Large Language Models as Graph Computational Solvers via Topology-aware Residual Attention

Wei Zhuo, Siqiang Luo
Nanyang Technological University
{wei.zhuo, siqiang.luo}@ntu.edu.sg

Abstract

Large language models (LLMs) increasingly serve as general-purpose reasoning engines, yet they remain unreliable on graph computation tasks whose inputs are discrete, permutation-equivalent, and algorithmic. We propose TRACER, a post-training framework that turns LLMs into native graph computational solvers. Our central thesis is that graph computation failures arise from mismatches within the Transformer computation itself: token order obscures graph symmetries, self-attention is biased toward textual proximity instead of graph topology, and autoregressive decoding weakly constrains algorithmic traces. TRACER addresses these challenges at three coupled levels. At the input level, we introduce permutation-invariant graph encoding, which encourages the LLM to map different serialization variants of a graph to a shared internal graph representation, thereby preserving graph symmetries. At the representation level, we enhance self-attention with the Topology-aware Residual Attention mechanism to inject graph topological signals into the attention kernel. At the reasoning level, we employ process-reward RL to encourage faithful execution of graph algorithmic traces. Across linear-time, polynomial-time, and NP-complete graph tasks, TRACER delivers substantial gains over baselines. These results suggest that equipping LLMs with graph-aware internal computation offers a practical path toward reliable neural solvers for structured algorithmic reasoning. The code is available at https://github.com/JhuoW/TRACER.

1 Introduction

Recent advances in mathematical problem solving [17, 9], code synthesis [7, 19], and multi-step question answering [46, 49] have demonstrated the growing reasoning capabilities of large language models (LLMs). A natural and increasingly pressing question is whether LLMs can also reason about graphs, the discrete combinatorial objects that underpin tasks such as routing [21], scheduling [28], knowledge base inference [6], and molecular analysis [13]. Recent benchmarks [8, 38, 43] reveal a striking gap: even strong frontier models attain only modest accuracy on elementary tasks such as connectivity, shortest path, and cycle detection, and degrade sharply as graph size grows. The promise of a single general-purpose reasoner that can also handle structured combinatorial computation therefore remains unrealized.

Two broad strategies have emerged in response. One line of work augments the LLM with external tools or symbolic graph solvers, delegating graph operations to a non-neural backend invoked through code generation or function calling [54, 45, 44]. Another line improves the way graphs are presented to a fixed LLM, through better textual encodings, retrieval, or prompting strategies [12, 43, 55]. Both directions are valuable, but they largely rest on the assumption that *LLMs are architecturally incapable of native graph computation, and that progress must come from routing graph reasoning around the model rather than enhancing the model itself.* In the first case, the LLM is cast primarily as a planner or controller that orchestrates external operations; in the second, the model remains

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

unchanged while the graph is reformulated to better fit its input interface. Neither approach directly asks whether the Transformer architecture itself can be adapted to better receive, organize, and propagate graph-structured information. This question is important because Transformers [41] are universal approximators in principle [51], and graph reasoning reduces in many cases to well-defined symbolic operations that a sufficiently expressive sequence model should be able to simulate. Thus, instead of routing graph computation around the LLM, this paper investigates what architectural support is needed for LLMs to perform graph computation natively. Further discussion is provided in Appendix A.

We identify three mismatches between graph computation and the text-native autoregressive prediction paradigm of LLMs. First, **tokenization destroys permutation invariance**. A graph can be serialized using many valid node relabelings and edge orderings (referred to as graph symmetries), all describing the same object. However, LLMs treat each as a distinct input and have no built-in mechanism to recognize that they are isomorphic. Consequently, mutually isomorphic graphs can elicit different reasoning traces and even different final answers. Second, **self-attention operates over token positions rather than graph topology**. Tokens that mention adjacent nodes may be far apart in the prompt, while neighboring tokens may refer to unrelated nodes. Standard self-attention can connect any token pair permitted by the attention mask, but its logits are not directly conditioned on graph adjacency. Third, **autoregressive decoding of LLMs misaligns with algorithmic traces on graphs**. Graph algorithms update visited sets, frontiers, and partial matchings, but next-token decoding does not by itself ensure that each generated step is a valid state transition or that traces remain consistent across equivalent serializations. These mismatches point to a need for graph-aware mechanisms at the input, representation, and reasoning levels.

In this paper, we propose TRACER (Topology-aware Residual Attention for Computational Execution and Reasoning), a post-training framework that equips LLMs with graph-aware internal computation. TRACER addresses the first challenge at the *input level* through permutation-invariant graph encoding, which is trained on multiple serializations of each graph to align graph-summary representations across equivalent serializations. It addresses the second challenge at the *representation level* with Topology-aware Residual Attention (TRA), which enhances the internal self-attention with a learned graph-conditioned residual, allowing attention between tokens associated with graph elements to incorporate their structural relationships. The third challenge is addressed at the *reasoning level* through reinforcement learning that jointly rewards answer correctness, trace agreement across equivalent serializations, and process-level trace validity. These components target the key failure modes that arise when sequence models are applied to graph-structured reasoning. We evaluate TRACER on nine graph computation tasks spanning linear-time, polynomial-time, and NP-complete complexity. The results show that TRACER improves graph computation accuracy across diverse structural reasoning tasks, highlighting the benefit of aligning LLM computation with graph topology to make LLMs more reliable graph computational solvers.

2 Problem Formulation and Motivation

2.1 Problem Formulation

Graphs and graph computation tasks. A graph is a tuple $G = (\mathcal{V}, \mathcal{E}, w)$, where $\mathcal{V}$ is a finite set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set, and $w : \mathcal{E} \to \mathbb{R}_{\geq 0}$ is an optional edge-weight function (used by tasks such as shortest path and maximum flow). We write $|\mathcal{V}| = n$ and $|\mathcal{E}| = m$. A graph computation task $\mathcal{T}$ is a function $\mathcal{T}(G, q) \mapsto a^\star$ that maps a graph and a query q (e.g., a source–target node pair) to a ground-truth answer $a^\star$.

Serializations and the permutation equivalence class. Because LLMs consume sequences of tokens, every graph passed to the model must first be rendered as text by a serialization function $\sigma : G \to \Sigma^\star$, where $\Sigma^\star$ denotes the space of token sequences over a vocabulary Σ. A serialization is determined by three independent choices: a node-relabeling permutation $\pi_n : \mathcal{V} \to \mathcal{V}'$, an ordering π_e of the edge list, and an ordering π_a of each node's adjacency list. We write $\sigma_{\pi_n, \pi_e, \pi_a}(G)$ for the resulting token sequence and, when the specific permutations are immaterial, simply $\sigma(G)$.

Definition 1 (Serialization equivalence class)**.** *Two serializations σ_1 and σ_2 of a graph G are equivalent, written $\sigma_1 \sim \sigma_2$, if they describe the same graph modulo node relabeling, edge ordering, and adjacency-list ordering. The set of all serializations equivalent to a fixed reference serialization of G forms an equivalence class $[G]_\sim$.*

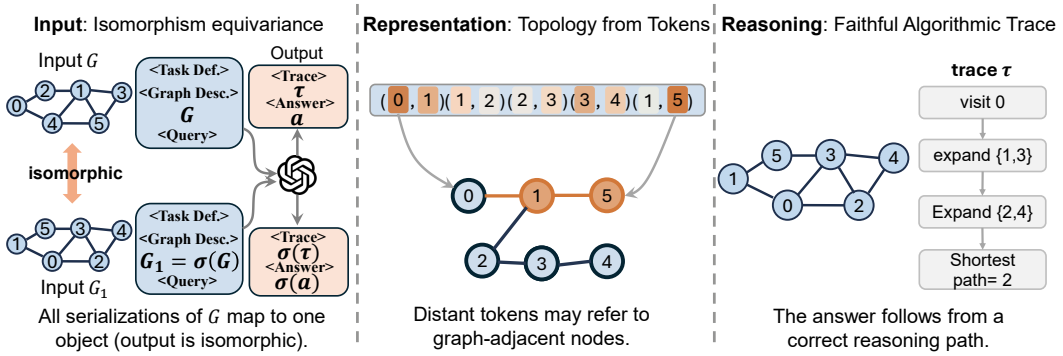

Figure 1: Three requirements for universal graph understanding. **Left**: All serializations of the same graph G should map to a single internal object, with outputs related by the corresponding permutation. **Center**: Tokens that are far apart in the prompt may refer to nodes that are adjacent in the graph, so the LLM must recover topology from the linear token stream. **Right**: The predicted answer should follow from a trace that faithfully executes the underlying algorithm.

LLMs as graph reasoners. We treat the backbone LLM abstractly as a conditional distribution $\mathcal{M}_\theta$ with parameters θ, and assume only that it is an autoregressive Transformer that admits access to its self-attention logits, a property shared by essentially all modern decoder-only architectures [40, 4, 1]. Given a tokenized prompt $x = [\sigma(G);\, q] \in \Sigma^\star$ formed by concatenating a serialization with a task query, the model generates an output sequence $y \sim \mathcal{M}_\theta(\cdot \mid x)$ that we decompose as $y = (\tau,\, a)$, where τ is a natural-language reasoning trace and a is the final answer extracted from a designated span of y (the exact delimiter is dataset-specific and inherited from the benchmark). For each task we define an answer-level loss $\ell_{\text{task}}(a, a^\star)$ that is the standard token-level cross-entropy on the answer span.

Answer equivariance and trace isomorphism. A competent LLM-based graph reasoner should recognize all elements of $[G]_\sim$ as equivalent descriptions of the same graph. We capture this expectation in two requirements. First, the predicted answers across equivalent serializations should be *equivariant*. For any two serializations $\sigma_i, \sigma_j \in [G]_\sim$, let a_i and a_j denote the answers generated under σ_i and σ_j, respectively, using consistently relabeled queries and the same deterministic decoding protocol. Let $\pi_n^{(i)}$ and $\pi_n^{(j)}$ denote the corresponding node-label permutations, which map the original node labels of G to those in σ_i and σ_j, respectively. Answer equivariance requires that a_i and a_j agree after being mapped to the original label space:

$$\left(\pi_n^{(i)}\right)^{-1}(a_i) \equiv \left(\pi_n^{(j)}\right)^{-1}(a_j)\,, \tag{1}$$

where the inverse permutations π^{-1} restore the original node labels of G in the generated answers and $\equiv$ denotes equality of answers in this common label space. These permutations act only on node references, leaving scalar and Yes/No answers unchanged.

Second, the reasoning traces τ_i, τ_j produced under any two equivalent serializations should be *structurally isomorphic*: once both are mapped back to a common label space, they should describe the same sequence of algorithmic events, such as the same nodes being visited, the same edges being relaxed, and the same frontiers being expanded, in the same order. Structural isomorphism is a relational property that holds between pairs of outputs. Supervision on individual reference traces does not directly enforce this cross-serialization agreement, motivating an objective that explicitly couples the model's behavior across equivalent serializations. However, distinct valid executions may follow different admissible event orders, so trace isomorphism and algorithmic validity remain separate properties.

2.2 Motivation

Universal graph understanding under tokenization. Tokenization is the sole interface through which a graph reaches an LLM, and this constraint determines what it means for an LLM-based reasoner to achieve universal graph understanding. We characterize such understanding by three requirements of increasing scope that a model must jointly satisfy, illustrated in Fig. 1:

Table 1: Three equivalent serializations of the same Connectivity instance, differing only by a node relabeling plus edge shuffling (Case 2, with $\pi_n : 0\to 2,\ 1\to 4,\ 2\to 0,\ 3\to 1,\ 4\to 3,\ 5\to 5$ mapping the query $(1,4)\to(4,3)$) or by edge-list reordering alone (Case 3).

Case 1: Original serialization $\sigma_1(G)$.	
Prompt	The nodes are numbered from 0 to 5, and the edges are: (0,1) (1,2) (3,4) (4,5). Is there a path between node 1 and node 4?
Trace	Node 1 is in the connected component $\{0,1,2\}$. Node 4 is in the connected component $\{3,4,5\}$. The two components are disjoint.
Predicted	NO. *(correct)*
Case 2: Equivalent serialization $\sigma_2(G)$ via node relabeling.	
Prompt	The nodes are numbered from 0 to 5, and the edges are: (1,3) (2,4) (3,5) (4,0). Is there a path between node 4 and node 3?
Trace	We trace reachability from node 4: edge (2,4) gives node 2, and edge (4,0) gives node 0. From node 3: edge (1,3) gives node 1, and edge (3,5) gives node 5. Continuing from node 0, we can reach node 3 via... the edges containing node 0 are (4,0). So from node 0 we reach back to 4. The components $\{0,2,4\}$ and $\{1,3,5\}$ appear connected through the edge list. Therefore, a path exists.
Predicted	YES. *(incorrect; ground truth is* NO*)*
Case 3: Equivalent serialization $\sigma_3(G)$ via edge-list reordering only.	
Prompt	The nodes are numbered from 0 to 5, and the edges are: (4,5) (1,2) (0,1) (3,4). Is there a path between node 1 and node 4?
Trace	Node 1 connects to node 2, which has no further connections. Node 4 connects to nodes 5 and 3. There is no overlap between these two reachability sets.
Predicted	NO. *(correct, but the trace structure differs from Case 1)*

- At the *input* level, the reasoner should achieve isomorphism equivariance, recognizing every element of $[G]_\sim$ as a single object rather than as a family of unrelated inputs and producing outputs that transform consistently with the serialization.
- At the *representation* level, the reasoner should recover the graph's topology from the linear token stream, identifying which pairs of tokens refer to adjacent nodes even when their mentions are scattered far apart in the prompt.
- At the *reasoning* level, the reasoner should produce the reasoning trace that faithfully executes the algorithm the task demands, so that the final answer is the consequence of a correct computation rather than of surface pattern matching.

These requirements form a hierarchy. Without recognizing isomorphic inputs, a reasoner cannot fully understand and recover graph topology from textual prompts, and without a faithful topological representation, its reasoning traces on $\mathcal{T}$ are not reliable. The formal definition of graph understanding is provided in Appendix B.1.

Can current LLMs achieve universal graph understanding? The three requirements above are not jointly satisfied by LLMs even on the simplest task. Table 1 presents three equivalent serializations of the same six-node Connectivity instance to Llama-3.1-8B-Instruct [2]. The three prompts are related by pure relabeling and reordering operations that preserve the ground-truth answer under the corresponding permutation of the query. Yet the model produces three qualitatively different outputs, each violating a distinct requirement from Fig. 1. Case 2 violates the *input-level* requirement, where relabeling the node set together with shuffling the edge list flips the predicted answer from NO to YES, even though the underlying graph and query are unchanged. The trace exposes the deeper *representation-level* failure that drives this error, where the LLM performs a correct reachability expansion from each query node, isolates the two disjoint components $\{0,2,4\}$ and $\{1,3,5\}$, and then declares them "connected through the edge list" without identifying any connecting edge. The structural separation between the two components, immediately evident from the graph itself, has been lost in the linear token stream. Case 3 reaches the correct answer but via a trace whose algorithmic skeleton differs from that of Case 1, an instance of *reasoning-level* nonisomorphism in which a mere reordering of the edge list induces a different sequence of algorithmic events for an identical

underlying computation. These cases show that all three requirements can be violated simultaneously, motivating architectural and training interventions on LLMs.

3 The TRACER Framework

In this section, we introduce TRACER, a post-training framework that addresses the three requirements of universal graph understanding through three coupled components built on the autoregressive Transformer backbone of LLMs. (1) *Topology-aware Residual Attention* (TRA) injects graph topology into the self-attention kernel, so that attention reflects the structural relationship between referenced entities rather than token positions alone (representation level). (2) *Permutation-invariant graph encoding* couples multi-augmentation supervised fine-tuning (SFT) with a group-relative RL objective, encouraging answer equivariance and trace isomorphism across $[G]_\sim$ (input level). (3) *Algorithmic trace supervision with process rewards* trains the model to emit faithful step-by-step traces under a learned verifier (reasoning level). TRA is described first because it modifies the attention kernel and serves as the substrate on which the other two components are trained.

3.1 Topology-aware Residual Attention (TRA)

The representation-level failure stems from a structural mismatch between what self-attention measures and what graph reasoning requires. Standard attention scores $a(i, j)$ depend only on token positions i and j, yet two tokens that refer to adjacent graph nodes may sit arbitrarily far apart in the prompt, and two tokens that sit adjacent in the prompt may refer to unrelated nodes. TRA closes this gap by adding a graph-conditioned residual to the attention logits, so that the effective attention between positions i and j is shaped not only by their token-level compatibility but also by the topological relationship between the graph entities they reference.

Specifically, let T denote the length of the tokenized prompt $x = [\sigma(G);\, q]$, and $[T] = \{1, \ldots, T\}$ index its token positions. We write $u = x_i \in \mathcal{V} \cup \{\perp\}$ for the graph node $u \in \mathcal{V}$ referenced by the token at position $i \in [T]$ in the prompt, with $\perp$ for tokens that do not correspond to a node mention (task definition, query text, punctuation, and other prompt scaffolding). For node u, let $\mathcal{N}^{(\rho)}(u)$ be the neighbor set of u within ρ hops in the graph. Given a token position j that refers to node v, i.e., $x_j = v$, we use $D_G(x_i, x_j)$ to denote the distance between u and v in the original graph, and set $D_G(x_i, x_j) = \infty$ when $v \notin \mathcal{N}^{(\rho)}(u)$ in G. Then we define the structure bias matrix $\mathbf{B} \in \mathbb{R}^{T\times T}$ at the token level to preserve graph topology:

$$\mathbf{B}_{ij} \;=\; \begin{cases} \varphi_{\text{same}}, & \text{if } x_i = x_j \neq \perp, \\ \varphi_{\text{dist}}\big(D_G(x_i, x_j)\big), & \text{if } x_i \neq x_j, \text{ both} \neq \perp, x_j \in \mathcal{N}^{(\rho)}(x_i) \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

where $\varphi_{\text{same}} \in \mathbb{R}$ is a learned scalar that couples coreferent mentions of the same node and $\varphi_{\text{dist}} : \{1, \cdots, \rho\} \to \mathbb{R}$ assigns learned weights to node pairs within ρ hops. Thus, non-graph tokens receive no structural residual. A deterministic preprocessing step parses the serialized graph, aligns node mentions with tokens, and constructs the local-hop relations in Eq. (2). These relations expose structure already specified by the input. The LLM learns how to integrate them with token representations and still generates the task-level reasoning trace and answer. In this work, we set $\rho = 2$, which is sufficient to capture graph locality while maintaining computational efficiency, because many graph algorithms propagate information through repeated local edge or frontier updates, and stacked Transformer layers can compose these local biases into longer-range reasoning without giving the model an expensive all-pairs distance table. We then inject $\mathbf{B}$ as a residual bias in the attention logits at each attended layer ℓ and head h:

$$\mathbf{A}^{\ell,h} \;=\; \text{softmax}\left(\frac{\mathbf{Q}^{\ell,h}(\mathbf{K}^{\ell,h})^\top}{\sqrt{d_k}} \;+\; \alpha^{\ell,h}\,\mathbf{B}^h\right), \tag{3}$$

where $\alpha^{\ell,h} \in \mathbb{R}_{\geq 0}$ is a learned per-layer, per-head gating scalar and $\mathbf{B}^h$ is a per-head instantiation of Eq. (2) with head-specific parameters $\varphi^h_{\text{same}}, \varphi^h_{\text{dist}}$. The per-layer, per-head parameterization lets the optimizer decide where and how strongly the structural bias should influence attention.

The Flash-Attention bottleneck and compatibility. Applied naively, Eq. (3) is incompatible with the IO-aware tiled attention kernels that modern fine-tuning pipelines depend on. Flash-Attention [11]

achieves $O(T)$ memory precisely by avoiding the full $T \times T$ logit matrix, but adding a graph-dependent bias $\mathbf{B}$ would require materializing that matrix in high-bandwidth memory (HBM), which undoes the central design choice of the kernel. Built-in bias support in Flash-Attention2 [10] handles only biases whose values are computable from position indices per tile, such as the linear distance bias of ALiBi [29], but Eq. (2) depends on per-input graph topology and cannot be reduced to a positional function. Without a more careful formulation, TRA would force a return to the quadratic-memory attention kernel, making SFT prohibitively expensive. We resolve this bottleneck by absorbing the structural bias into the augmented $\mathbf{Q}$ and $\mathbf{K}$ via a low-rank factorization. Specifically, $\mathbf{B}$ is determined by the entity assignment, which is shared across tokens of the same node, together with a small node-level bias matrix. Let $\mathbf{E} \in \{0,1\}^{T \times n}$ be the entity indicator matrix with $\mathbf{E}_{iu} = 1$ iff $x_i = u$, and let $\mathbf{\Phi}^h \in \mathbb{R}^{n \times n}$ be the node-level bias induced by Eq. (2). Then $\mathbf{\Phi}^h_{uv} = \varphi^h_{\text{same}}$ when $u = v$, $\mathbf{\Phi}^h_{uv} = \varphi^h_{\text{dist}}(D_G(u,v))$ when $v \in \mathcal{N}^{(\rho)}(u)$, and $\mathbf{\Phi}^h_{uv} = 0$ otherwise, giving:

$$\mathbf{B}^h \;=\; \mathbf{E}\,\mathbf{\Phi}^h\,\mathbf{E}^\top. \tag{4}$$

Since $n \ll T$, $\mathbf{B}^h$ is a heavily rank-deficient object represented as a $T \times T$ matrix. Suppose $\mathbf{\Phi}^h \approx \mathbf{L}^h(\mathbf{R}^g)^\top$ with $\mathbf{L}^h, \mathbf{R}^g \in \mathbb{R}^{n \times r}$, where g is the GQA [3] key/value group associated with attention head h. The factorization is exact when r is large enough to represent $\mathbf{\Phi}^h$, and it can be compressed for larger graphs. Omitting layer, head, and group indices for readability, we append the structural factors to the query and key as:

$$\tilde{\mathbf{Q}} \;=\; \left[\tfrac{1}{\sqrt{d_k}}\mathbf{Q},\; \sqrt{\alpha}\,\mathbf{E}\mathbf{L}\right] \;\in\; \mathbb{R}^{T \times (d_k + r)}, \quad \tilde{\mathbf{K}} \;=\; \left[\mathbf{K},\; \sqrt{\alpha}\,\mathbf{E}\mathbf{R}\right] \;\in\; \mathbb{R}^{T \times (d_k + r)}. \tag{5}$$

A direct calculation of TRA gives $\tilde{\mathbf{Q}}\tilde{\mathbf{K}}^\top \;=\; \mathbf{Q}\mathbf{K}^\top/\sqrt{d_k} + \alpha\,\mathbf{E}\mathbf{L}\mathbf{R}^\top\mathbf{E}^\top$, which exactly recovers the logits in Eq. (3) when $\mathbf{L}\mathbf{R}^\top = \mathbf{\Phi}$. The augmented $\tilde{\mathbf{Q}}$ and $\tilde{\mathbf{K}}$ are then fed through the standard Flash-Attention kernel with $\mathrm{softmax_scale} = 1$, with no $T \times T$ matrix ever materialized, and the IO complexity of the kernel is preserved. The exact local-hop factorization establishes the representability of the structural bias (see Appendix B.2). In practice, we employ learned rank-8 landmark factors $\mathbf{L}^h$ and $\mathbf{R}^g$ to efficiently capture graph-structural relationships through compact query-key augmentations while preserving compatibility with Flash-Attention. The construction is also compatible with Grouped-Query Attention [3], where the Q-side local-hop factor is head-specific, while the K-side structural key space is shared within each key/value group, mirroring the backbone's content projections. We further analyze the efficiency of TRA in Appendix E.1, and a full description of the implementation (e.g., φ) is deferred to Appendix C.

3.2 Permutation-Invariant Graph Encoding

TRA equips the LLM with a mechanism for reading graph topology, but does not by itself ensure that the model treats every element of $[G]_\sim$ as the same underlying object. Recall from Section 2 that the input-level requirement has two facets: Eq. (1) demands *answer equivariance* across serializations, and the *trace isomorphism* requirement further demands that the reasoning paths produced from different serializations agree once mapped back to a common label space. The two differ in kind. The former is expressible as an example-level loss and can be taught by SFT, whereas the latter is a relational constraint between outputs of the same graph and requires an objective that couples those outputs. `TRACER` operationalizes both through a two-stage procedure, described in turn.

For each training instance $(G, q, \tau^\star, a^\star)$ with ground-truth trace $\tau^\star$ and answer $a^\star$, we draw k augmentations $\{\sigma_1(G), \ldots, \sigma_k(G)\} \subset [G]_\sim$ by independently sampling $\{\pi_n^{(i)}, \pi_e^{(i)}, \pi_a^{(i)}\}$ from the uniform distribution over each group for the i-th augmentation. The ground-truth trace, answer, and query are then rewritten under the node relabeling $\pi_n^{(i)}$ into $(\tau_i^\star, a_i^\star)$, as illustrated by the transition from Case 1 to Case 2 in Table 1. This yields an augmentation set $\{(\sigma_i(G), q_i, \tau_i^\star, a_i^\star)\}_{i=1}^k$ whose elements all describe the same graph, with $\pi_n^{(i)}$ providing the bijection between any two of them.

Stage 1: multi-augmentation SFT for answer equivariance. Given the augmentation set, we combine two supervised terms. For each augmentation $\sigma_i(G)$ with prompt $x_i = [\sigma_i(G); q_i]$, let $y_i^\star = [\tau_i^\star; a_i^\star] = \left(y^\star_{i,1}, \ldots, y^\star_{i,M_i}\right)$ denote the target sequence of length M_i formed by concatenating the ground-truth trace and answer tokens. The task loss for G is the standard token-level cross-entropy of $\mathcal{M}_\theta$ on the target sequence, averaged over augmentations:

$$\mathcal{L}_{\text{task}}(G, q) = \frac{1}{k}\sum_{i=1}^{k}\sum_{t=1}^{M_i} -\log p_\theta\left(y^\star_{i,t} \mid y^\star_{i,<t}, x_i\right), \tag{6}$$

where p_θ is the next-token distribution induced by $\mathcal{M}_\theta$ and the loss is applied only to target tokens, with prompt tokens masked. Minimizing Eq. (6) teaches $\mathcal{M}_\theta$ to produce a valid trace and answer for every serialization, which is important for answer equivariance. To encourage the LLM to internally treat $[G]_\sim$ as the same graph and thereby enhance the permutation-invariant understanding, we insert a designated summary token [GRAPH] at the end of the graph description block in the prompt and extract its last-layer hidden state $h_i \in \mathbb{R}^d$, which serves as a natural summary of everything the model has read about $\sigma_i(G)$ up to that point. We then penalize representational drift across k augmentations with an anchor-based cosine distance:

$$\mathcal{L}_{\text{repr}}(G, q) = \frac{1}{k-1} \sum_{i=2}^{k} \text{CosineDistance } (h_1, h_i)\,, \tag{7}$$

where σ_1 serves as the anchor. The anchor formulation reduces the number of pairwise comparisons from $O(k^2)$ to $O(k)$ while yielding gradients essentially equivalent to those of the all-pairs form, because the anchor itself is updated at every step. The full SFT objective is:

$$\mathcal{L}_{\text{SFT}}(G, q) = \mathcal{L}_{\text{task}}\,(G, q) + \lambda_{\text{repr}} \mathcal{L}_{\text{repr}}(G, q). \tag{8}$$

Stage 2: RL for trace isomorphism. SFT teaches $\mathcal{M}_\theta$ to imitate the training traces by memorizing the mapping from each training serialization to its target trace, however, it does not yield trace isomorphism, which is a relational property between outputs of different serializations that an SFT loss factorizing over examples cannot express. What we need is a **policy** that produces structurally consistent traces across all of $[G]_\sim$, including unseen serializations, and the natural way to shape a policy is reinforcement learning. We thus refine the SFT checkpoint with Group Relative Policy Optimization (GRPO) [36], which samples a group of rollouts per prompt and computes advantages within the group. Specifically, for a training graph G, we sample several serializations, draw multiple rollouts (traces) from the current policy under each serialization, and score every rollout with two rewards: a correctness reward $R_{\text{correct}} \in \{0, 1\}$ indicating whether the extracted answer matches the ground-truth, and a consistency reward $R_{\text{consis}}(\tau_i^{(j)}) \in [0, 1]$ measuring structural agreement of traces across serializations. The consistency reward introduces the cross-serialization coupling that SFT cannot express, and it is not a group-level constant but varies across rollouts within the GRPO group, so the centered advantage retains the term $\lambda_c(R_{\text{consis}}(\tau_i^{(j)}) - \bar{R}_{\text{consis},i})$. A rollout that is individually correct but structurally inconsistent with its peers receives a lower advantage, pushing the policy toward a single canonical reasoning strategy that is invariant by construction rather than by memorization. The details of the trace-isomorphism RL are provided in Appendix B.3.

3.3 Algorithmic Trace Supervision with Process Rewards

To address the reasoning-level graph understanding, which concerns what appears inside the reasoning traces, we introduce a Process Reward Model (PRM) that provides step-level supervisory signal during RL training. The PRM $\mathcal{F}_\phi$ can be a small trained verifier (e.g., a 0.5B decoder-only model) that scores individual trace steps on state-update correctness, i.e., whether each intermediate reasoning state described in the trace is faithful to the underlying graph. $\mathcal{F}_\phi$ is trained on step-level supervision derived programmatically from the ground-truth traces $\tau^\star$. For each intermediate state described in $\tau^\star$, we extract the expected post-step state from the graph itself and construct positive and negative step-level labels by comparing candidate trace steps against these references. This yields a binary classification dataset over (context, step) pairs on which $\mathcal{F}_\phi$ is fine-tuned. At RL time, $\mathcal{F}_\phi$ scores each intermediate state in a rollout trace τ_i and returns the trace-averaged prediction as $R_{\text{PRM}}(\tau_i)$, which enters the compound reward alongside R_{correct} and R_{consis}.

3.4 Overall Post-training Pipeline

TRACER is trained in two phases on a TRA-augmented LLM backbone to improve graph topology understanding. The first phase performs SFT on a multi-augmented corpus by optimizing Eq. (8) to enhance understanding of isomorphism equivariance. The second phase refines the SFT checkpoint with reinforcement learning under the compound reward combining a correctness term R_{correct}, the cross-permutation consistency reward R_{consis}, and the process reward R_{PRM} to encourage trace isomorphism and algorithmic faithfulness at the reasoning level. Together, the two stages address all three universal graph understanding requirements of Section 2 on a single backbone.

Table 2: Accuracy (%) on the nine graph computational tasks. The best result is **bolded** and the second-best is underlined.

	Easy (Linear)				Medium (Poly.)			Hard (NP-c.)		
Method	Cycle	Connect.	Bipartite	Topo.	Shortest	Triangle	Flow	Hamilton	Subgraph	**Avg.**
GPT-4 (0-shot)	38.75	17.00	65.25	5.00	9.25	5.75	3.25	59.25	45.50	27.67
GPT-3.5 (2-shot)	51.25	43.75	70.75	4.50	3.50	17.25	8.50	54.25	43.00	32.97
GPT-4 (2-shot)	52.50	62.75	74.25	25.25	18.25	31.00	7.75	**75.75**	46.75	43.81
Llama-3.1-8B-Instruct	64.75	81.00	58.75	11.50	3.50	4.25	9.25	19.25	45.00	33.03
Qwen3-8B	79.25	75.50	42.25	6.50	13.25	31.75	11.25	72.75	62.75	43.92
GCN	84.00	74.00	82.00	–	5.75	6.75	9.25	–	68.00	–
GIN	87.50	73.00	85.25	–	7.25	7.30	12.00	–	66.50	–
GAT	87.50	79.25	85.25	–	7.25	7.50	12.50	–	66.25	–
SFT (Mistral-7B)	73.75	83.50	78.50	1.00	23.00	47.00	28.75	31.75	41.25	45.39
SFT (Llama-3.1-8B-Instruct)	75.50	83.50	76.75	53.75	24.50	45.50	32.75	23.75	75.00	54.56
GraphWiz-RFT	88.75	89.50	78.25	22.75	30.25	40.75	28.50	28.25	80.75	54.19
GraphWiz-DPO	87.50	82.25	88.75	45.75	24.25	52.75	45.00	30.50	77.25	59.33
G1	77.75	89.25	93.75	31.50	11.00	22.25	42.00	23.50	58.25	49.92
TRACER-SFT	90.00	89.50	**97.25**	63.75	23.50	53.00	43.75	31.25	77.75	63.31
TRACER	**93.75**	**91.50**	95.50	**64.25**	**40.50**	**59.50**	**46.25**	33.50	**81.00**	**67.31**

4 Experiments

We evaluate TRACER on the GraphInstruct benchmark [8], which defines nine graph computation tasks spanning three complexity classes: linear-time tasks (Connectivity, Cycle Detection, Bipartite Checking, Topological Sort), polynomial-time tasks (Shortest Path, Maximum Triangle Sum, Maximum Flow), and NP-complete tasks (Hamiltonian Path, Subgraph Matching). Each test instance consists of a textual graph description, a task query, and a ground-truth answer. Following the evaluation protocol of [8], we report exact-match accuracy on the designated answer span for every task.

We compare against four groups of baselines: (1) Closed-source LLMs: GPT-4 [1] in zero-shot and 2-shot CoT settings, and GPT-3.5 with 2-shot CoT. (2) Open-source LLMs: Llama-3.1-8B-Instruct [2] and Qwen3-8B [30]. (3) Graph Neural Networks: GCN [20], GIN [48], and GAT [42], which operate on the graph structure directly but are restricted to the subset of tasks expressible as node or graph classification (Topological Sort and Hamiltonian Path are excluded). (4) Post-trained LLMs: Naive SFT [18, 2], GraphWiz-RFT, GraphWiz-DPO [8], and G1 [16]. Full details of the benchmark and experimental setup are provided in Appendix D. Additional experiments are provided in Appendix E.

4.1 Main Results

Table 2 reports the test accuracy, where TRACER-SFT denotes the SFT-trained variant and TRACER denotes the full model obtained by further applying RL after SFT. TRACER achieves the strongest overall performance, obtaining an average accuracy of 67.31%, outperforming the best prior all-task baseline GraphWiz-DPO by 7.98%. Even before RL, TRACER-SFT reaches 63.31%, already surpassing baselines by 3.98%, indicating that TRA and multi-serialization SFT provide a strong structural inductive bias for graph computation. Moreover, the gains are consistent across task families. On linear-time tasks, TRACER achieves the best results on Cycle, Connect., and Topo., while TRACER-SFT attains the best results on Bipartite. The improvement is particularly pronounced on Topo., where TRACER reaches 64.25%, substantially exceeding the strongest baseline. On polynomial tasks, TRACER obtains the best accuracy on all three tasks. These results suggest that explicitly biasing attention toward graph topology helps the model recover algorithmic structure that is poorly captured by standard token-level self-attention. On NP-c tasks, TRACER improves over post-trained LLM baselines. Although Hamiltonian Path remains challenging and closed-source or stronger general-purpose LLMs perform better on this specific task, TRACER still improves over GraphWiz-DPO and G1 among post-trained graph-reasoning models. Overall, the results demonstrate that TRACER provides a more effective and general graph computation interface for LLMs than prompting, post-training, and graph neural baselines, especially on tasks requiring topology-sensitive reasoning.

Table 3: Scaling TRACER to a 3B backbone.

Method	Easy (Linear)				Medium (Poly.)			Hard (NP-c.)		
	Cycle	Connect.	Bipartite	Topo.	Shortest	Triangle	Flow	Hamilton	Subgraph	Avg.
Llama-3.2-3B-Instruct	32.00	53.75	25.75	7.50	2.75	3.75	2.50	38.25	12.00	19.81
TRACER-SFT-3B	82.00	81.50	**96.00**	**35.50**	18.25	52.00	27.25	**42.50**	**70.75**	56.19
TRACER-3B	**86.00**	**84.75**	**96.00**	35.00	**26.50**	**54.50**	**36.50**	39.50	67.00	**58.42**

Table 4: Held-out-task transfer.

Method	Topo.	Flow	Subgraph	Avg.
Naive SFT	28.25	15.25	53.25	32.25
TRACER-SFT	41.50	25.75	62.00	43.08
TRACER	**51.00**	**32.50**	**66.50**	**50.00**

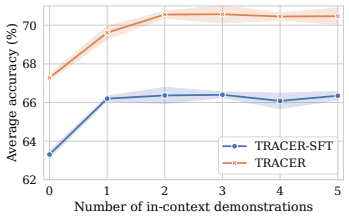

Figure 4: Few-shot performance of TRACER.

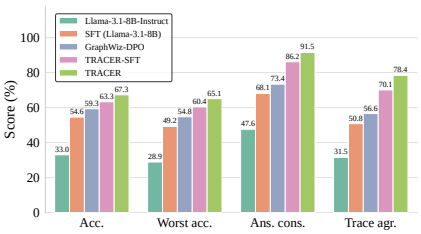

Figure 2: Serialization robustness.

Robustness Analysis. A key claim of TRACER is that equivalent serializations of the same graph should induce consistent answers and structurally similar traces. We therefore evaluate each test graph under multiple randomly sampled serializations from $[G]_\sim$. In addition to standard accuracy on the original serialization, we report three robustness metrics: *worst-serialization accuracy*, the average accuracy of the least favorable serialization for each graph; *answer consistency*, the fraction of graphs for which all sampled serializations produce equivalent final answers after canonicalizing node labels; and *trace agreement*, the longest common subsequence (LCS) score between normalized reasoning traces. Fig. 2 shows that TRACER is substantially more stable than baselines. The gap between TRACER-SFT and TRACER reveals that SFT already improves consistency through permutation augmentation and representation alignment, but the RL stage further increases trace agreement.

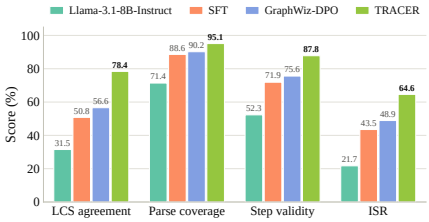

Figure 3: Trace agreement and symbolic validity.

Symbolic Evaluation of Trace Validity. To assess algorithmic validity independently of textual agreement, we segment generated traces at sentence boundaries, parse each step into a typed event, and replay the events with a task-specific nondeterministic reference automaton. Events include node visits, frontier expansions, and edge relaxations. A transition is accepted when its resulting state is a legal successor, allowing multiple admissible execution orders. We report parse coverage, step validity, and the invariant satisfaction rate (ISR), the percentage of traces for which every step is parsed and legal. Unparseable steps count as violations. Fig. 3 shows that TRACER improves both step validity and ISR. Its ISR of 64.6% is below its LCS agreement of 78.4%, illustrating that agreement and valid execution measure different properties. Two valid executions can follow different admissible orders, and similar traces can share an invalid transition.

Smaller Backbones. To demonstrate the effectiveness of TRACER on smaller LLMs, we apply the same training recipe to Llama-3.2-3B-Instruct. Table 3 reports the results, which show that TRACER transfers effectively to the 3B setting. SFT substantially improves the base model, while RL further boosts average performance, especially on the Hamiltonian Path and Bipartite Checking tasks, enabling the 3B model to rival or surpass stronger 8B baselines like GraphWiz and G1.

4.2 Few-shot Performance

We study the effect of in-context demonstrations [56] by prepending $k \in \{0, 1, 2, 3, 4, 5\}$ solved examples to the prompt. As shown in Fig. 4, few-shot prompting consistently improves performance over the zero-shot setting, with accuracy increasing from 0-shot to 1–2 shots. This suggests that only a few examples are sufficient to elicit the model's graph reasoning ability.

4.3 Transfer to Held-out Tasks

To test whether TRACER learns reusable graph-computation primitives rather than memorizing task templates, we design a held-out-task transfer experiment: train on all tasks except one target task and evaluate on the held-out task using the target task's prompt template. Topo., Flow, and Subgraph are particularly useful held-out targets because they test node-list equivariance, weighted directed

Table 5: Ablation of TRACER components. The third row isolates RL with trace-isomorphism consistency rewards before adding the process reward model.

	Easy (Linear)				Medium (Poly.)			Hard (NP-c.)		
Variant	**Cycle**	**Connect.**	**Bipartite**	**Topo.**	**Shortest**	**Triangle**	**Flow**	**Hamilton**	**Subgraph**	**Avg.**
GraphWiz-RFT	88.75	89.50	78.25	22.75	30.25	40.75	28.50	28.25	80.75	54.19
TRACER-SFT	90.00	89.50	**97.25**	63.75	23.50	53.00	43.75	31.25	77.75	63.31
TRACER-SFT + trace-iso RL	**93.75**	91.00	96.50	62.75	33.75	51.75	45.75	33.00	**81.00**	65.47
TRACER	**93.75**	**91.50**	95.50	**64.25**	**40.50**	**59.50**	**46.25**	**33.50**	**81.00**	**67.31**

Table 6: Reported metrics by graph-size range.

Nodes	**Prompt tokens**	**Decoding tokens**	**Prep. (ms)**	**Latency (s)**	**Memory (GB)**	**Acc. (%)**
2–25	285	135	0.35	2.1	18.6	76.8
26–50	505	225	0.68	3.5	19.0	70.4
51–75	745	350	1.12	5.4	19.6	62.3
76–100	960	505	1.70	7.6	20.3	54.8

Table 7: Accuracy (%) after training only on graphs with at most 50 nodes. The last two columns evaluate unseen size ranges.

Method	**26–50 nodes**	**51–75 nodes**	**76–100 nodes**
Naive SFT	56.0	43.5	32.1
TRACER-SFT	64.2	58.9	49.3
TRACER	**68.5**	**61.4**	**51.7**

reasoning, and subgraph matching, respectively. Table 4 shows that TRACER achieves the best transfer performance on all held-out tasks. The gains are most pronounced on Topo. and Flow, suggesting that topology-aware representation and post-training induce transferable primitives for order-invariant traversal and directed weighted propagation, rather than merely fitting task-specific templates.

4.4 Graph Size, Density and Size Extrapolation

We evaluate graph-size and density sweeps. Table 6 reports accuracy, prompt and decoding lengths, topology-preprocessing time, end-to-end latency, and peak GPU memory. As graph size grows, token counts and latency increase while accuracy declines. In these measurements, preprocessing remains below 2 ms, so the larger latency increase comes primarily from model computation and decoding. At a fixed size of 50 nodes, increasing the average degree from 2 to 12 raises the prompt length from 345 to 980 tokens and latency from 3.0 to 5.6 s. Topology preprocessing increases from 0.50 to 1.36 ms, while accuracy falls from 72.1% to 62.4%. The density sweep thus shows the same combination of longer serializations, higher model cost, and reduced accuracy.

To test size extrapolation, we restrict training to graphs with $|\mathcal{V}| \leq 50$ and evaluate larger held-out graphs without further training, keeping task definitions and graph generation unchanged. Table 7 shows that TRACER retains higher accuracy than both SFT variants in the two unseen size ranges.

4.5 Ablation Study

We ablate the components of TRACER to separate the effect of task-only supervision $\mathcal{L}_{\text{task}}$, serialization-aware SFT, trace-isomorphism RL in Section 3.2, and process-reward RL in Section 3.3. The first variant optimizes only the task supervision term $\mathcal{L}_{\text{task}}$ and corresponds to GraphWiz-RFT's task-SFT baseline reported in Table 2. The second variant, TRACER-SFT, uses the TRA-augmented backbone with the SFT objective as Eq. (8). The third variant starts from TRACER-SFT and applies RL with trace-isomorphism rewards, but does not use the process reward model. The last variant is the full TRACER model, which further includes R_{PRM}. Table 5 shows that the full model obtains the best average accuracy, while also making clear that the contribution of each component is not uniform across tasks. These results suggest that the three stages are complementary, with each contributing to different aspects of graph-solving behavior. More fine-grained ablation experiments are in Appendix E.2.

5 Conclusion

We present TRACER, a post-training framework for using LLMs as graph computational solvers. TRACER addresses graph reasoning failures by aligning LLM computation with three structural requirements of graph algorithms: permutation invariance, topology awareness, and trace-level procedural consistency. At the input and representation levels, TRACER combines permutation-invariant graph encoding with Topology-aware Residual Attention to reduce sensitivity to graph serialization and inject topological information into self-attention. At the reasoning level, process-reward RL encourages intermediate reasoning steps to follow valid algorithmic execution. Experiments demonstrate that TRACER achieves superior performance across graph tasks of varying difficulty.

Acknowledgments and Disclosure of Funding

This research / project is supported by the National Research Foundation, Singapore, under its Frontier CRP Grant (NRF-F-CRP-2024-0005), and under its AI Singapore Programme (AISG Award No: AISG3-RP-2024-034). This research is also supported by Singapore MOE AcRF Tier-2 Award (MOE-T2EP20224-0005) and NTU SUG-NAP. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the National Research Foundation, Singapore.

References

[1] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] AI@Meta. Llama 3 model card. 2024. URL `https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md`.

[3] Ainslie, J., Lee-Thorp, J., De Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023.

[4] Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[5] Belrose, N., Ostrovsky, I., McKinney, L., Furman, Z., Smith, L., Halawi, D., Biderman, S., and Steinhardt, J. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

[6] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[7] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[8] Chen, N., Li, Y., Tang, J., and Li, J. Graphwiz: An instruction-following language model for graph computational problems. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 353–364, 2024.

[9] Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[10] Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=mZn2Xyh9Ec`.

[11] Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Re, C. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=H4DqfPSibmx`.

[12] Fatemi, B., Halcrow, J., and Perozzi, B. Talk like a graph: Encoding graphs for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=IuXR1CCrSi`.

[13] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. Pmlr, 2017.

[14] Gugger, S., Debut, L., Wolf, T., Schmid, P., Mueller, Z., Mangrulkar, S., Sun, M., and Bossan, B. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate, 2022.

[15] Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., Xu, R., Zhang, R., Ma, S., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[16] Guo, X., Li, A., Wang, Y., Jegelka, S., and Wang, Y. G1: Teaching LLMs to reason on graphs with reinforcement learning. In *1st ICML Workshop on Foundation Models for Structured Data*, 2025. URL https://openreview.net/forum?id=nCcZgxQD2P.

[17] Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[18] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[19] Jiang, J., Wang, F., Shen, J., Kim, S., and Kim, S. A survey on large language models for code generation. *ACM Transactions on Software Engineering and Methodology*, 35(2):1–72, 2026.

[20] Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[21] Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByxBFsRqYm.

[22] Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. Rethinking graph transformers with spectral attention. *Advances in neural information processing systems*, 34: 21618–21629, 2021.

[23] Li, Z., Jiang, G., Xie, H., Song, L., Lian, D., and Wei, Y. Understanding and patching compositional reasoning in llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 9668–9688, 2024.

[24] Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

[25] Lin, C.-Y. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pp. 74–81, 2004.

[26] Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Syx72jC9tm.

[27] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[28] Park, J., Chun, J., Kim, S. H., Kim, Y., and Park, J. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. *International journal of production research*, 59(11):3360–3377, 2021.

[29] Press, O., Smith, N., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

[30] Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[31] Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

[32] Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC20: international conference for high performance computing, networking, storage and analysis*, pp. 1–16. IEEE, 2020.

[33] Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

[34] Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 3505–3506, 2020.

[35] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[36] Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[37] Shaw, P., Uszkoreit, J., and Vaswani, A. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 464–468, 2018.

[38] Tang, J., Zhang, Q., Li, Y., Chen, N., and Li, J. 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.

[39] Tenney, I., Das, D., and Pavlick, E. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4593–4601, 2019.

[40] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[41] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[42] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

[43] Wang, H., Feng, S., He, T., Tan, Z., Han, X., and Tsvetkov, Y. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36: 30840–30861, 2023.

[44] Wang, R., Liang, S., Chen, Q., Zhang, J., and Qin, K. Graphtool-instruction: Revolutionizing graph reasoning in llms through decomposed subtask instruction. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 1492–1503, 2025.

[45] Wei, C., Hu, W., Hao, X., Wang, X., Yang, Y., Wang, Y., Tian, Y., and Chen, Y. Graphchain: Large language models for large-scale graph analysis via tool chaining. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[46] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[47] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

[48] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

[49] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[50] Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? *Advances in neural information processing systems*, 34:28877–28888, 2021.

[51] Yun, C., Bhojanapalli, S., Rawat, A. S., Reddi, S., and Kumar, S. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxRM0Ntvr.

[52] Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[53] Zelikman, E., Wu, Y., Mu, J., and Goodman, N. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

[54] Zhang, J. Graph-toolformer: To empower llms with graph reasoning ability via prompt augmented by chatgpt. *arXiv preprint arXiv:2304.11116*, 2023.

[55] Zhao, Q., Ren, W., Li, T., Liu, H., He, X., and Xu, X. GraphGPT: Generative pre-trained graph eulerian transformer. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=4RdzeucFmW.

[56] Zhuo, W. and Luo, S. Modality-free graph in-context alignment. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=cDc95lucVL.

[57] Zhuo, W. and Tan, G. Efficient graph similarity computation with alignment regularization. *Advances in Neural Information Processing Systems*, 35:30181–30193, 2022.

A Related Work

LLMs for Graph Computation. Early benchmarks such as NLGraph [43] evaluated whether prompted LLMs can solve graph problems described in natural language and reported sensitivity to task complexity and textual artifacts. GraphInstruct and GraphWiz [8] then introduced instruction data and preference tuning for explicit graph-computation tasks, while GraphArena broadened evaluation to polynomial-time and NP-complete graph computations [38]. Another line studies graph-to-text or graph-to-sequence interfaces, including textual graph encodings for frozen LLMs [12] and Eulerian graph tokenization for graph pretraining [55]. Tool-augmented systems instead prompt LLMs to call graph APIs, decompose tool arguments, or chain graph operations at test time [54, 44, 45]. G1 [16] is closest in using RL to improve LLM graph reasoning, but it trains on synthetic graph-theoretic tasks rather than modifying attention with an explicit topology bias and cross-serialization trace objective. These works motivate graph-aware LLMs, but most either evaluate fixed LLMs, optimize textual encodings, or delegate computation to tools. TRACER differs by making graph topology part of the LLM's internal attention computation and by training answers and traces to agree across equivalent serializations.

Graph-aware Attention and Invariance. Transformers for sequences incorporate structure through relative-position or linear attention biases [41, 37, 29, 57], while graph Transformers show that shortest-path encodings, spectral encodings, and local-global attention can improve graph representation learning [50, 22, 33]. More general invariant and equivariant architectures formalize permutation symmetries for sets and graphs [52, 26]. TRACER draws on this principle but targets a different setting, where a pretrained LLM receives the graph as text and node mentions occupy arbitrary token positions. Existing graph Transformers and equivariant networks are not designed to make multiple textual serializations of one graph induce compatible token-level attention and reasoning traces. TRACER differs by adding a topology-aware residual to LLM self-attention and aligning graph-summary representations across serializations.

LLM Post-training. Post-training turns a pretrained LLM into an instruction-following or task-specialized policy through supervised demonstrations, preference optimization, and reward-based policy updates: RLHF combines demonstration SFT, reward modeling [27], and PPO-style policy optimization [35], while DPO directly optimizes preferences without an explicit reward model or RL loop [31]. For reasoning, post-training has moved from outcome verifiers and rationale bootstrapping to step-level process supervision and RL with verifiable rewards [9], including STaR [53], process reward models for mathematical reasoning [24], DeepSeekMath's GRPO [36], and DeepSeek-R1's reasoning-oriented RL pipeline [15]. Graph-specific post-training is emerging as well: GraphWiz [8] applies instruction tuning and DPO to graph-computation tasks, and G1 [16] trains LLMs with RL on synthetic graph-theoretic tasks. The gap is that prior post-training methods generally optimize single-prompt correctness, preferences, or step quality, rather than coupling outputs from isomorphic graph serializations. TRACER differs by combining multi-serialization SFT, representation alignment, trace-isomorphism rewards, and process rewards on a topology-aware LLM backbone.

B Model Details

B.1 Formal Definition of Graph Understanding

Fix a domain of graph–query–task triples and a reference algorithm for each task. Let H_σ denote the LLM's internal state after processing the complete graph description under a specific serialization σ.

Definition 2 (Universal graph understanding)**.** *A model has universal graph understanding on the domain if the following hold for every admitted $(G, q, \mathcal{T})$ and every $\sigma \in [G]_\sim$:*

- *Input. Its answer is equivariant under serialization changes, as specified in Eq.* (1)*.*
- *Representation. A fixed readout g, independent of G and σ, recovers adjacency from H_σ, so that $g\left(H_\sigma, \pi_n^{(\sigma)}(u), \pi_n^{(\sigma)}(v)\right) = \mathbf{1}_{\{(u,v)\in\mathcal{E}\}}$ for every node pair u and v.*
- *Reasoning. With probability one, the canonicalized trace is a complete valid execution of the reference algorithm. It starts at $s_0 = \mathrm{init}(G, q)$, satisfies $s_{t+1} \in \Delta_{\mathcal{T}}(s_t; G, q)$ at*

each step, and ends at a terminal state whose readout equals the canonicalized reported answer $\left(\pi_n^{(\sigma)}\right)^{-1}(a_\sigma) \in \mathcal{A}_\mathcal{T}(G,q)$. *Here* a_σ *is the reported answer,* $\mathcal{A}_\mathcal{T}(G,q)$ *is the set of valid answers, and* $\Delta_\mathcal{T}$ *is the set of legal successor states, allowing multiple valid execution orders.*

The three conditions jointly define the target throughout the chosen domain. They concern complementary properties of the input behavior, internal representation, and generated reasoning. Serialization equivariance is needed for consistent answers across equivalent inputs, while topology recovery concerns the information retained internally.

B.2 Local-hop Bias Construction and Factorization

We provide the concrete construction of the node-level bias $\mathbf{\Phi}$ used in Eq. (4) and the factorization used to absorb it into Flash-Attention. Let $\mathbf{A} \in \{0,1\}^{n\times n}$ be the binary adjacency matrix under the task-specific graph convention: directed tasks preserve edge direction, while undirected tasks use the symmetrized adjacency relation. For $\rho = 1$, the retained local relation is the direct-edge mask $\mathbf{M}^{(1)} = \mathbf{A}$. For $\rho = 2$, we also construct the exact two-hop mask:

$$\mathbf{M}^{(2)}_{uv} \;=\; \begin{cases} 1, & \text{if } u \neq v,\ \mathbf{A}_{uv} = 0, \text{ and } \exists k \in \mathcal{V} \text{ such that } \mathbf{A}_{uk}\mathbf{A}_{kv} = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Thus direct neighbors keep hop type 1 even when they are also connected by a length-two walk, and self-pairs are reserved for the coreference parameter. For head h, the node-level local-hop bias can be written as:

$$\mathbf{\Phi}^h = \varphi^h_{\text{same}}\mathbf{I} + \varphi^h_{\text{dist}}(1)\mathbf{M}^{(1)} + \mathbf{1}_{\{\rho=2\}}\,\varphi^h_{\text{dist}}(2)\mathbf{M}^{(2)}. \tag{10}$$

Equivalently, $\mathbf{\Phi}^h_{uv} = \varphi^h_{\text{same}}$ for $u = v$, $\mathbf{\Phi}^h_{uv} = \varphi^h_{\text{dist}}(D_G(u,v))$ for retained one- or two-hop pairs, and 0 otherwise. All entries outside the diagonal and the retained local-hop masks are exactly zero. The masks are built by bounded adjacency-list expansion rather than all-pairs shortest-path computation. For $\rho = 1$, construction is just edge-list parsing and costs $O(n+m)$. For $\rho = 2$, for each edge $u \to k$ we iterate over the out-neighbors of k and mark $u \to v$ as a two-hop pair unless $v = u$ or $u \to v$ is already a direct edge. The cost is $O(\sum_u \sum_{k\in\mathcal{N}^+(u)} \deg^+(k)) \le O(m\Delta)$, where Δ is the maximum out-degree. This preprocessing is performed once per input graph and reused across all attended layers and heads.

At the benchmark scale, we use an exact factorization. For each GQA [3] key/value group g, set $\mathbf{R}^g = \mathbf{I}_n$, and for each head h in that group set $\mathbf{L}^h = \mathbf{\Phi}^h$, padding both factors to the batch maximum node count when graphs of different sizes are batched. Then,

$$(\mathbf{E}\mathbf{L}^h)(\mathbf{E}\mathbf{R}^g)^\top = \mathbf{E}\mathbf{\Phi}^h\mathbf{E}^\top = \mathbf{B}^h, \tag{11}$$

which reproduces Eq. (3) exactly while avoiding materialization of the token-level $T \times T$ bias. The Q-side factor $\mathbf{L}^h$ remains head-specific through φ^h_{same} and φ^h_{dist}, while the K-side factor $\mathbf{R}^g$ is shared within the GQA group.

B.3 GRPO Training for Trace Consistency

In this section, we provide the full algorithmic specification of the Stage 2 RL procedure for trace isomorphism summarized in Section 3.2. We detail the construction of the consistency reward, the per-serialization advantage estimator, and the PPO-style clipped policy update, followed by a brief analysis of why the design realizes the cross-serialization coupling that SFT cannot express.

Given a training graph G, we sample m serializations $\sigma_1(G), \ldots, \sigma_m(G)$, where each $\sigma_i = \sigma_{\pi_n^{(i)},\pi_e^{(i)},\pi_a^{(i)}}$ for $i \in \{1,\ldots,m\}$ is drawn by independently sampling a node relabeling $\pi_n^{(i)}$, an edge-list ordering $\pi_e^{(i)}$, and a set of adjacency-list orderings $\pi_a^{(i)}$ uniformly at random. For each serialization $\sigma_i(G)$, we draw n independent rollouts (traces) $\tau_i^{(1)}, \ldots, \tau_i^{(n)}$ from the current policy by sampling autoregressively from $\mathcal{M}_\theta(\cdot \mid \sigma_i(G), q_i)$ with temperature larger than 0. Different rollouts from the same prompt will generally differ because each token is sampled stochastically. These n completions form the group that GRPO uses to estimate advantages by centering rewards against the within-group mean, replacing the learned value network required by standard policy gradient methods. Every rollout is scored on two axes.

Correctness reward. For each rollout $\tau_i^{(j)}$, we extract the final answer $a_i^{(j)}$ from the designated answer span and compare it to the ground-truth answer $a_i^\star$ expressed in the relabeled node space of σ_i:

$$R_{\text{correct}}(\tau_i^{(j)}) \;=\; \begin{cases} 1, & \text{if } a_i^{(j)} = a_i^\star, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

Consistency reward. The consistency reward couples rollouts across different serializations, but it must be computed as a rollout-specific quantity so that it survives GRPO's within-group mean centering. For every sampled trace, we first normalize it to the canonical label space of G by inverting the permutations that define its serialization:

$$\tilde{\tau}_i^{(j)} = \sigma_i^{-1}(\tau_i^{(j)}).$$

Concretely, this inverts the node relabeling $\pi_n^{(i)}$ on all node references in the trace, the edge-list ordering $\pi_e^{(i)}$ on any edge enumerations, and the adjacency-list ordering $\pi_a^{(i)}$ on any neighbor listings. Since the prompt template for each task inherits the trace format from the GraphInstruct benchmark [8] and the model is trained during trace SFT by Eq. (6) to follow this format, the normalized traces are structured sequences of algorithmic steps (e.g., component expansions for Connectivity, edge relaxations for Shortest Path, candidate explorations for Hamiltonian Path) and can be compared directly without a separate event-extraction stage. We define

$$\text{align}(x, y) = \frac{\text{LCS}(x, y)}{\max(|x|, |y|)},$$

where LCS denotes Longest Common Subsequence [25]. Let $w_{\ell k} = R_{\text{correct}}(\tau_\ell^{(k)})$, so only correct rollouts from other serializations are used as references. The per-rollout consistency reward is

$$R_{\text{consis}}(\tau_i^{(j)}) = \begin{cases} R_{\text{correct}}(\tau_i^{(j)}) \dfrac{\displaystyle\sum_{\substack{\ell=1 \\ \ell \neq i}}^{m} \sum_{k=1}^{n} w_{\ell k} \; \text{align}\Big(\tilde{\tau}_i^{(j)}, \tilde{\tau}_\ell^{(k)}\Big)}{\displaystyle\sum_{\substack{\ell=1 \\ \ell \neq i}}^{m} \sum_{k=1}^{n} w_{\ell k}}, & \text{if } \displaystyle\sum_{\substack{\ell=1 \\ \ell \neq i}}^{m} \sum_{k=1}^{n} w_{\ell k} > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

This score lies in $[0, 1]$. The correctness gate prevents an incorrect rollout from receiving additional reward merely because its trace is similar to other traces, while the reference set encourages a correct rollout to follow a canonicalized trace structure that agrees with correct rollouts sampled from equivalent serializations.

The per-rollout total GRPO reward combines both terms to get the compound reward:

$$R_{\text{GRPO}}(\tau_i^{(j)}) \;=\; R_{\text{correct}}(\tau_i^{(j)}) \;+ \lambda_c \cdot R_{\text{consis}}(\tau_i^{(j)}), \tag{14}$$

where $\lambda_c > 0$ controls the relative weight of consistency. The consistency term is not shared across all rollouts of a graph. Instead, it varies across rollouts within the same GRPO group according to how well each canonicalized trace agrees with correct traces generated from other serializations.

Given the compound reward, we follow the standard GRPO procedure [36]. Within each serialization group i, advantages are computed by centering and normalizing $R_{\text{GRPO}}(\tau_i^{(j)})$ against the within-group mean and standard deviation, and the policy is updated with the clipped surrogate objective and a KL penalty against the SFT reference policy $\mathcal{M}_{\theta_{\text{ref}}}$. Because $R_{\text{consis}}(\tau_i^{(j)})$ is rollout-specific, the centered numerator contains

$$R_{\text{GRPO}}(\tau_i^{(j)}) - \bar{R}_i = \big(R_{\text{correct}}(\tau_i^{(j)}) - \bar{R}_{\text{correct},i}\big) + \lambda_c \big(R_{\text{consis}}(\tau_i^{(j)}) - \bar{R}_{\text{consis},i}\big), \tag{15}$$

so the consistency reward does not cancel unless all rollouts in the group have identical consistency scores. The permutation augmentations provide the cross-serialization reference pool for R_{consis}, while the n stochastic completions for each serialization provide the within-prompt group used by GRPO for relative advantage estimation.

Algorithm 1 GRPO for Trace Consistency Training

Require: Training graph set $\mathcal{G}$, number of permutations m, number of rollouts n, current policy $\mathcal{M}_\theta$, clipping parameter ϵ

for each training graph $G \in \mathcal{G}$ **do**
  Step 1: Sample m permutations: $\sigma_1(G), \ldots, \sigma_m(G)$
  Step 2: For each σ_i, sample n rollouts from $\mathcal{M}_\theta$: $\tau_i^{(1)}, \ldots, \tau_i^{(n)}$
  Step 3: Compute rewards:
  for $i = 1$ **to** m **do**
    for $j = 1$ **to** n **do**
      Compute $R_{\text{correct}}(\tau_i^{(j)})$ {Eq. (12)}
    end for
  end for
  Set reference weights $w_{ik} = R_{\text{correct}}(\tau_i^{(k)})$ for all i, k
  for $i = 1$ **to** m **do**
    for $j = 1$ **to** n **do**
      Normalize rollout: $\tilde{\tau}_i^{(j)} = \sigma_i^{-1}(\tau_i^{(j)})$
      Compute rollout-specific $R_{\text{consis}}(\tau_i^{(j)})$ {Eq. (13)}
      Compute $R_{\text{GRPO}}(\tau_i^{(j)}) = R_{\text{correct}}(\tau_i^{(j)}) + \lambda_c R_{\text{consis}}(\tau_i^{(j)})$ {Eq. (14)}
    end for
  end for
  Step 4: Within each permutation group, compute advantage: {GRPO [36]}
  for $i = 1$ **to** m **do**
    for $j = 1$ **to** n **do**
      $\hat{A}(\tau_i^{(j)}) = \big(R_{\text{GRPO}}(\tau_i^{(j)}) - \bar{R}_i\big) \,/\, (\sigma_{R,i} + \epsilon)$
    end for
  end for
  Step 5: Policy gradient update with clipping (PPO-style): {GRPO [36]}
    $\theta \leftarrow \theta + \alpha \nabla_\theta \sum_{i,j} \min\Big(r_\theta(\tau_i^{(j)})\, \hat{A}(\tau_i^{(j)}),\ \text{clip}\big(r_\theta(\tau_i^{(j)}),\, 1-\epsilon,\, 1+\epsilon\big)\, \hat{A}(\tau_i^{(j)}) \Big)$
end for

This reward favors rollouts that are both answer-correct and trace-consistent. A correct rollout whose canonicalized reasoning trace disagrees with correct traces from equivalent serializations receives a lower consistency component than a correct rollout whose trace follows the same structural execution pattern. Over policy updates, this supplies a learning signal toward serialization-stable reasoning traces without relying on a group-level constant reward. Algorithm 1 shows the detailed GRPO algorithm for trace-isomorphism consistency training.

C Implementation Details

We attach TRA to the middle 21 layers ($\ell \in [8, 28]$) of the 32-layer Llama-3.1-8B-Instruct backbone, leaving the eight bottom layers and the three top layers unmodified, based on the rationale that lower layers tend to specialize in local lexical and syntactic processing while the top layers serve as a vocabulary-space projection that sharpens the next-token distribution [39, 5], with structural composition concentrated in the intermediate layers [23]. The exact local-hop factorization in Appendix B.2 establishes that the structural bias in Eq. (2) can be represented through augmented queries and keys. For practical efficiency, we instead employ a learned rank-r landmark representation with $r = 8$ by default. Specifically, each node v is assigned an r-dimensional fingerprint $\mathbf{f}_v$ containing normalized shortest-path distances to r landmarks, selected as the top-r degree-ranked nodes of the input graph. These distances are truncated at a fixed threshold before normalization. Let $\mathbf{F} \in \mathbb{R}^{n \times r}$ collect the node fingerprints. The structural factors are constructed as $\mathbf{L}^{\ell,h} = \mathbf{F}\mathbf{W}_Q^{\ell,h}$ and $\mathbf{R}^{\ell,g} = \mathbf{F}\mathbf{W}_K^{\ell,g}$, where $\mathbf{W}_Q^{\ell,h}, \mathbf{W}_K^{\ell,g} \in \mathbb{R}^{r \times r}$ are learnable projections, and g denotes the GQA key/value group containing head h. Their bilinear interaction $\mathbf{f}_u^\top \mathbf{W}_Q^{\ell,h} (\mathbf{W}_K^{\ell,g})^\top \mathbf{f}_v$ defines the learned structural bias between nodes u and v. This compact representation does not guarantee exact recovery

of the local-hop bias. The gate $\alpha^{\ell,g}$ is parameterized as a softplus of a learnable scalar initialized to a small positive value. Graph parsing, fingerprint computation, and node-token alignment are performed once per training instance and reused across the attended layers.

D Experimental Details

D.1 Dataset Details

Table 8 provides a summary of the GraphInstruct benchmark [8] used throughout our experiments. The benchmark comprises nine graph computation tasks spanning three complexity classes, with graphs generated using the Erdős–Rényi random graph model. Node counts range from 2 to 100 and edge counts from 5 to 500, producing a diverse distribution of graph densities. Each task is paired with a natural-language prompt template that describes the graph in edge-list format and poses a specific computational query. The ground-truth answer includes both a chain-of-thought reasoning trace and a final answer delimited by `###`. We follow the official train/test split provided by Chen et al. [8].

Table 8: Summary statistics of the GraphInstruct benchmark. **Dir.**: whether the graph is directed. **Wt.**: whether edges carry weights. **Answer type**: the format of the ground-truth answer. $|\mathcal{V}|$ and $|\mathcal{E}|$ denote the node and edge count ranges across the dataset. Train and test sizes are the number of instances per task.

Task	Key	Complexity	Dir.	Wt.	Answer type	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	Train	Test
Cycle Detection	`cycle`	Linear	✗	✗	Yes/No	2–100	5–500	2,717	400
Connectivity	`connectivity`	Linear	✗	✗	Yes/No	2–100	5–500	2,687	400
Bipartite Check.	`bipartite`	Linear	✓	✗	Yes/No	2–100	5–500	2,013	400
Topological Sort	`topology`	Linear	✓	✗	Node list	2–100	5–500	872	400
Shortest Path	`shortest`	Poly.	✗	✓	Scalar	2–100	5–500	1,392	400
Max Triangle Sum	`triplet`	Poly.	✗	✗	Scalar	2–100	5–500	2,756	400
Maximum Flow	`flow`	Poly.	✓	✓	Scalar	2–100	5–500	405	400
Hamiltonian Path	`hamilton`	NP-c.	✗	✗	Yes/No + path	2–100	5–500	2,097	400
Subgraph Matching	`substructure`	NP-c.	✓	✗	Yes/No	2–100	5–500	3,186	400
Total								18,125	3,600

Task descriptions. We briefly describe each task for completeness. Full prompt templates are provided in [8].

- **Cycle Detection.** Given an undirected graph, determine whether it contains a cycle (Yes/No).
- **Connectivity.** Given an undirected graph and a pair of nodes, determine whether a path exists between them (Yes/No).
- **Bipartite Checking.** Given a directed graph, determine whether it is bipartite (Yes/No).
- **Topological Sort.** Given a directed acyclic graph, produce a valid topological ordering of its nodes.
- **Shortest Path.** Given a weighted undirected graph and a source–target pair, compute the weight of the shortest path.
- **Maximum Triangle Sum.** Given an undirected graph with node weights, find the maximum sum of weights among all triangles (cliques of size three).
- **Maximum Flow.** Given a weighted directed graph and a source–sink pair, compute the maximum flow.
- **Hamiltonian Path.** Given an undirected graph, determine whether a Hamiltonian path exists and, if so, output one.
- **Subgraph Matching.** Given a directed graph G and a smaller directed graph G', determine whether G' is isomorphic to a subgraph of G.

D.2 Representative Examples

Below, we present representative examples from the GraphInstruct dataset. Each box shows the **Query** (the full prompt presented to the model) and the **Answer** (the ground-truth reasoning trace

followed by the final answer after the ### delimiter). The header of each box lists the task name, dataset key, and complexity class.

Connectivity `connectivity` (Linear)

Query:
Determine whether two nodes are connected in an undirected graph. In an undirected graph, (i,j) means that node i and node j are connected with an undirected edge. Given a graph and a pair of nodes, you need to output Yes or No, indicating whether the node i and node j are connected. Q: The nodes are numbered from 0 to 5, and the edges are: (0,1) (1,2) (3,4) (4,5). Is there a path between node 1 and node 4?

Answer:
Node 1 is in the connected block consisted of node 0, node 1, and node 2. Node 4 is in the connected block consisting of node 3, node 4, and node 5. Node 1 and node 4 are not in the same connected block, so the answer is no. ### No.

Cycle Detection `cycle` (Linear)

Query:
Determine whether or not there is a cycle in an undirected graph. In an undirected graph, (i,j) means that node i and node j are connected with an undirected edge. Given a graph, you need to output Yes or No, indicating whether there is a cycle in the graph. Q: The nodes are numbered from 0 to 5, and the edges are: (3,5) (1,0) (3,0) (3,4) (4,1) (2,3). Is there a cycle in this graph?

Answer:
Starting from node 3, we can go to nodes 5, 0, 4, and 2. From node 0, we can reach nodes 3 and 1. From node 4, we can reach nodes 3 and 1. From node 1, we can reach nodes 0 and 4. By looking at these connections, we can see there is indeed a cycle. For example, starting from node 3, we can go to node 0 (via edge 3-0), then to node 1 (via edge 1-0), then to node 4 (via edge 4-1), and back to node 3 (via edge 3-4). This forms a cycle [3-0-1-4-3] without revisiting any edge. Thus, there is a cycle in this graph. ### Yes.

Shortest Path `shortest` (Polynomial)

Query:
Find the shortest path between two nodes in an undirected graph. In an undirected graph, (i,j,k) means that node i and node j are connected with an undirected edge with weight k. Given a graph and a pair of nodes, you need to output the shortest path between the two nodes. Q: The nodes are numbered from 0 to 4, and the edges are: (0,4,1) (0,1,3) (1,2,6) (3,4,3). Give the weight of the shortest path from node 3 to node 4.

Answer:
When looking at the graph with nodes 0 to 4 and the specified edges, our focus is on finding the shortest path from node 3 to node 4. Here, we have a direct edge (3,4) with a weight of 3. It's important to note that there's no alternative route involving other nodes that could offer a shorter path, as the connections don't facilitate another pathway between these two nodes. Therefore, the direct route between nodes 3 and 4, with its weight of 3, stands as the shortest path. ### 3.

Maximum Triangle Sum `triplet` (Polynomial)

Query:
Find the maximum sum of the weights of three interconnected nodes. In an undirected graph, [i, k] means that node i has the weight k. (i,j) means that node i and node j are connected with an undirected edge. Given a graph, you need to output the maximum sum of the weights of three interconnected nodes. Q: The nodes are numbered from 0 to 4, weights of nodes are: [0, 2] [1, 9] [2, 6] [3, 10] [4, 4], and the edges are: (0, 1) (0, 3) (1, 3) (2, 4) (3, 4). What is the maximum sum of the weights of three interconnected nodes?

Answer:
The nodes and their weights are as follows: Node 0 with weight 2, Node 1 with weight 9, Node 2 with weight 6, Node 3 with weight 10, and Node 4 with weight 4. Upon examining the

connections between these nodes, it becomes evident that only Nodes 0, 1, and 3 form a fully interconnected set, with each node directly connected to the other two. The sum of their weights is 2 (Node 0) + 9 (Node 1) + 10 (Node 3) = 21. Therefore, the maximum sum of the weights of three interconnected nodes in this graph is 21. ### 21.

Hamiltonian Path `hamilton` (NP-complete)

Query:
Determine whether or not there is a Hamiltonian path in an undirected graph. In an undirected graph, (i,j) means that node i and node j are connected with an undirected edge. Given a graph, you need to output Yes or No, indicating whether there is a Hamiltonian path in the graph. Q: The nodes are numbered from 0 to 5, and the edges are: (0,3) (0,2) (0,1) (0,5) (1,4) (1,3) (1,2) (3,5) (4,5). Is there a Hamiltonian path in this graph?

Answer:
To determine if a Hamiltonian path exists in an undirected graph, we need to check if there's a path that visits each node exactly once. Starting at Node 0, we can go to Node 1 (which connects to Nodes 2, 3, 4). From Node 1, moving to Node 4 seems a strategic choice because Node 4 only connects back to Node 1 and to Node 5. After reaching Node 4, we must go to Node 5. From Node 5, we can go to Node 3, as Node 3 connects to Nodes 0 and 1 (which we've visited) and to Node 5. Finally, from Node 3, we can go to Node 2. So, one possible Hamiltonian path is: [0,1,4,5,3,2]. Therefore, there is a Hamiltonian path in this graph. ### Yes, [0,1,4,5,3,2].

Answer equivariance classification. Following the formulation in Section 2, we note that the nine tasks fall into two categories with respect to answer equivariance. Tasks with scalar or Yes/No outputs (Cycle Detection, Connectivity, Bipartite Checking, Shortest Path, Maximum Triangle Sum, Maximum Flow, Subgraph Matching) produce *invariant* answers: the ground-truth answer does not change under node relabeling. Tasks with node-list outputs (Topological Sort, Hamiltonian Path) produce *equivariant* answers: the output is a valid permutation of the answer under the same relabeling applied to the input. This distinction affects how the ground-truth trace and answer are rewritten during augmentation (Section 3.2).

D.3 Baseline Configurations

We use four classes of baselines.

- **Prompted LLMs.** Closed-source LLM results are evaluated using the benchmark prompt format. Zero-shot prompting provides only the task instruction and graph query, while two-shot CoT prompting prepends two training examples from the same task.
- **Open-source instruction LLMs.** Llama-3.1-8B-Instruct and Qwen3-8B are evaluated without task-specific fine-tuning. Decoding is greedy for exact-match evaluation unless otherwise stated.
- **Graph neural networks.** GCN, GIN, and GAT consume graph structure directly. They are included on tasks that can be formulated as graph-level classification or scalar prediction. They are not evaluated on tasks requiring free-form node-list generation.
- **Post-trained LLMs.** Naive SFT fine-tunes the backbone on the original GraphInstruct training set. GraphWiz-RFT and GraphWiz-DPO follow the training objectives introduced by Chen et al. [8]. G1 is included as a recent RL-based graph-reasoning baseline. We use Llama-3.1-8B-Instruct as the default LLM backbone for these baselines and our model.

D.4 Hardware and Packages

All experiments are conducted on a workstation with 3 NVIDIA RTX PRO 6000 Blackwell GPUs ($\approx$ GDDR7 memory each). All post-training stages are implemented with the HuggingFace Transformers stack [47]. We use HuggingFace Accelerate [14] to manage distributed launching, device placement, gradient accumulation, and mixed-precision execution across the post-training pipeline. DeepSpeed [34] is used as the distributed training backend for memory-efficient optimization, including ZeRO-style optimizer-state and gradient partitioning [32]. This setup is used for both trace-supervised fine-tuning and the RL stages, so that the training code can share the same model-loading, checkpointing, and distributed execution path across stages.

Table 9: Efficiency analysis of TRA

Attention variant	Rank	Tok/s	Mem. (GB)	Avg. acc.
Standard Flash-Attention (multi-aug SFT)	–	38.2	63.1	60.72
Dense topology bias (multi-aug SFT)	–	OOM	>95.0	–
Low-rank TRA (multi-aug SFT)	4	36.9	66.4	63.05
Low-rank TRA (multi-aug SFT)	8	35.8	67.9	63.31
Low-rank TRA (multi-aug SFT)	16	33.5	70.4	63.55

For RL post-training, we initialize the policy from the trace-SFT checkpoint and keep the SFT model as the reference policy. Rollouts are generated by the current policy, scored by the task reward, trace-isomorphism consistency reward, and, in the full model, the process reward model. The resulting rewards are used to form the GRPO objective described in Appendix B.3. We use the same distributed infrastructure for rollout generation and policy updates, with DeepSpeed handling the memory-intensive optimizer states and Accelerate coordinating process-level execution.

D.5 Hyperparameter Settings

Unless otherwise stated, all experiments use Llama-3.1-8B-Instruct as the backbone model. We set the maximum input length to 1024 tokens and the maximum generation length to 1024 tokens. Models are optimized with AdamW. For supervised fine-tuning, we train for 3 epochs with a learning rate of 2×10^{-5}. Each training graph is augmented with $k = 4$ permutation-equivalent serializations, and the representation-consistency coefficient is set to $\lambda_{\text{repr}} = 0.5$. For Topology-aware Residual Attention, we use rank $r = 8$.

For reinforcement learning, we use GRPO with $m = 4$ graph serializations and $n = 4$ rollouts per serialization. The policy is sampled with temperature 0.7, the learning rate is set to 5×10^{-7}, and the clipping parameter is $\epsilon = 0.2$. The trace-consistency reward coefficient is $\lambda_c = 0.5$, the process-reward coefficient is $\lambda_p = 1$, and the KL coefficient is initialized to 0.01. These hyperparameters are kept fixed across tasks to evaluate the robustness of a single training recipe rather than task-specific tuning.

E More Experiments

E.1 Efficiency of TRA

We evaluate the computational overhead of TRA relative to standard Flash-Attention and to a dense topology-bias implementation. To isolate the architectural effect of the attention modification from the data-level effect of multi-augmentation, all variants in this analysis are trained under the same multi-augmentation SFT recipe used for TRACER-SFT, with $k = 4$ permutation-equivalent serializations per graph and the representation-consistency loss active. The variants differ only in their structural attention mechanisms. For low-rank TRA, structural preprocessing constructs landmark-distance fingerprints from the input graph, which are computed once per graph and reused across attended layers. Unlike a dense topology bias, which explicitly materializes pairwise structural scores in the attention logits, TRA absorbs the topological bias into low-rank query/key augmentations. Specifically, each attended layer appends r structural dimensions to Q and K while leaving V unchanged. This increases the attention kernel arithmetic by approximately $(d_k + r)/d_k$, but preserves compatibility with Flash-Attention and avoids materializing an additional dense attention-bias matrix.

Table 9 reports throughput, peak memory, and average accuracy. The dense topology-bias variant exceeds the memory budget, requiring more than 96 GB of GPU memory and confirming that a naive realization of the structural bias is impractical at this scale. Low-rank TRA remains memory efficient: with rank $r = 8$, it reduces throughput by only about 6% relative to standard Flash-Attention while raising average accuracy from 60.72% to 63.31%. Because the multi-augmentation training data and loss are held fixed across rows, this 2.59-point gain reflects the contribution of the attention modification itself rather than of the broader TRACER-SFT recipe. Increasing the rank from 8 to 16 yields a smaller additional gain, from 63.31% to 63.55%, while further decreasing throughput and increasing memory usage. We therefore adopt $r = 8$ as the default setting, which provides the best trade-off between accuracy and efficiency.

Table 10: Fine-grained ablation of TRACER components. *Naive SFT* is single-serialization training with $\mathcal{L}_{\text{task}}$. *Multi-aug SFT* replaces this with $k = 4$ permutation-equivalent serializations and the representation-consistency loss $\mathcal{L}_{\text{repr}}$. *Trace-iso RL* adds the trace-isomorphism reward, and *PRM* adds the process reward model. All variants share Llama-3.1-8B as the backbone.

	Easy (Linear)				Medium (Poly.)			Hard (NP-c.)		
Variant	Cycle	Connect.	Bipartite	Topo.	Shortest	Triangle	Flow	Hamilton	Subgraph	**Avg.**
Naive SFT (no multi-aug, no TRA)	75.50	83.50	76.75	53.75	24.50	45.50	32.75	23.75	75.00	54.56
+ TRA only (no multi-aug)	79.25	85.25	81.50	55.00	24.50	47.50	35.50	25.00	76.50	56.67
+ multi-aug SFT only (no TRA)	84.00	88.00	94.00	62.00	23.00	50.50	38.50	29.00	77.50	60.72
TRACER-SFT (TRA + multi-aug SFT)	90.00	89.50	97.25	63.75	23.50	53.00	43.75	31.25	77.75	63.31
+ trace-iso RL	93.75	91.00	96.50	62.75	33.75	51.75	45.75	33.00	81.00	65.50
TRACER (TRA + multi-aug + RL + PRM)	93.75	91.50	95.50	64.25	40.50	59.50	46.25	33.50	81.00	67.31

The remainder of the 8.75-point gap between TRACER-SFT (63.31%) and the naive Llama-3.1-8B SFT baseline (54.56% in Table 2) is therefore attributable to the multi-augmentation SFT recipe rather than to the attention modification, with the two effects acting complementarily.

E.2 Fine-grained Ablation Study

The ablation reported in Table 5 treats TRACER-SFT as a single unit, leaving the data-level effect of multi-augmentation entangled with the architectural effect of TRA. To make the decomposition explicit, we run a fine-grained ablation in which the two factors are toggled independently on top of the naive Llama-3.1-8B SFT baseline. The first new variant adds TRA to the attention kernel while keeping the original single-serialization $\mathcal{L}_{\text{task}}$ objective, isolating the architectural contribution of the topology-aware residual. The second new variant retains standard Flash-Attention but adopts the multi-augmentation SFT recipe with $k = 4$ permutation-equivalent serializations and the representation-consistency loss $\mathcal{L}_{\text{repr}}$, isolating the data-level contribution of the SFT recipe. All variants share the Llama-3.1-8B backbone, optimizer, and training budget, so the only manipulated factors across rows are the attention kernel and the SFT objective.

Table 10 reports the resulting decomposition. Adding TRA alone to naive SFT improves the average accuracy from 54.56% to 56.67%, a 2.11-point gain that reflects the standalone effect of injecting topology into self-attention. Adding multi-augmentation SFT alone improves the average to 60.72%, a 6.16-point gain that reflects the data-level effect of training under permutation-equivalent serializations together with $\mathcal{L}_{\text{repr}}$. Combining the two yields TRACER-SFT at 63.31%, an 8.75-point gain over naive SFT. The combined gain slightly exceeds the sum of the two standalone effects ($2.11 + 6.16 = 8.27$), indicating that the two interventions act largely complementarily, with a small superadditive component that is consistent with the intuition that an explicit topological prior is most useful when the model also sees varied serializations of the same graph.

Per-task patterns. The per-task numbers refine this picture. Standalone TRA produces its largest gains on tasks whose answers depend directly on graph structure, including Bipartite Checking ($+4.75$), Cycle Detection ($+3.75$), and Maximum Flow ($+2.75$), and is essentially neutral on Shortest Path, where the model has too few weighted-edge examples in the single-serialization regime to exploit the structural bias. Standalone multi-augmentation, by contrast, contributes most on tasks whose ground-truth answer or trace is sensitive to node relabeling, including Bipartite Checking ($+17.25$), Cycle Detection ($+8.50$), and Topological Sort ($+8.25$). Tasks that benefit from both factors, such as Hamiltonian Path ($+1.25$ and $+5.25$ standalone, $+7.50$ combined) and Maximum Flow ($+2.75$ and $+5.75$ standalone, $+11.00$ combined), see the largest combined improvements. Shortest Path is the one task on which both interventions slightly hurt the SFT-stage performance, an effect that the RL stage subsequently overturns.

RL contributions on top of TRACER-SFT. The RL stage adds an additional 4.00 points to the average, decomposed into 2.16 points from the trace-isomorphism reward and 1.84 points from the process reward model. The largest absolute gains from RL fall on Shortest Path ($+17.00$, from 23.50% to 40.50%) and Maximum Triangle Sum ($+6.50$), tasks for which structural reasoning over edge weights and node weights most benefits from process supervision. Combined with the SFT-stage decomposition above, this indicates that the architectural, data-level, and reasoning-level interventions in TRACER address distinct failure modes and contribute roughly additively without redundancy.

E.3 Failure Case Analysis

We manually inspect 100 incorrect predictions sampled uniformly from TRACER's test-set errors on GraphInstruct and classify them by failure mode.

Table 11: Distribution of failure modes among 100 manually inspected incorrect predictions of TRACER on the GraphInstruct test set.

Failure mode	Count	Most affected tasks
Trace truncation at the generation length cap	38	Hamilton, Topo., Subgraph
Incomplete coverage of the search space	24	Hamilton, Subgraph
Arithmetic slip in weighted summation	19	Shortest, Flow, Triangle
Misidentification of disconnected components	11	Connect., Bipartite
Format or extraction error after correct trace	8	Topo.

The dominant failure mode is reasoning truncation on large graphs, in which the model produces a partial trace that exceeds the maximum generation length before reaching the answer span. This affects 38% of inspected failures and concentrates on graphs with $|V| > 50$, where the algorithmic trace itself is long. The second most common mode is incomplete search-space coverage on NP-complete tasks: on Hamiltonian Path, the model often verifies a candidate path correctly but fails to exhaustively rule out alternatives when no Hamiltonian path exists, leading to a false positive. The third mode is arithmetic slips on weighted tasks, where the model identifies the correct shortest path's edges but then sums the weights incorrectly. The remaining cases are divided between component-identification errors on undirected linear-time tasks and formatting failures in which the trace is correct but the final answer is not extracted into the designated span.

These failure modes suggest two complementary directions for future work: longer-context generation or explicit trace-continuation strategies to handle the 38% of cases lost to length limits, and tighter coupling between TRA's structural representation and the arithmetic operations performed during decoding, which currently rely on the backbone's general numerical competence rather than on any graph-aware mechanism.

F Broader Impacts

This work contributes to the development of language models that can reason more reliably over graph-structured inputs. Graph computation is a core abstraction in many scientific and engineering domains, including networks, molecules, programs, knowledge bases, and logistics. By improving how language models represent graph topology, maintain reasoning traces, and remain consistent across equivalent serializations, this work can support more accessible tools for graph analysis and algorithmic problem solving. The proposed framework can also help researchers better understand the relationship between textual representations, structural invariance, and multi-step reasoning in large language models, providing a useful direction for future studies on structured computation.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction include the claims made in the paper.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Appendix E.1 explicitly identifies a computational and memory trade-off requiring a low-rank approximation for the structural attention matrix.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The paper includes experiments in Section 4, and the information to reproduce the results in Appendix D.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: See the code link in the abstract.

Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: See Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

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

   Answer: [No]

   Justification: Error bar does not apply to our work.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: See Appendix D.4 and Appendix E.1.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research conforms the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: See Appendix F.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: The paper has cited the original paper that produced the code package or dataset, i.e., GraphWiz [8].

    Guidelines:

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

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

16. **Declaration of LLM usage**

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

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

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