023; Wu et al., 2023; Choi et al., 2024; Gao et al., 2022; Zhao et al., 2023; Chen et al., 2025; Wang et al., 2024b; Bai et al., 2025; OpenAI, 2026; Anthropic, 2026) We study how performance changes with path length and traversal depth, how operator composition affects difficulty, what latent-space geometry reveals about learned representations, and whether the models truly rely on graph structure when solving the tasks. By focusing on numerical reasoning over serialized graphs rather than broad graph QA, GraphTraceBench provides a controlled diagnostic benchmark for evaluating graph Transformers and LLMs on long-range graph computation.

Contributions. Our contributions are threefold. First, we introduce GraphTraceBench, a synthetic benchmark suite for evaluating long-range graph reasoning from serialized graph text, with two complementary tasks: GraphLR-Path and GraphLR-BFS. Second, we provide a controlled empirical study of large language models on algorithmic graph execution tasks, using graph transformers as structured baselines. Third, we go beyond standard accuracy metrics by systematically analyzing operator sensitivity, latent representations, and structural interventions to better understand what current models learn—and fail to learn—when reasoning over graph-structured text.

2 Related Work

Graph Transformers and Long-Range Graph Modeling. Long-range graph learning is tightly linked to bottlenecks and over-squashing in message passing (Alon & Yahav, 2020; Topping et al., 2021). Graph transformers address this via global attention, tokenization, or stronger structural encodings, spanning SAN (Kreuzer et al., 2021), Graphormer (Ying et al., 2021), TokenGT (Kim et al., 2022), SAT (Chen et al., 2022a), GPS (Rampášek et al., 2022), Exphormer (Shirzad et al., 2023), GRIT (Ma et al., 2023), DIFFormer (Wu et al., 2023), and NAGphormer (Chen et al., 2022b). LRGB crystallized this challenge in benchmark form (Dwivedi et al., 2022), while subsequent analyses questioned whether existing datasets cleanly isolate long-range interactions from other sources of difficulty (Tönshoff et al., 2024; Bamberger et al., 2025).

Neural Algorithmic Reasoning on Graphs. Neural algorithmic reasoning asks models to execute procedures rather than fit end labels (Velickovic & Blundell, 2021). Core milestones include Neural Execution of Graph Algorithms and CLRS (Velickovic et al., 2019; Velivckovi'c et al., 2022), together with extensions without intermediate hints, open-book training, and transformer-centric analyses of graph algorithms (Rodionov & Prokhorenkova, 2023; Li et al., 2024; Sanford et al., 2024). GraphTraceBench complements this line by focusing on two minimal graph programs—unique-path arithmetic execution and BFS-ordered execution—with explicit control of depth, operator composition, and distractors for systematic analysis of execution.

LLMs for Textual Graph Reasoning. Text-serialized graph reasoning has been studied through benchmarks, encodings, and instruction-tuned models, including NLGraph (Wang et al., 2023), Talk like a Graph (Fatemi et al., 2024), GraphToken (Perozzi et al., 2024), GraphInstruct (Luo et al.,

2024), GraphWiz (Chen et al., 2024), InstructGraph (Wang et al., 2024a), Graph-CoT (Jin et al., 2024), AskGNN (Hu et al., 2024), and GraphArena (Tang et al., 2025). These works show that LLM graph reasoning is highly sensitive to serialization choices, prompting format, and benchmark design. Related efforts such as GraphText (Zhao et al., 2024), GraphLLM (Chai et al., 2023), GraCoRe (Yuan et al., 2024), and GraphWalks (OpenAI, 2025) further explore graph-to-text interfaces, graph-aware augmentation, language-centric graph comprehension, and multi-hop operations over textualized graphs. Recent analyses also report persistent brittleness and memorization/generalization gaps (Hu et al., 2025; Zhang et al., 2024; Xypolopoulos et al., 2024). In contrast, GraphTraceBench fixes the latent graph program and varies only graph structure and reasoning horizon, enabling controlled analysis of path length, BFS depth, operator composition, and distractor structure.

3 DATASET

We introduce **GraphTraceBench**, a family of synthetic benchmarks for systematically evaluating long-range reasoning over directed graphs. GraphTraceBench contains two subsets, **GraphLR-Path** and **GraphLR-BFS** (**Graph L**ong Range **R**easoning), which share the same graph vocabulary but differ in the latent computation required to recover the graph-level target. As illustrated in Figure 1, GraphTraceBench consists of two complementary task families: GraphLR-Path, which concentrates reasoning along a unique source-to-target chain, and GraphLR-BFS, which defines the target label through an ordered breadth-first traversal program.

Preliminary. Formally, let $\mathcal{D} = \{(G_k, \boldsymbol{y}_k)\}_{k=1}^{N}$ denote a dataset of N directed graphs. Each graph is represented as $G = (\mathcal{V}, \mathcal{E}, \boldsymbol{X}, \boldsymbol{R}, s, t)$, where $\mathcal{V} = \{v_1, \dots, v_n\}$ is the node set, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the directed edge set, $\boldsymbol{X} \in \mathbb{R}^{n \times d_{\text{node}}}$ is the node feature matrix, and $\boldsymbol{R} \in \mathbb{R}^{m \times d_{\text{edge}}}$ is the edge attribute matrix with $m = |\mathcal{E}|$. The nodes $s, t \in \mathcal{V}$ denote the designated source and destination nodes, respectively. The graph structure can be equivalently described by an adjacency matrix $\boldsymbol{A} \in \{0,1\}^{n \times n}$, where $A_{ij} = 1$ iff $(v_i, v_j) \in \mathcal{E}$, or by the implementation-friendly representation $\text{edge_index} \in \{1, \dots, n\}^{2 \times m}$. In GraphTraceBench, node features instantiate both symbolic value and task markers, e.g., $\boldsymbol{x}_i = [\text{val}(v_i), \mathbf{1}[v_i = s], \mathbf{1}[v_i = t]]$, while each edge $e \in \mathcal{E}$ is associated with an arithmetic operator $\text{op}(e) \in \mathcal{O} = \{+, -, \times, \div\}$, encoded in $\boldsymbol{R}$ either as an operator ID. Each graph is paired with a graph-level scalar target $y \in \mathbb{Z}$ (or $y \in \mathbb{R}$ after normalization). The learning objective is to train a predictor f_θ that maps a given graph to its target, $f_\theta : G \mapsto y$. For graph-native models, f_θ takes $(\boldsymbol{X}, \text{edge_index}, \boldsymbol{R})$ as input, whereas for LLMs, the same graph structure is converted into a textual representation and optionally paired with a task-specific prompt, while the target semantics remain unchanged.

Across both subsets, node values are sampled from a bounded integer range, and graph generation enforces safe arithmetic execution throughout the process. In particular, division is allowed only when exact division is possible, multiplicative operands are restricted to prevent uncontrolled growth, and samples are rejected whenever intermediate or final values exceed a magnitude bound.

3.1 GRAPHLR-PATH

GraphLR-Path evaluates long-range reasoning along a unique source-to-destination chain. Each graph contains exactly one directed path $\mathcal{P}(s,t) = (p_0 = s, p_1, \dots, p_L = t)$, where $p_k \in \mathcal{V}$ is the k-th node on the path. Let $v(u)$ denote the scalar value associated with node u, and, for an edge between nodes u and w, let $\text{op}(u,w) \in \mathcal{O}$ denote the arithmetic operator associated with $\text{edge}(u,w)$, where L is the prescribed reasoning horizon. The target is obtained by executing the sequence of edge operators along this path from left to right, maintaining a running scalar accumulator:

$$\begin{aligned} z_0 &= v(p_0), \\ z_k &= z_{k-1} \ \text{op}(p_{k-1}, p_k) \ v(p_k), \qquad k = 1, \dots, L, \\ y &= z_L. \end{aligned} \tag{1}$$

Accordingly, solving GraphLR-Path requires recovering the unique relevant path in the presence of irrelevant graph structure and then carrying out a multi-step arithmetic program over that path.

The generation process begins by constructing a backbone chain of length L from s to t. Node values and edge operators on this chain are sampled sequentially under the safe-execution constraints described above, so that every intermediate accumulator remains valid and the final label stays within

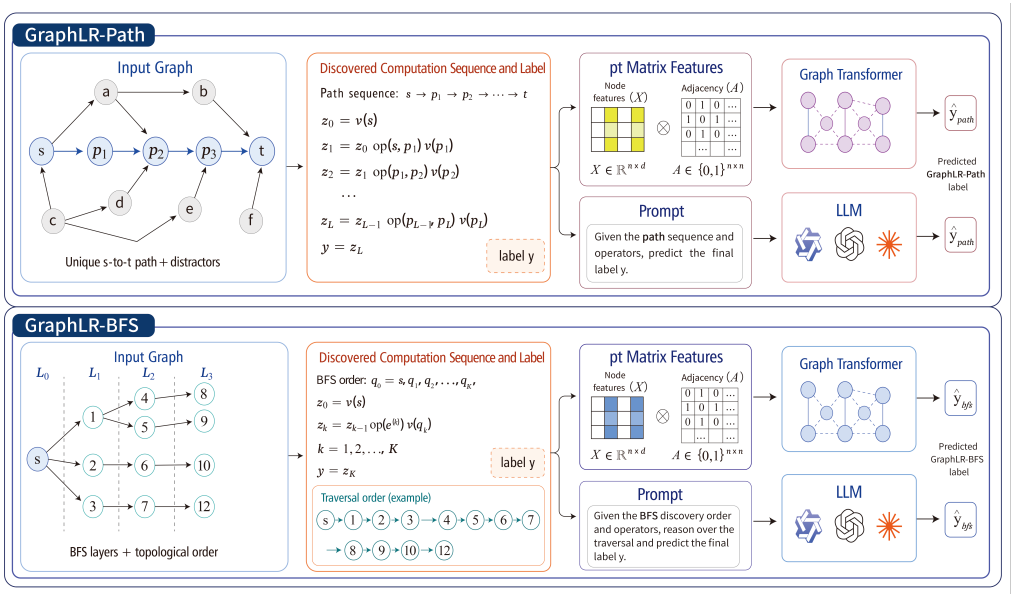

Figure 1: Overview of the GraphTraceBench benchmark. GraphLR-Path evaluates long-range reasoning along a unique source-to-target path, while GraphLR-BFS requires recovering and executing a breadth-first traversal program. For each task, the computation sequence and label are converted into pseudotime matrix features for graph transformers and natural-language prompts for LLMs.

the target range. After the backbone has been fixed, additional nodes and edges are attached as distractors. These distractors increase graph size and local branching, but they are added under a strict uniqueness constraint:$|\mathcal{P}(s,t)| = 1$, where $\mathcal{P}(s,t)$ denotes the set of directed paths from s to t. As a result, the graph may contain substantial irrelevant structure while still admitting exactly one valid execution trajectory. Each released sample stores the explicit path annotation through $\mathrm{path_nodes}$ and $\mathrm{path_edge_indices}$, together with the graph-level label y.

GraphLR-Path isolates the setting in which long-range reasoning is concentrated on a single latent chain within the graph. Its difficulty is controlled directly by the path length L, while the number and density of distractor structures determine how much irrelevant connectivity the model must identify and ignore before executing the correct arithmetic program.

3.2 GRAPHLR-BFS

GraphLR-BFS extends the task from path-based execution to traversal-based execution. Instead of depending only on a single path, the label is defined by an implicit program executed over the BFS discovery order starting from the source node s. Let $\pi_{\mathrm{BFS}}(s) = \big(q_0 = s, q_1, q_2, \ldots, q_K\big)$ denote the node discovery sequence obtained by breadth-first search up to depth L. For each newly discovered node q_k, let e_k be the BFS tree edge through which it is first reached. The target is then computed by

$$\begin{aligned} z_0 &= v(s), \\ z_k &= z_{k-1}\ \mathrm{op}(e^{(k)})\ v(q_k), \qquad k = 1, \ldots, K, \\ y &= z_K. \end{aligned} \tag{2}$$

This formulation requires a model to infer not merely a relevant route, but an ordered graph traversal together with the associated arithmetic execution trace.

To make the BFS program precisely controllable, GraphLR-BFS is generated by explicitly constructing BFS layers $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_L$, with $\mathcal{L}_0 = \{s\}$ and at least one node in every subsequent layer. The first node of each layer defines a designated compatibility spine $r_0 = s, r_1, \ldots, r_L = t$, where $r_\ell \in \mathcal{L}_\ell$, $l \in [1, L]$. This spine guarantees the existence of a long path embedded inside the graph even though the label depends on the full BFS computation rather than on the spine alone. Additional reachable nodes are distributed across layers to create richer branching structure, so that

the BFS sequence is typically substantially longer than the minimal spine. Surplus nodes may also be placed into disconnected distractor subgraphs; these nodes receive valid values and edges but are not reachable from s within depth L and therefore do not affect the target.

Edges between adjacent BFS layers are added in two stages. First, each node in layer $\mathcal{L}_d$ is assigned a first-discovery parent in $\mathcal{L}_{d-1}$, which defines the BFS tree and the arithmetic program used to compute the label. During this step, operators and child values are sampled jointly under the same safe-execution constraints as in GraphLR-Path. Second, additional forward edges may be inserted between adjacent layers to increase graph density. However, these extra edges are only allowed when they do not alter the first-discovery parent, the BFS layer decomposition, or the BFS discovery order. After graph construction, BFS is simulated explicitly, and the sample is accepted only if the simulated layers, node order, tree edges, and parent sequence exactly match the planned construction.

Each GraphLR-BFS sample stores not only the path annotation inherited from GraphLR-Path, but also the complete traversal supervision, including $\mathrm{bfs_layers}$, $\mathrm{bfs_order}$, $\mathrm{bfs_tree_edge_indices}$, $\mathrm{bfs_parent_nodes}$, $\mathrm{layer_index}$, and $\mathrm{reachable_within_depth}$. These annotations make it possible to disentangle failures in graph perception, traversal recovery, and arithmetic execution.

Compared with GraphLR-Path, GraphLR-BFS is designed to be more challenging because it requires traversal-order recovery in addition to arithmetic execution across multiple discovered nodes. It therefore probes whether a model can recover and execute an ordered traversal procedure over a nontrivial directed graph, instead of merely identifying one relevant path.

The generation algorithm of the dataset is presented in Appendix A.

3.3 Dataset Statistics

Both GraphLR-Path and GraphLR-BFS are released with explicit train, validation, and test splits. Their difficulty can be controlled through the number of nodes, reasoning horizon L, operator set, node value range, multiplier bound, maximal label magnitude, and density of extra branch edges. GraphLR-Path isolates unique-path execution under structural distraction, whereas GraphLR-BFS evaluates traversal-conditioned program execution over the BFS discovery process.

Table 1: Default generation hyperparameters and summary statistics of GraphLR.

Parameter	Value
Graphs per fixed-length set	5,000
Graphs per mixed set	20,000
Path lengths	1–6
BFS depths	1–5
Nodes per graph	24
Node value range	$[1, 9]$
Operators	$\{+, -, *, /\}$
Mul. abs. cap	3
Max. abs. label	256
Extra branch edge prob.	0.15
Train/val/test split	0.8 / 0.1 / 0.1
Avg. edges per graph	Path: 35.42 ± 11.27; BFS: 16.38 ± 8.90
Edge count range	Path: $[23, 77]$; BFS: $[3, 42]$
Target range	Path: $[-206, 252]$; BFS: $[-256, 256]$
Path Operator dist. $(+, -, \times, \div)$	$\sim 25\%$ each
BFS Operator dist. $(+, -, \times, \div)$	$\sim 25\%$ each

Table 1 reports the statistics of the four-operator datasets used in our experiments, covering GraphLR-Path with lengths 1–6 and GraphLR-BFS with depths 1–5. The reported statistics are computed over the combined fixed-length datasets for each subset, including edge-count distributions, target ranges, and operator distributions. GraphLR-Path contains denser graphs than GraphLR-BFS, while both subsets maintain a near-uniform operator distribution over addition, subtraction, multiplication, and division. Specifically, the operator distribution for GraphLR-Path is $25.03\%/24.96\%/24.99\%/25.02\%$ for $(+, -, \times, \div)$, and that for GraphLR-BFS is $25.16\%/25.05\%/24.62\%/25.17\%$.

For representation analysis in RQ3, Path-Mix uniformly samples path lengths 1–6, while BFS-Mix uniformly samples BFS depths 1–5. For OOD evaluation, each length or depth i is held out for testing, and the remaining lengths or depths are used for training.

For LLM evaluation, each graph is serialized into a prompt containing full node values and edges, together with $\mathrm{path_hops}$ or BFS depth. The required output format is $\{\text{“answer”} : \mathrm{integer}\}$, and predictions are evaluated by exact match against the ground-truth target value.

4 EXPERIMENTS

In this section, we conduct systematic experiments on GraphLR-Path and GraphLR-BFS to evaluate the long-range reasoning capabilities of graph transformers (GTs) and large language models (LLMs). Our experiments are organized around the following research questions.

• **RQ1:** How do GTs and LLMs perform on graph reasoning tasks as path length and traversal depth increase across progressively more challenging settings?

• **RQ2:** How sensitive are GTs and LLMs to different arithmetic operators, and how does the prompting regime (zero-shot vs. few-shot) influence the robustness and accuracy of LLM performance?

• **RQ3:** What do the learned latent representations of GTs and LLMs reveal about their underlying reasoning behavior and internal solution strategies on GraphTraceBench?

• **RQ4:** How does partial shuffling of the training data affect the reasoning performance of GTs and fine-tuned LLMs in terms of test accuracy and MAE?

4.1 EXPERIMENTAL SETTINGS

Datasets. We evaluate all models on the two subsets of GraphTraceBench, namely GraphLR-Path and GraphLR-BFS. GraphLR-Path focuses on unique-path arithmetic execution, where the target is obtained by executing operators along the only valid path from the source node to the destination node. GraphLR-BFS is more challenging, as the target depends on arithmetic execution over the BFS discovery order induced by the graph. For both datasets, we vary the reasoning horizon by controlling the path length (for GraphLR-Path) or BFS depth (for GraphLR-BFS). We further vary the distractor complexity to test robustness under increasing structural difficulty.

Models. We compare two classes of models. The first class consists of graph-native or graph-pretrained transformer architectures, including Graphormer (Ying et al., 2021), GraphGPS (Rampášek et al., 2022), Exphormer (Shirzad et al., 2023), GRIT (Ma et al., 2023), DIFFormer (Wu et al., 2023), TIGT (Choi et al., 2024), PatchGT (Gao et al., 2022), GraphGPT (Zhao et al., 2023), and G2PT (Chen et al., 2025). The second class consists of large language and vision-language models, including Qwen2-VL (Wang et al., 2024b), Qwen3-VL (Bai et al., 2025), GPT-5.4 (OpenAI, 2026), and Claude Opus 4.7 (Anthropic, 2026). However, since our evaluation uses text-serialized graph inputs without visual modality, we use the term LLMs to collectively refer to these language-based models throughout the paper. For GTs, we use the native structured representation $(\boldsymbol{X}, \mathrm{edge_index}, \boldsymbol{R})$. For LLMs, we consider two settings based on model accessibility: open-source models are fine-tuned on the training split before evaluation, whereas closed-source models are evaluated under a zero-shot prompting setting. These protocols are not matched. For text prompting, each graph is serialized as a task-specific instruction followed by the required task metadata, including $\mathrm{src/dst}$ or $\mathrm{bfs_depth}$, $\mathrm{num_nodes}$, the full node-value list, and the ordered directed edge list with edge operators. For GraphLR-BFS, the edge list is presented in the exact inspection order used during BFS expansion so that the BFS first-discovery sequence is fully determined by the prompt.

Evaluation Metrics. Since both GraphLR-Path and GraphLR-BFS are graph-level regression tasks, we report exact-match accuracy (ACC) and mean absolute error (MAE). In addition, to understand reasoning difficulty, we report bucketed performance as a function of path length, BFS depth, graph size, and distractor density. For operator-specific analysis, we report per-operator accuracy by grouping test instances according to the operator composition appearing in the execution trace.

4.2 SCALING WITH REASONING DEPTH (RQ1)

We first evaluate whether GTs and LLMs can maintain graph reasoning performance as the reasoning horizon increases across the evaluated task configurations. In GraphLR-Path, the horizon is controlled by the unique path length, while in GraphLR-BFS it is controlled by the BFS traversal depth. In RQ1, all GT models and open-source LLMs were evaluated following task-specific fine-tuning, while closed-source LLMs were assessed directly in a zero-shot setting.

Figure 4 shows that both GTs and LLMs degrade as the reasoning horizon increases. On GraphLR-Path, some GTs such as Exphormer and GraphGPS achieve high accuracy on short paths, but their performance drops sharply on longer paths. Other GTs remain low across most path lengths, indicating limited multi-hop arithmetic reasoning ability. LLMs show a similar trend.

GraphLR-BFS is substantially harder: accuracy is much lower across nearly all models, even at small depths. This is because the number of possible computational paths grows exponentially with depth, and BFS reasoning requires aggregating information across multiple paths rather than following a single path. Models must track the traversal frontier, discovery order, and arithmetic execution sequence, making BFS-style algorithmic graph traversal challenging for both GTs and LLMs.

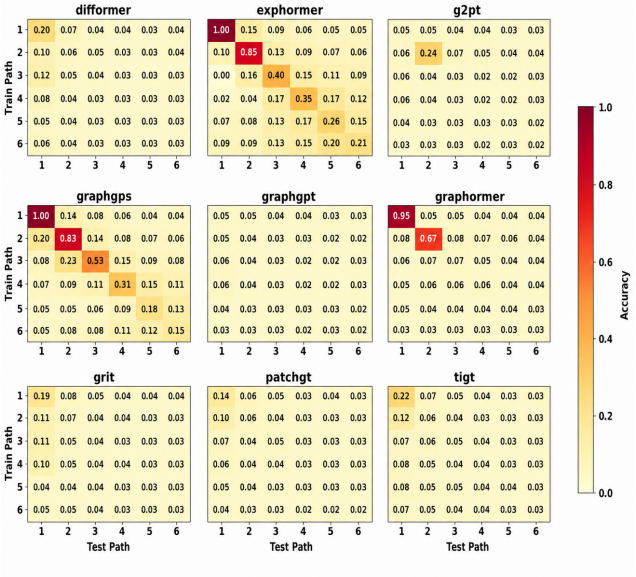

Figure 2: Train/test path-length generalization of nine GTs on GraphLR-Path. Diagonal entries correspond to ID settings, while off-diagonal entries correspond to OOD settings.

The ID/OOD curves in Figure 4(c,d) show weak generalization. We define ID as $\mathrm{train_path} = \mathrm{test_path}$ and all other cases as OOD. Across datasets, OOD accuracy is lower and OOD MAE is higher, indicating that distribution shift leads to fewer exact predictions and larger numerical errors.

Figure 2 provides a finer view of the OOD failure on GraphLR-Path. Accuracy is mostly concentrated near the diagonal, especially for Exphormer and GraphGPS, while off-diagonal entries drop sharply. This suggests that GTs often learn path-length-specific patterns rather than a length-invariant execution algorithm. Overall, increasing path length and BFS depth reveals a core limitation of both GTs and LLMs under the evaluated training configurations: they perform well mainly in shallow or matching-horizon settings, but their performance degrades substantially as the execution horizon increases on tasks combining long-range trace recovery with discrete scalar execution.

4.3 Operator Sensitivity and Prompting Effects (RQ2)

We evaluate operator sensitivity under three arithmetic settings to investigate which types of arithmetic operators the models can better capture: addition/subtraction only (as), multiplication/division only (md), and all operators (allop). As shown in Figure 3, each axis denotes one model, and the radial value denotes accuracy in log scale. Both GTs and LLMs exhibit clear operator-dependent behavior on GraphLR-Path and GraphLR-BFS. On average, allop is the most challenging setting, while md achieves the highest mean accuracy on both tasks. Specifically, the average accuracies of allop, as, and md are 0.056, 0.125, and 0.188 on GraphLR-BFS, and 0.205, 0.285, and 0.328 on GraphLR-Path, respectively.

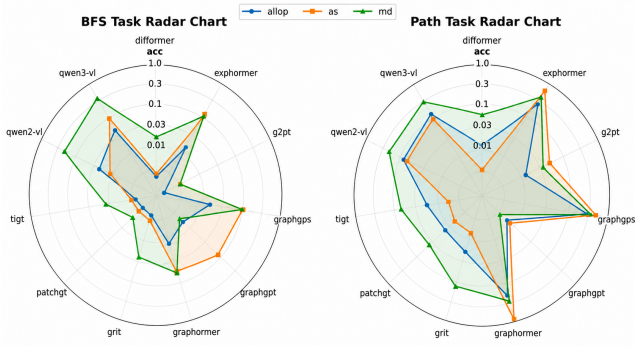

Figure 3: Operator-wise accuracy on GraphLR-BFS and GraphLR-Path.

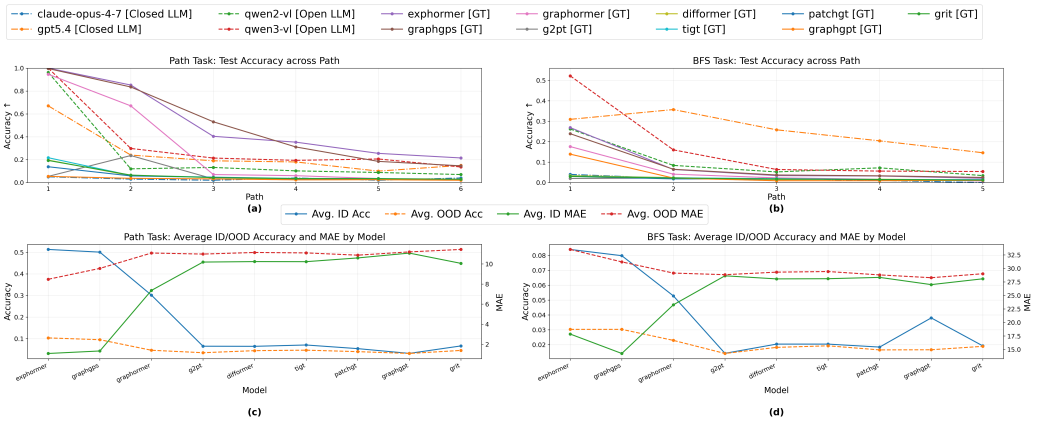

Figure 4: Performance of GTs and LLMs on GraphLR-Path and GraphLR-BFS. Panels (a,b) show test accuracy under increasing path length or BFS depth. Panels (c,d) show model-wise ID/OOD accuracy and MAE for GTs, highlighting cross-horizon generalization performance.

The results indicate that mixed-operator reasoning is more difficult under the current generation constraints, and GraphLR-BFS yields lower accuracy than GraphLR-Path. The relative ranking of as and md varies across models, indicating that performance depends on both the evaluated model and task setting, rather than operator type alone.

4.4 Representation Analysis in Latent Space (RQ3)

We visualize the learned representations of GraphGPS and Qwen2-VL under mixed-hop training. GraphLR-Path contains path lengths from 1 to 6, while GraphLR-BFS contains BFS depths from 1 to 5. As shown in Figure 5, Qwen2-VL produces more clearly hop-separated representations than GraphGPS on both tasks in the displayed t-SNE projections.

This indicates that the LLM is highly sensitive to explicit difficulty cues in the prompt, such as path length, BFS depth, and input complexity. However, this separation does not imply successful reasoning. For example, Qwen2-VL achieves 99.5% exact accuracy on one-hop Path instances, but drops to 9.5% on six-hop instances. On BFS, it still forms clear depth-wise clusters, yet its overall exact accuracy is only 6.2%.

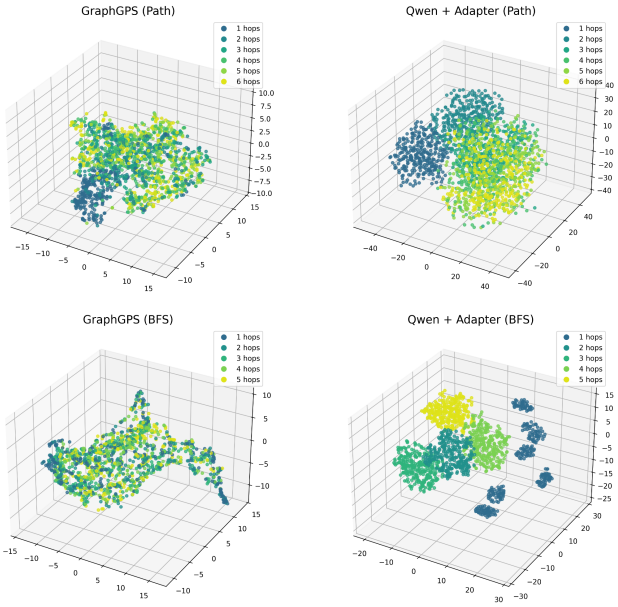

Figure 5: 3D t-SNE visualization of learned latent representations on GraphLR-Path and GraphLR-BFS. Points are colored by reasoning horizon.

GraphGPS exhibits a more continuous and less hop-separable latent geometry. On Path, this representation partially separates short-hop and long-hop examples, while its performance degrades smoothly as the reasoning horizon increases. This suggests that GraphGPS learns a more structure-dependent representation involving graph topology, source–destination states, node values, and edge operations, rather than simply encoding hop count. On BFS, however, GraphGPS representations are highly entangled across depths and its performance is poor, indicating that BFS discovery-order reasoning remains difficult for GraphGPS.

Overall, the latent representations reveal a key distinction between the two model families in the evaluated model configurations. LLMs organize GraphTraceBench inputs strongly according to apparent difficulty, but this difficulty awareness does not reliably translate into algorithmic execution. GTs learn more graph-structured and task-state-dependent representations, yielding smoother degradation on Path reasoning, but they also struggle with BFS-style long-range computation.

4.5 Robustness to Partially Shuffled Training Data (RQ4)

To study the effect of corrupted supervision, we randomly shuffle a portion of the training labels (0%, 5%, 10%, 20%, 50%, and 100%) and fine-tune the models on the perturbed training sets, while always evaluating on the original test sets. We report both test accuracy and MAE on GraphLR-Path and GraphLR-BFS.

Figure 6 shows that partial shuffling consistently hurts performance, but the magnitude differs greatly across model families and tasks. On GraphLR-Path, GraphGPS is highly sensitive to label perturbation: even small shuffle ratios cause clear MAE growth and accuracy drop, and the degradation becomes severe at 50%–100%. In contrast, Qwen2-VL degrades more gradually, indicating stronger robustness to partial supervision noise. Interestingly, Qwen2-VL exhibits a slight increase in accuracy under a small degree of data perturbation, which may be attributed to minor fluctuations arising from its robustness.

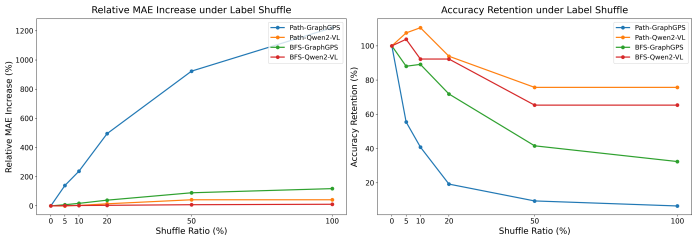

Figure 6: Effect of training-label shuffling on GraphLR-Path and GraphLR-BFS. Left: relative MAE increase over the 0% shuffle setting. Right: accuracy retention relative to the 0% shuffle setting.

A similar trend appears on GraphLR-BFS, although the gap is smaller. GraphGPS degrades monotonically as the shuffle ratio increases, while Qwen2-VL remains comparatively stable, especially at low and moderate ratios. This suggests that GTs are more tightly coupled to the training graph–label correspondence, whereas fine-tuned LLMs are less sensitive to supervision corruption.

Overall, partial shuffling degrades both test accuracy and MAE, but its effect is substantially stronger on GTs than on fine-tuned LLMs, and stronger on GraphLR-Path than on GraphLR-BFS.

Additional Diagnostics. We further examine trace recovery, oracle-path evaluation, symbolic execution, edge ordering, distractor structure, and operator distributions in Appendix C.

5 Conclusion

GraphTraceBench provides a controlled testbed for long-range graph reasoning through two complementary tasks, GraphLR-Path and GraphLR-BFS. Across extensive experiments, we find that both graph transformers and large language models remain far from robust graph execution: performance drops sharply with increasing path length or BFS depth, generalization beyond the training horizon is weak, and GraphLR-BFS is consistently harder than GraphLR-Path. We further show that operator choice substantially affects difficulty, that latent-space separation does not imply correct multi-step execution, and that GTs are more tightly coupled to graph–label structure while LLMs are comparatively less sensitive to corrupted supervision. Taken together, these results suggest that current models can often recognize superficial difficulty cues, but still struggle to carry out faithful long-range computation on graphs, making GraphTraceBench a useful benchmark for future work on structure-aware and length-generalizable reasoning. Nevertheless, our benchmark is still limited to synthetic directed graphs and relatively simple symbolic programs, and the observed behavior of language models may depend in part on the chosen graph serialization format and prompting protocol. Extending GraphTraceBench to more realistic graph domains and more diverse textual interfaces under a broader evaluation setup is an important direction for future work.

AI USE STATEMENT

We used generative AI tools to assist with manuscript drafting and language editing, identify relevant literature during the early stages of the research, and generate scripts for supplementary experiments. We also used these tools to help discuss experimental results and refine the presentation of their interpretations and limitations. The authors reviewed the AI-assisted text, checked suggested references against the original sources, and inspected and tested the generated scripts before incorporating their outputs into the reported results. We take full responsibility for the final manuscript, including its claims, experimental results, and released artifacts.

ETHICS STATEMENT

GraphTraceBench consists of synthetically generated graphs and arithmetic execution tasks. This work does not involve human participants, personal information, or sensitive real-world data. We do not identify direct risks to individuals arising from the benchmark. Its results should be interpreted within the evaluated synthetic tasks and experimental protocols, rather than as a comprehensive assessment of model capabilities or reliability in real-world applications.

REPRODUCIBILITY STATEMENT

Dataset definitions and generation settings are provided in Section 3.3, with generation algorithms in Appendix A. Training configurations, decoding settings, and computational resources are documented in Appendix B. Source code for data generation, training, and evaluation is available at `https://anonymous.4open.science/r/GraphTraceBench-BB2C/`, and datasets are available at `https://huggingface.co/datasets/GraphTraceBench/GraphTraceBench`.

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *ArXiv*, abs/2006.05205, 2020. URL https://api.semanticscholar.org/CorpusID:219558760.

Anthropic. Claude opus 4.7 model. https://www.anthropic.com/system-cards, 2026.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xiong-Hui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Rongyao Fang, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *ArXiv*, abs/2511.21631, 2025. URL https://api.semanticscholar.org/CorpusID:283262018.

Jacob Bamberger, Benjamin Gutteridge, Scott le Roux, Michael M. Bronstein, and Xiaowen Dong. On measuring long-range interactions in graph neural networks. *ArXiv*, abs/2506.05971, 2025. URL https://api.semanticscholar.org/CorpusID:279245071.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charlie Nash, Victoria Langston, Chris Dyer, Nicolas Manfred Otto Heess, Daan Wierstra, Pushmeet Kohli, Matthew M. Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *ArXiv*, abs/1806.01261, 2018. URL https://api.semanticscholar.org/CorpusID:46935302.

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. Graphllm: Boosting graph reasoning ability of large language model. *IEEE Transactions on Big Data*, 12:475–483, 2023. URL https://api.semanticscholar.org/CorpusID:263830019.

Dexiong Chen, Leslie O'Bray, and Karsten M. Borgwardt. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, 2022a. URL https://api.semanticscholar.org/CorpusID:246634635.

Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. Nagphormer: A tokenized graph transformer for node classification in large graphs. In *International Conference on Learning Representations*, 2022b. URL https://api.semanticscholar.org/CorpusID:252846362.

Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. Graphwiz: An instruction-following language model for graph problems. *ArXiv*, abs/2402.16029, 2024. URL https://api.semanticscholar.org/CorpusID:267938586.

Xiaohui Chen, Yinkai Wang, Jiaxing He, Yuanqi Du, Soha Hassoun, Xiaolin Xu, and Li-Ping Liu. Graph generative pre-trained transformer. *ArXiv*, abs/2501.01073, 2025. URL https://api.semanticscholar.org/CorpusID:275212036.

Yuncheol Choi, Sun Woo Park, Minho Lee, and Youngho Woo. Topology-informed graph transformer. *ArXiv*, abs/2402.02005, 2024. URL https://api.semanticscholar.org/CorpusID:267411777.

Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 43909–43934, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/bf72f65f30eedf5d48da6980ee02b589-Paper-Conference.pdf.

Han Gao, Xu Han, Jiaoyang Huang, Jian-Xun Wang, and Liping Liu. PatchGT: Transformer over non-trainable clusters for learning graph representations. In *The First Learning on Graphs Conference*, 2022. URL https://openreview.net/forum?id=Vbfr1jiMxYS.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1263–1272. JMLR.org, 2017.

Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio', and Michael M. Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. *ArXiv*, abs/2302.02941, 2023. URL https://api.semanticscholar.org/CorpusID:256615363.

Yuwei Hu, Xinyi Huang, Zhewei Wei, Yongchao Liu, and Chuntao Hong. Rethinking and benchmarking large language models for graph reasoning. *ArXiv*, abs/2509.24260, 2025. URL https://api.semanticscholar.org/CorpusID:281675602.

Zhengyu Hu, Yichuan Li, Zhengyu Chen, Jingang Wang, Han Liu, Kyumin Lee, and Kaize Ding. Let's ask gnn: Empowering large language model for graph in-context learning. In *Conference on Empirical Methods in Natural Language Processing*, 2024. URL https://api.semanticscholar.org/CorpusID:273228104.

Bowen Jin, Chulin Xie, Jiawei Zhang, Kashob Kumar Roy, Yu Zhang, Suhang Wang, Yu Meng, and Jiawei Han. Graph chain-of-thought: Augmenting large language models by reasoning on graphs. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://api.semanticscholar.org/CorpusID:269033196.

Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 14582–14595. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/5d84236751fe6d25dc06db055a3180b0-Paper-Conference.pdf.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 21618–21629. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/b4fd1d2cb085390fbbadae65e07876a7-Paper.pdf.

Hefei Li, Chao Peng, Chenyang Xu, and Zhengfeng Yang. Open-book neural algorithmic reasoning. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 9819–9836. Curran Associates, Inc., 2024. doi: 10.52202/079017-0314. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/12ffe4499085e9a51beb02441212e26b-Paper-Conference.pdf.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. *ArXiv*, abs/1907.03199, 2019. URL https://api.semanticscholar.org/CorpusID:195833273.

Zihan Luo, Xiran Song, Hong Huang, Jianxun Lian, Chenhao Zhang, Jinqi Jiang, Xing Xie, and Hai Jin. Graphinstruct: Empowering large language models with graph understanding and reasoning capability. *ArXiv*, abs/2403.04483, 2024. URL https://api.semanticscholar.org/CorpusID:268264817.

Liheng Ma, Chen Lin, Derek Lim, Adriana Romero-Soriano, Puneet Kumar Dokania, Mark Coates, Philip H. S. Torr, and Ser Nam Lim. Graph inductive biases in transformers without message passing. In *International Conference on Machine Learning*, 2023. URL https://api.semanticscholar.org/CorpusID:258960285.

OpenAI. Graphwalks dataset (version 1.0). https://huggingface.co/datasets/openai/graphwalks, 2025. Accessed: 2026-05-07.

OpenAI. Gpt-5.4 model. https://openai.com/index/introducing-gpt-5-4/, 2026.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan J. Halcrow. Let your graph do the talking: Encoding structured data for llms. *ArXiv*, abs/2402.05862, 2024. URL https://api.semanticscholar.org/CorpusID:267547812.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 14501–14515. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/5d4834a159f1547b267a05a4e2b7cf5e-Paper-Conference.pdf.

Gleb Rodionov and Liudmila Prokhorenkova. Neural algorithmic reasoning without intermediate supervision. *ArXiv*, abs/2306.13411, 2023. URL https://api.semanticscholar.org/CorpusID:259243699.

Clayton Sanford, Bahare Fatemi, Ethan Hall, Anton Tsitsulin, Mehran Kazemi, Jonathan Halcrow, Bryan Perozzi, and Vahab Mirrokni. Understanding transformer reasoning capabilities via graph algorithms. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 78320–78370. Curran Associates, Inc., 2024. doi: 10.52202/079017-2490. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/8f395480c04ac6dfb2c2326a639df88e-Paper-Conference.pdf.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J. Sutherland, and Ali Kemal Sinop. Exphormer: Scaling graph transformers with expander graphs, 2023. URL https://openreview.net/forum?id=8Tr3v4ueNd7.

Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. Grapharena: Evaluating and exploring large language models on graph computation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Y1r9yCMzeA.

Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe. Where did the gap go? reassessing the long-range graph benchmark. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=Nm0WX86sKv.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *ArXiv*, abs/2111.14522, 2021. URL https://api.semanticscholar.org/CorpusID:244713935.

Petar Velickovic and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2, 2021. URL https://api.semanticscholar.org/CorpusID:233864602.

Petar Velickovic, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. *ArXiv*, abs/1910.10593, 2019. URL https://api.semanticscholar.org/CorpusID:204837843.

Petar Velivckovi'c, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Mikhail Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:249210177.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? *ArXiv*, abs/2305.10037, 2023. URL https://api.semanticscholar.org/CorpusID:258740923.

Jianing Wang, Junda Wu, Yupeng Hou, Yao Liu, Ming Gao, and Julian McAuley. InstructGraph: Boosting large language models via graph-centric instruction tuning and preference alignment. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13492–13510, Bangkok, Thailand, August 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.801. URL https://aclanthology.org/2024.findings-acl.801/.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Ke-Yang Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *ArXiv*, abs/2409.12191, 2024b. URL https://api.semanticscholar.org/CorpusID:272704132.

Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. DIFFormer: Scalable (graph) transformers induced by energy constrained diffusion. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=j6zUzrapY3L.

Christos Xypolopoulos, Guokan Shang, Xiao Fei, Giannis Nikolentzos, Hadi Abdine, Iakovos Evdaimon, Michail Chatzianastasis, G. Stamou, and Michalis Vazirgiannis. Graph linearization methods for reasoning on graphs with large language models. *ArXiv*, abs/2410.19494, 2024. URL https://api.semanticscholar.org/CorpusID:273638030.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? *ArXiv*, abs/2106.05234, 2021. URL https://api.semanticscholar.org/CorpusID:235376852.

Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. Gracore: Benchmarking graph comprehension and complex reasoning in large language models. In *International Conference on Computational Linguistics*, 2024. URL https://api.semanticscholar.org/CorpusID:270924146.

Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xiaochuang Han, Tianxing He, and Yulia Tsvetkov. Can llm graph reasoning generalize beyond pattern memorization? *ArXiv*, abs/2406.15992, 2024. URL https://api.semanticscholar.org/CorpusID:270702523.

Jianan Zhao, Le Zhuo, Yikang Shen, Meng Qu, Kai Liu, Michael M. Bronstein, Zhaocheng Zhu, and Jian Tang. Graphtext: Graph reasoning in text space. In *Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*, 2024. URL https://openreview.net/forum?id=6cRM0OT03F.

Qifang Zhao, Weidong Ren, Tianyu Li, Xiaoxiao Xu, and Hong Liu. Graphgpt: Generative pre-trained graph eulerian transformer. In *International Conference on Machine Learning*, 2023. URL https://api.semanticscholar.org/CorpusID:266377542.

A DATASET ALGORITHM

Algorithm 1 GraphLR-Path generation

Require: number of nodes N, path length L, value range $\mathcal{V}$, operator set $\mathcal{O}$, multiplication bound $B_\times$, label bound B_y

Ensure: directed graph $G = (V, E)$ with unique path from s to t, path annotation, and label y

Sample $L + 1$ distinct nodes $(p_0, \ldots, p_L)$ and set $s \leftarrow p_0$, $t \leftarrow p_L$

Initialize directed backbone path

$$p_0 \to p_1 \to \cdots \to p_L$$

Sample $v(p_0)$ from non-zero values in $\mathcal{V}$ and set $z \leftarrow v(p_0)$

for $k = 1$ to L **do**

  repeat

    Sample operator $\mathrm{op} \sim \mathcal{O}$

    Sample node value $v(p_k) \sim \mathcal{V}$

    if $\mathrm{op} = \div$ **then**

      reject unless $v(p_k) \neq 0$ and $z \bmod v(p_k) = 0$

    end if

    if $\mathrm{op} = \times$ **then**

      reject unless $|v(p_k)| \leq B_\times$

    end if

    Compute $z' \leftarrow z \ \mathrm{op} \ v(p_k)$

  until $|z'| \leq B_y$

  Add directed edge (p_{k-1}, p_k) with operator op

  Update $z \leftarrow z'$

end for

Set $y \leftarrow z$

Let $V_{\mathrm{dist}} \leftarrow V \setminus \{p_0, \ldots, p_L\}$

Attach distractor nodes and distractor edges to increase graph size and branching

while a proposed distractor edge would create another directed path from s to t **do**

  reject this edge

end while

Record

$$\text{path_nodes} = (p_0, \ldots, p_L), \qquad \text{path_edge_indices}$$

return graph fields (node_values, edge_index, edge_ops, src, dst, path_nodes, path_edge_indices, y)

Algorithm 2 GraphLR-BFS generation (simplified)

Require: number of nodes N, BFS depth L, value range $\mathcal{V}$, operator set $\mathcal{O}$, multiplication bound $B_\times$, label bound B_y, extra-edge probability ρ
Ensure: directed graph $G = (V, E)$, BFS annotations, path annotation, and label y

Sample BFS layer sizes $(|\mathcal{L}_0|, \ldots, |\mathcal{L}_L|)$, $|\mathcal{L}_0| = 1$, $|\mathcal{L}_d| \geq 1$
Allocate surplus nodes to reachable layers $\mathcal{L}_1 \ldots \mathcal{L}_L$; place remaining in distractor set U
Randomly assign node identities; set s as unique node in $\mathcal{L}_0$
Define compatibility spine $p_d \leftarrow$ first node in $\mathcal{L}_d$, $d = 0, \ldots, L$, and $t \leftarrow p_L$
Sample $v(s)$ and initialize accumulator $z \leftarrow v(s)$
Initialize empty lists E, E_{BFS}, P_{BFS}
for $d = 1$ to L **do**
    Let parents = $\mathcal{L}_{d-1}$, children = $\mathcal{L}_d$
    Sample child-count allocation over parents
    for each child $c \in \mathcal{L}_d$ **do**
        Assign first-discovery parent $u \in \mathcal{L}_{d-1}$
        repeat
            Sample operator $\mathrm{op} \sim \mathcal{O}$ and value $v(c) \sim \mathcal{V}$
            if $\mathrm{op} = \div$ **then** reject unless $v(c) \neq 0$ and $z \bmod v(c) = 0$
            end if
            if $\mathrm{op} = \times$ **then** reject unless $|v(c)| \leq B_\times$
            end if
            Compute $z' \leftarrow z \mathrm{\ op\ } v(c)$
        until $|z'| \leq B_y$
        Add edge (u, c) to E with operator op, append to E_{BFS} and P_{BFS}, update $z \leftarrow z'$
    end for
end for
for $d = 1$ to L **do**
    for child $c \in \mathcal{L}_d$ **do**
        for later parent u **do**
            if Bernoulli$(\rho) = 1$ **then** Add extra edge (u, c)
            end if
        end for
    end for
end for
Attach distractor subgraph on U and simulate BFS; reject & restart if layers/order/tree differ
Compute label: $z_0 \leftarrow v(s)$, $z_k \leftarrow z_{k-1}\ \mathrm{op}(e^{(k)})\ v(q^{(k)})$, $y \leftarrow z_K$
Record path_nodes, bfs_layers, bfs_order, bfs_tree_edge_indices, bfs_parent_nodes
return (node_values, edge_index, edge_ops, src, dst, path_nodes, path_edge_indices, bfs_layers, bfs_order, bfs_tree_edge_indices, bfs_parent_nodes, y)

B Implementation Details

Graph Transformers. All GTs are trained on the training split, with checkpoints selected based on validation performance. Unless otherwise specified, each model is trained for 100 epochs.

Language Models. We use a unified prompting template and deterministic decoding unless otherwise noted. Open-source LLMs are fine-tuned using LoRA for 3 epochs, with a learning rate of 2×10^{-4}, a per-device batch size of 1, gradient accumulation over 16 steps, and a maximum sequence length of 4096 tokens. The LoRA configuration uses rank $r = 16$, scaling parameter $\alpha = 32$, and dropout 0.05. Closed-source LLMs are evaluated through their APIs under zero-shot prompting without task-specific fine-tuning. Results obtained under these different adaptation protocols should not be interpreted as a controlled ranking of model capabilities.

Computational Resources. Local training and evaluation are conducted on NVIDIA RTX 5090 and RTX 4090 GPUs. Under this setup, the average training time is approximately 2 hours for GTs and 1.5 hours for LoRA-based LLM fine-tuning, while inference-time evaluation for all models

typically completes within a few minutes. These runtimes are approximate and depend on the model and task configuration.

C Additional Diagnostic Experiments

We supplement final-answer evaluation with six diagnostic analyses: joint trace-and-answer prediction, oracle-path evaluation, deterministic symbolic solving, edge-order randomization, distractor-structure sensitivity, and arithmetic-setting distributions. These analyses examine trace recovery, execution with ground-truth paths, symbolic solvability, sensitivity to input serialization and distractors, and differences across arithmetic settings.

C.1 Joint Trace-and-Answer Prediction

Setup and metrics. We evaluate joint prediction of the ordered path and final answer on GraphLR-Path. The ground-truth trace is the complete source-to-destination node sequence, including both endpoints. Trace and answer predictions are produced by the same jointly trained model.

For test instance i, let τ_i and y_i denote the ground-truth trace and answer, and let $\hat{\tau}_i$ and $\hat{y}_i$ denote their predictions. We report

$$\mathrm{TraceAcc} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\hat{\tau}_i = \tau_i], \tag{3}$$

$$\mathrm{AnswerAcc} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\hat{y}_i = y_i], \tag{4}$$

$$\mathrm{JointAcc} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\hat{\tau}_i = \tau_i \wedge \hat{y}_i = y_i]. \tag{5}$$

Trace accuracy requires exact agreement with the entire ordered sequence. We additionally report answer accuracy conditioned on a correct trace:

$$\mathrm{AnswerAcc} \mid \mathrm{CorrectTrace} = \frac{\mathrm{JointAcc}}{\mathrm{TraceAcc}}, \tag{6}$$

when $\mathrm{TraceAcc} > 0$; otherwise, it is undefined.

Results. Table 2 shows a substantial gap between trace and answer accuracy for Qwen2-VL. Trace accuracy remains between 98.2% and 100% across path lengths 1–6, whereas answer accuracy decreases from 98.4% at length 1 to 9.0% at length 6. At length 2, answer accuracy conditioned on a correct trace is approximately 63.1%. Correct path prediction therefore does not ensure correct arithmetic execution in this joint-output setting.

Exphormer and GraphGPS exhibit low exact trace accuracy, despite high answer accuracy at length 1. This discrepancy shows that final-answer correctness and explicit trace prediction measure different aspects of performance. Low trace accuracy may reflect limitations of the learned representations, trace decoder, or joint training objective; it does not by itself establish that the original answer-only models fail to use graph structure.

These results concern the joint-prediction configurations on GraphLR-Path. They should not be interpreted as direct observations of the internal computations of answer-only models, or as evidence of the same trace-level behavior on GraphLR-BFS.

C.2 Oracle-Path Evaluation

Setup. We evaluate Exphormer, GraphGPS, and Qwen2-VL under original and oracle-path conditions on GraphLR-Path. The oracle-path condition supplies the ground-truth path to examine whether making the execution route available improves final-answer prediction. We report exact-match answer accuracy for all three models and additionally report MAE for Qwen2-VL.

Table 2: Joint trace-and-answer prediction on GraphLR-Path. All values are proportions. A | T denotes answer accuracy conditioned on a correct trace and is computed from the reported joint and trace accuracies. A dash indicates that no trace was predicted correctly.

Model	Length	Trace Acc.	Answer Acc.	Joint Acc.	A | T
Exphormer	1	0.0788	0.998	0.0776	0.9848
	2	0.0000	0.841	0.0000	–
	3	0.0000	0.553	0.0000	–
	4	0.0000	0.365	0.0000	–
	5	0.0000	0.288	0.0000	–
	6	0.0000	0.218	0.0000	–
GraphGPS	1	0.0682	0.998	0.0682	1.0000
	2	0.0000	0.892	0.0000	–
	3	0.0000	0.563	0.0000	–
	4	0.0000	0.331	0.0000	–
	5	0.0000	0.271	0.0000	–
	6	0.0000	0.148	0.0000	–
Qwen2-VL	1	1.0000	0.984	0.9840	0.9840
	2	0.9820	0.638	0.6200	0.6314
	3	1.0000	0.170	0.1700	0.1700
	4	1.0000	0.144	0.1440	0.1440
	5	1.0000	0.088	0.0880	0.0880
	6	1.0000	0.090	0.0900	0.0900

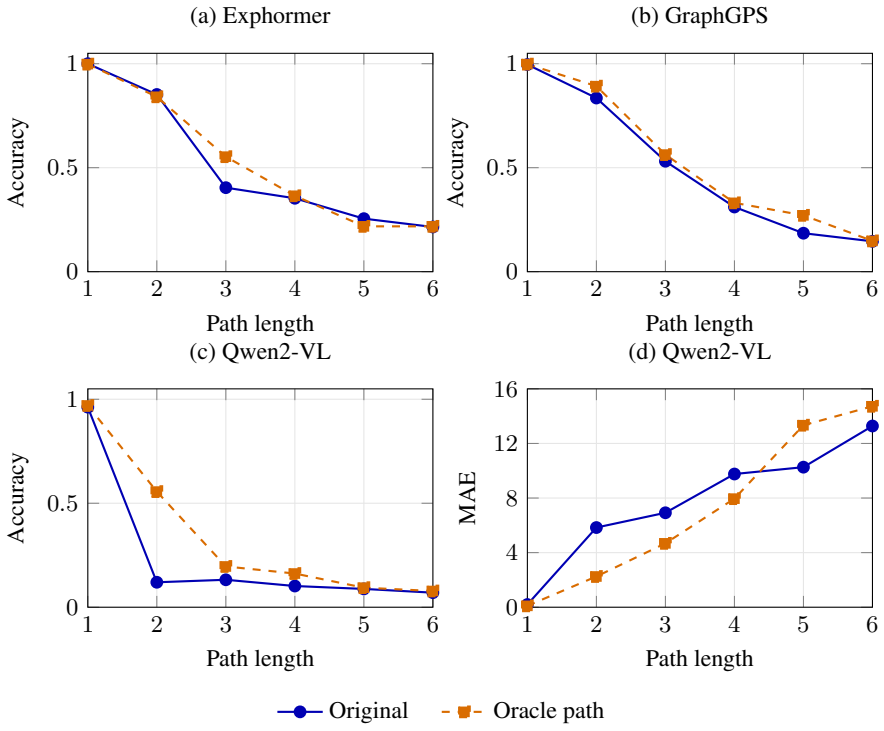

Figure 7: Original and oracle-path evaluation on GraphLR-Path. Panels (a–c) report exact-match answer accuracy for Exphormer, GraphGPS, and Qwen2-VL; panel (d) reports Qwen2-VL MAE. Higher accuracy and lower MAE are better. Comparisons concern the two conditions within each model.

Results. Figure 7 shows that the benefit of providing the ground-truth path varies across models and execution horizons. For Exphormer, accuracy increases from 0.404 to 0.553 at length 3, but decreases from 0.255 to 0.218 at length 5. GraphGPS improves most clearly at lengths 2 and 5, while

Table 3: Deterministic symbolic solver results. The solver achieves exact-match answer accuracy of 1.000 at every evaluated horizon in both task families.

Task	Evaluated horizons	Answer Acc.
GraphLR-Path	Path lengths 1–6	1.000
GraphLR-BFS	BFS depths 1–5	1.000

its accuracy at length 6 remains nearly unchanged. Qwen2-VL exhibits its largest gain at length 2, increasing from 0.120 to 0.556. However, its oracle-path accuracy falls to 0.196 at length 3 and 0.078 at length 6.

Across the three models, supplying the correct path does not prevent substantial accuracy degradation over the evaluated horizons. This indicates that path availability alone is insufficient for reliable multi-step execution under these configurations; the ability to use the supplied trace and perform the associated arithmetic remains relevant.

The effect also depends on the evaluation metric. For Qwen2-VL, oracle-path MAE is lower at lengths 1–4 but higher at lengths 5–6, despite slightly improved exact-match accuracy at the latter lengths. Thus, more exact predictions can coexist with greater average numerical error. The oracle-path condition should be interpreted as a diagnostic intervention rather than a guaranteed performance upper bound.

C.3 Deterministic Symbolic Solver

Setup. We evaluate a non-learning symbolic baseline that parses the input graph, recovers the task-specific execution trace, and computes the final answer using deterministic arithmetic. For GraphLR-Path, the solver searches for the unique directed source-to-destination path. For GraphLR-BFS, it performs breadth-first search under the benchmark's traversal convention and records first-discovery edges. The associated operators and node values are then executed in sequence to obtain the prediction.

Results and interpretation. The deterministic solver achieves 100% answer accuracy across all evaluated tasks and horizons (Table 3). This provides a consistency check between the implemented solver and the reference labels on the evaluated instances, and establishes a non-learning reference for task solvability.

The solver is hand-coded rather than generated by an LLM. Its success therefore does not establish the reliability of an LLM-based tool-use pipeline. Conversely, low direct-answer accuracy characterizes performance under the evaluated tool-free protocol; it does not imply that a model equipped with an external traversal and arithmetic executor would also fail.

C.4 Sensitivity to Edge Ordering

Setup. We compare the original edge ordering with a randomized ordering on GraphLR-Path and GraphLR-BFS. The intervention preserves graph topology, node values, edge operators, reference traces, and target labels. For LLMs, the intervention changes the serialized edge list. For GTs, edge-order controls concern the ordering of edges in the structured input and should be distinguished from textual serialization.

Results on GraphLR-BFS. Table 4 shows small changes in average BFS accuracy: Exphormer changes from 0.084 to 0.090, GraphGPS from 0.080 to 0.078, and Qwen2-VL from 0.101 to 0.103. Qwen2-VL's mean MAE changes from 23.434 to 23.660 (Table 5). Changes occur in both directions across depths, with no systematic accuracy decline in these reported runs.

Results on GraphLR-Path. The Path results show model- and horizon-dependent sensitivity. Average accuracy decreases from 0.513 to 0.486 for Exphormer and from 0.501 to 0.487 for GraphGPS, while increasing from 0.246 to 0.262 for Qwen2-VL. Some individual changes are appreciable, including Exphormer at length 2 and GraphGPS at length 3. The results therefore do not establish

Table 4: Answer accuracy under original (Orig.) and randomized (Rand.) edge ordering. The two task blocks report BFS depth and Path length, respectively. Mean rows are unweighted averages across the reported horizons.

	Exphormer		GraphGPS		Qwen2-VL	
Horizon	Orig.	Rand.	Orig.	Rand.	Orig.	Rand.
GraphLR-BFS						
1	0.269	0.287	0.239	0.227	0.262	0.288
2	0.064	0.072	0.065	0.066	0.084	0.084
3	0.034	0.041	0.037	0.045	0.052	0.056
4	0.032	0.028	0.033	0.030	0.072	0.046
5	0.021	0.023	0.025	0.023	0.034	0.042
Mean	0.084	0.090	0.080	0.078	0.101	0.103
GraphLR-Path						
1	1.000	0.993	0.996	0.998	0.962	0.982
2	0.852	0.747	0.835	0.859	0.120	0.190
3	0.404	0.385	0.531	0.410	0.132	0.132
4	0.353	0.361	0.311	0.323	0.102	0.088
5	0.255	0.251	0.185	0.190	0.088	0.098
6	0.215	0.179	0.146	0.139	0.070	0.080
Mean	0.513	0.486	0.501	0.487	0.246	0.262

Table 5: Qwen2-VL MAE under original and randomized edge ordering. Lower values are better.

Task	Horizon	Original MAE	Randomized MAE
GraphLR-BFS	1	3.794	4.122
	2	15.562	15.334
	3	25.678	26.146
	4	30.794	31.808
	5	41.340	40.890
GraphLR-Path	1	0.198	0.058
	2	5.838	4.594
	3	6.920	7.242
	4	9.760	10.142
	5	10.262	10.312
	6	13.278	13.166

complete order invariance, although they show no uniform performance collapse across the evaluated models.

Interpretation. For Qwen2-VL, the reported randomization results do not show a consistent dependence of answer accuracy on the original textual edge ordering. However, unchanged answer accuracy alone does not establish correct traversal recovery or rule out other shortcuts. In particular, low absolute accuracy at larger BFS depths limits the diagnostic strength of this control. For GTs, changes under edge permutation characterize sensitivity of the evaluated structured-input pipeline, rather than reliance on textual ordering cues.

C.5 Sensitivity to Distractor Structure

Setup. We evaluate Qwen2-VL on GraphLR-Path under three graph configurations: the original 24-node graphs, 36-node graphs with additional non-path nodes, and graphs containing only the target path. The construction preserves the ground-truth path, its node values and edge operators, and the target label, while varying the surrounding distractor structure.

Results. Removing distractors improves accuracy at every evaluated path length (Table 6). The largest improvements occur at lengths 2 and 3, where accuracy increases from 0.120 to 0.830 and

Table 6: Qwen2-VL answer accuracy under different distractor configurations on GraphLR-Path. The no-distractor condition contains only the target path and therefore has $L + 1$ nodes for path length L.

Path length	Original (24 nodes)	36 nodes	No distractors
1	0.962	0.954	1.000
2	0.120	0.498	0.830
3	0.132	0.136	0.514
4	0.102	0.110	0.268
5	0.088	0.072	0.196
6	0.070	0.068	0.132

Table 7: AS/MD distribution audit on 5,000 examples per setting. $H(Y)$ is answer entropy; Unique is the number of distinct labels; Mode is the most frequent label's proportion; Identity is the proportion of execution steps multiplying or dividing by 1; Div.-by-1 is measured among division steps only.

Task	Setting	$H(Y)$	Unique	Mode	Identity	Div.-by-1
Path-3	AS	5.369	58	3.72%	0.00%	–
Path-3	MD	3.806	45	28.62%	42.39%	51.20%
BFS-3	AS	6.306	124	2.34%	0.00%	–
BFS-3	MD	3.622	52	28.26%	43.55%	53.13%

from 0.132 to 0.514, respectively. Nevertheless, accuracy still decreases to 0.132 at length 6 even without distractors, indicating that removing irrelevant structure does not eliminate longer-horizon execution difficulties.

Increasing graph size from 24 to 36 nodes has mixed effects. Accuracy improves substantially at length 2, while changes at other lengths are smaller and occur in both directions. These results do not support a monotonic relationship between node count and prediction difficulty.

The no-distractor intervention also reduces input length and simplifies path identification. Its benefits therefore cannot be attributed exclusively to reduced structural distraction.

C.6 Distribution Audit of Arithmetic Settings

Setup. To investigate the higher aggregate accuracy observed under multiplication/division (MD) than addition/subtraction (AS), we audit 5,000 examples per task–operator setting on Path-3 and BFS-3. We report empirical answer entropy, the number of unique target labels, and the frequency of the most common label. The latter describes the accuracy of always predicting the modal label on the audited sample; it is a distribution statistic rather than a separately trained baseline.

We additionally report the proportion of execution steps that multiply or divide by 1, and the proportion of division steps whose divisor is 1.

Results. Table 7 shows that MD has lower answer entropy and more concentrated target distributions than AS. The modal-label frequency increases from 3.72% to 28.62% on Path-3 and from 2.34% to 28.26% on BFS-3. Moreover, approximately 42–44% of MD execution steps multiply or divide by 1, and over half of division steps use divisor 1.

These statistics expose potential shortcuts and differences in effective arithmetic difficulty under the current safe-execution sampling procedure. They are consistent with a distributional contribution to the observed MD advantage, but do not establish whether or how individual models exploit these regularities. Accordingly, the MD–AS ranking should not be interpreted as evidence of intrinsically stronger multiplication/division ability.

A controlled assessment of the effects of the multiplier cap, exact-division constraint, and label bound would require additional generation and evaluation conditions. The present audit characterizes the existing distributions rather than isolating the causal effect of each constraint.