TEACHING LLMS TO SEE GRAPHS: UNIFYING TEXT AND STRUCTURAL REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Applying Large Language Models (LLMs) to graph-structured data usually involves multi-step pipelines in which textual node attributes are compressed into single tokens and further processed by GNNs, discarding most of their semantic content. We introduce the Graph Transformer Language Model (GTLM), which enables a pretrained LLM to process graph topology natively and removes this bottleneck entirely. GTLM injects graph-aware attention biases directly into the LLM's attention modules, adding only 0.015% structure-related parameters relative to the base model. Training updates only the structural parameters together with a LoRA adapter on the base model. We prove that our bidirectional attention prefix is permutation-equivariant over nodes and that GTLM reduces exactly to the pretrained model when no graph is present. Having no global node ordering, GTLM shows no positional degradation and does not *get lost in the middle*: needle-in-a-graph accuracy stays flat from 1k to 64k tokens and 4× past the training length, while an identically trained flat-text baseline collapses. Comprehensive evaluations show that a GTLM matches or exceeds domain-specific state-of-the-art models on text-attributed graph benchmarks, GraphRAG on WebQSP, and molecular benchmarks, while meaningfully improving over strong baselines on GraphQA. We further show that GTLM's attention heads implicitly learn to simulate message passing, explaining its strength on algorithmic tasks. Together, these results suggest that a minimally adapted pretrained LLM can serve as a general backbone for graph learning.

1 INTRODUCTION

Many real-world problems involve interconnected structures which Large Language Models (LLMs) have to work around due to their inherent inability to perceive structure. The standard way to give such data to an LLM is to serialize the graph into text (Fatemi et al., 2024). Serialization introduces an arbitrary node order, so the output depends on a choice that carries no meaning.

However, from an architectural point of view, a decoder-only LLM is already a Graph Transformer (GT) (Ying et al., 2021): its input is a path graph over tokens, and the positional encodings under the causal mask are the structure that its attention sees. Moving from a path to any graph should mean changing the way attention sees structure, not adding a separate graph encoder to the model.

Most existing approaches do the latter. Graph-Tokenizing LLMs (GTokenLLMs) (Zhang et al., 2026) typically rely on disjointed, multi-step pipelines. These frameworks condense each text-attributed node into a single token via a preliminary encoder, optionally process these representations with Graph Neural Networks (GNNs), and pass the representations into rigid serialization templates. While effective on some benchmarks, this paradigm introduces significant bottlenecks: the fixed input formats constrain versatility, and the lossy compression of rich textual data into solitary tokens prevents the model from accessing fine-grained, node-level information.

In this paper, we introduce the Graph Transformer Language Model (GTLM), a novel architecture that addresses the limitations of multi-step pipelines by integrating structural information directly into the LLM's attention mechanism without lossy text compression (Figure 1). Inspired by the Relative Positional Encodings (RPEs) in Graphormer (Ying et al., 2021), GTLM leverages the global topology to broadcast graph attention biases to text token pairs. To preserve permutation equivariance, we flatten the node texts, reset their position IDs, and apply bidirectional prefix attention.

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

In order to demonstrate the versatility of GTLM, we evaluate it on a wide variety of benchmarks related to text and graphs, all previously attempted by different highly specialized approaches. We empirically show that GTLM is capable of bridging the gap between LLMs and structured data and that the generality of LLMs carries over to these graph-related tasks.

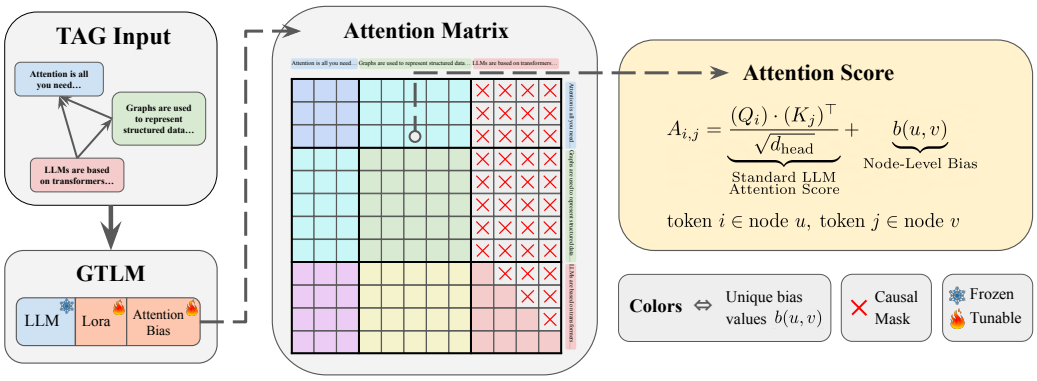


Figure 1: **GTLM architecture.** Node texts are concatenated, and node-level topological biases $b(u, v)$ are broadcasted to constituent token pairs (where $i \in u, j \in v$) within the attention matrix. Graph structure is integrated using tunable attention biases and LoRA on a frozen base LLM.

In summary, our main contributions are:

- **Architectural Innovation & Theoretical Grounding:** We introduce **GTLM**, a unified architecture that builds upon LLMs and enables them to perceive graphs. Furthermore, we prove that GTLM is a natural generalization of LLMs, satisfying prefix node permutation equivariance and maintaining exact backward compatibility with standard self-attention.
- **Empirical Performance:** Through extensive evaluation on a wide variety of standardized benchmarks, we demonstrate that GTLM consistently achieves or exceeds state-of-the-art performance against highly specialized niche approaches.
- **Superior Long-Context Retrieval over LLMs:** On a synthetic needle-in-a-graph task, we demonstrate that GTLM does not get *lost in the middle* and that its multi-step retrieval abilities do not degrade with difficulty and context length, while an LLM collapses.

2 Related Work

2.1 Large Language Models for Graphs

Initial approaches to graph-LLM integration relied on explicit text serialization (e.g., Fatemi et al., 2024), which inherently suffers from poor graph-aware reasoning and rapid context exhaustion. To mitigate these constraints, research pioneered by Perozzi et al. (2024) shifted toward multimodal graph alignment, a paradigm further advanced by models such as GraphGPT (Tang et al., 2024), LLaGA (Chen et al., 2024), and RGLM (Zhang et al., 2026). These architectures project graph nodes into "soft tokens" within the LLM's embedding space to facilitate structural processing.

Despite their efficiency, these alignment-based methods introduce a significant **compressive bottleneck**: rich, multi-sentence node attributes are condensed into solitary dense vectors, abstracting away fine-grained semantics and preventing the LLM from executing token-level attention between nodes. Unlike these multi-step projection frameworks, **GTLM** eliminates the compression bottleneck entirely by natively routing topological awareness through the attention mechanism while preserving the full text token sequence of every node.

2.2 Structural Encodings in Graph Transformers

Since Transformers were adapted to graphs (Vaswani et al., 2017; Yuan et al., 2025), structural injection typically falls into two categories: **Absolute** Positional Embeddings (APEs), which modify

initial node features using structural priors, and **Relative** Positional Encodings (RPEs), which infer topology via additive biases applied to raw attention scores. Our architecture relies on RPEs, building upon three foundational methods.

First, **Shortest Path Distances (SPD)** index a learnable scalar bias based on the discrete shortest path distance between any node pair, natively modulating token interaction strength based on structural proximity without message passing (Ying et al., 2021). Second, **Relative Random Walk Probabilities (RRWP)** capture complex structural dynamics using K-step walk probabilities $\mathbf{P}_{u,v} \in \mathbb{R}^K$ derived from the graph's transition matrix (Ma et al., 2023).

Finally, to capture directed graph topology, the **Magnetic Laplacian** utilizes a matrix that encodes edge directionality into a phase matrix $\mathbf{\Theta}^{(q)} \in \mathbb{R}^{N\times N}$ to obtain the Normalized Magnetic Laplacian $\mathbf{L}_N^{(q)} \in \mathbb{C}^{N\times N}$ (Zhang et al., 2021). By performing spectral decomposition on $\mathbf{L}_N^{(q)}$, basis-invariant edge-level features $\mathbf{K} \in \mathbb{C}^{N\times N\times d_{\text{Mag}}}$ can be constructed, similar to Huang et al. (2025).

3 METHODOLOGY

To unify the graph and text modalities, we represent all data through an underlying graph structure where nodes (and optionally edges) consist of text sequences. The entire input graph is processed by a modified LLM, inspired by GTs, which we call **Graph Transformer Language Model (GTLM)**. It accounts for both the global topology and the relative positions of text tokens within each node.

3.1 PROBLEM SETUP

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ be a Text-Attributed Graph (TAG), with vertices $\mathcal{V} = \{v_0, v_1, \ldots, v_N\}$, edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, and tokenized textual attributes $\mathcal{T} = \{t_0, t_1, \ldots, t_N\}$. Let $v_{\text{target}} = v_0$ denote the **target node** for which we want to perform autoregressive text generation, with its corresponding text denoted as $y = t_0$. Further, let L be the total sequence length equal to $\sum_{i=0}^{N} \text{len}(t_i)$.

Assuming all context nodes $\{v_1, \ldots, v_N\}$ are permutation invariant, we define an arbitrary permutation $\pi \in S_N$. Finally, let S be a set of **structural features** derived from $(\mathcal{V}, \mathcal{E})$, which GTLM utilizes for attention bias computation. Our objective is to maximize the log-likelihood of the target sequence y given the context nodes and the structural features. The autoregressive generation of token y_j, conditioned on the target prefix $y_{<j}$ and the permuted context tokens, is formulated as: $\max_\theta \quad \sum_{j=1}^{|y|} \log P_\theta\Big(y_j \mid \big[t_{\pi(1)} \oplus \cdots \oplus t_{\pi(N)} \oplus y_{<j}\big], S\Big)$, where P_θ denotes the probability distribution parameterized by the GTLM network θ, and $\oplus$ denotes sequence concatenation.

3.2 MODIFIED ATTENTION

The main innovation of our approach is the combination of sequential RPEs traditionally used by LLMs, such as RoPE (Su et al., 2023), and RPEs for graphs. To ensure **permutation equivariance**, the position IDs for RoPE are counted from zero onwards inside of each node individually. While in this paper we primarily focus on RoPE, our architecture is compatible with other PEs such as ALiBi (Press et al., 2022) or dual RoPE (Gemma Team, 2025), which we empirically demonstrate in Appendix D.

The raw attention score $A_{i,j}^{(l,h)}$ between tokens i and j (inside of nodes u and v, respectively) for the h-th attention head within the l-th layer of GTLM is calculated as shown below:

$$A_{i,j}^{(l,h)} = \underbrace{\frac{(Q_i^{(l,h)}) \cdot (K_j^{(l,h)})^\top}{\sqrt{d_{\text{head}}}}}_{\text{Standard LLM Attention Score}} + \underbrace{b_{\text{SPD}}^{(l,h)}(u,v)}_{\text{SPD Bias}} + \underbrace{b_{\text{RRWP}}^{(l,h)}(u,v)}_{\text{RRWP Bias}} + \underbrace{b_{\text{Mag}}^{(l,h)}(u,v)}_{\text{Magnetic Laplacian Bias}} .$$

Critically, all nodes before the target node are treated as a prefix, following the PrefixLM architecture (Raffel et al., 2020), where a fully-visible (non-causal) mask is applied to the prefix to allow for bidirectional context, while maintaining causal constraints for the target node's text generation. In our architecture, we propose three main ways of encoding the structural relative position of tokens

and we can utilize any combination of the following approaches simultaneously, each relying on mathematical features derived in Appendix A.

Shortest Path Distance (SPD) Bias. This is a learned lookup table mapping the SPD of the corresponding nodes to an attention bias added to the raw attention scores.

$$b_{\text{SPD}}^{(l,h)}(u,v) = \text{b}_{\text{spd}}^{(l,h)}(D_{\text{hops}}(u,v)), \text{ where } b_{\text{spd}} \in \mathbb{R}^{L\times H\times \text{max_spd}} \text{ is learned, with } b_{\text{SPD}}^{(l,h)}(u,u) = 0.$$

Relative Random Walk Probabilities (RRWP) Bias. Building upon the structural formulation introduced earlier, let $\mathbf{P}_{u,v} \in \mathbb{R}^K$ represent the K-step random walk probabilities between nodes u and v. We transform these initial node-pair-level features into a scalar attention bias using a layer-specific, learned Multi-Layer Perceptron (MLP_{RRWP}). Crucially, to preserve the intra-node attention properties essential for compatibility with pretrained LLMs, this bias is explicitly zeroed out for tokens belonging to the same node:

$$b_{\text{RRWP}}^{(l,h)}(u,v) = \begin{cases} 0 & u = v \\ [\text{MLP}_{\text{RRWP}}^{(l)}(\mathbf{P}_{u,v})]_h & u \neq v \end{cases},$$

where $\text{MLP}_{\text{RRWP}}^{(l)} : \mathbb{R}^K \to \mathbb{R}^{\text{n_heads}}$ maps the structural features to the respective attention heads of the l-th layer.

Magnetic Laplacian Bias. To natively incorporate directed spectral geometry into our model, we derive relative attention biases from the basis-invariant Magnetic Laplacian kernel $\mathbf{K} = \mathbf{V}\text{diag}(\phi(\lambda))\mathbf{V}^\dagger \in \mathbb{C}^{N\times N\times d_{\text{Mag}}}$, as introduced in Section 2. To instantiate the required permutation-invariant transformation for the eigenvalues, we specifically employ a Deep Set architecture (Zaheer et al., 2017):

$$\phi(\lambda_i) = \sigma(\mathbf{W}_2[\text{Linear}(\lambda_i) \oplus \text{mean}(\text{Linear}(\lambda))]) \in \mathbb{R}^{d_{\text{Mag}}},$$

where d_{Mag} is a hyperparameter dictating the spectral feature dimension. By decomposing the resulting node-pair interactions into their real and imaginary components, $\text{Re}(\mathbf{K})$ and $\text{Im}(\mathbf{K})$, we map the rich spectral information to a scalar bias for each attention head via a learned MLP.

Consistent with our approach of maintaining intra-node attention integrity for compatibility with pretrained LLMs, we explicitly zero out this bias when tokens belong to the same node:

$$b_{\text{Mag}}^{(l,h)}(u,v) = \begin{cases} 0 & u = v \\ [\text{MLP}_{\text{Mag}}^{(l)}([\text{Re}(\mathbf{K}_{u,v}) \oplus \text{Im}(\mathbf{K}_{u,v})])]_h & u \neq v \end{cases}.$$

3.3 Theoretical Properties of GTLM

Our proposed GTLM architecture satisfies two fundamental properties that successfully bridge the theoretical gap between Graph Transformers and Large Language Models. We provide proofs for both theoretical property claims, as well as empirical results to verify them in Appendix B.

Property 1—Prefix Node Permutation Equivariance: The internal representations and final outputs of GTLM are equivariant with respect to the serialization order of the prefix nodes.

Explanation: When flattening the nodes of a graph into a sequence, their order does not affect the attention scores between any two tokens, because the attention score depends only on the position ID (which is independent of the node order, due to the resetting of the RoPE ID) and the relative bias at the node level, which is by definition invariant under permutations.

Property 2—Pretraining Backward Compatibility: For any graph $\mathcal{G}$ consisting of a single node (representing a standard text sequence), the GTLM forward pass is mathematically identical to that of the unmodified base LLM.

Explanation: Since the condition that the bias is zero on the diagonal of the node-level bias matrix is imposed by definition, the bias is a null operation in a single-node graph, as is resetting the RoPE IDs. Finally, the full input is causally masked, since the only node is the target node—making the architecture mathematically equivalent to the base LLM.

4 EXPERIMENTS

We conduct comprehensive experiments across a diverse set of established benchmarks. Furthermore, we conduct a thorough ablation study (Section 4.6) evaluating the importance of each attention bias type. Unless stated otherwise, `Llama-3.2-1B` (Meta AI, 2024) served as the base model, and a Low-Rank Adapter (LoRA) (Hu et al., 2021) was used. Specific details about the training setup and hyperparameters for each experiment can be found in Appendices C–I. Each result that reports standard deviations was measured using three random seeds; otherwise only one was used. Our base-model scaling analysis (Appendix J) finds that performance is mostly saturated at 1B for classification tasks, while open-ended generation (such as WebQSP, CWQ, and ChEBI-20) benefits from scaling to an 8B-parameter base model.

4.1 GRAPHQA

GraphQA is a question-answering benchmark about abstract graphs and their properties (node degree, reachability, etc.). This problem was previously attempted by pure LLMs (Fatemi et al., 2024), and later by the GraphToken architecture (Perozzi et al., 2024). While GraphToken's performance meaningfully depended on the graph preprocessing method, our GTLM demonstrated exceptional performance and surpassed the baselines in all but one task, as seen in Table 1. The experiment was conducted under two settings: (1) where the input graphs were unchanged (*Standard*), and (2) where the input graphs were transformed into *Incidence* graphs—a bipartite transformation where nodes represent the original vertices and edges.

Table 1: GraphQA per-task accuracy (%) (**best**, <u>second-best</u>). The baselines are a Zero-Shot LLM and the top-3 highest scoring graph encoding methods used by GraphToken.

	Graph Tasks				Node Tasks		Edge Tasks		
Method	Node Count	Edge Count	Cycle Check	Triangle Count	Node Degree	Conn. Nodes	Reach-ability	Edge Exist.	Shortest Path
Zero-Shot LLM	21.7	12.4	76.0	1.5	14.0	14.7	84.9	44.5	11.5
3rd GraphToken	79.2 $_{\text{MPNN}}$	26.4 $_{\text{MHA}}$	96.2 $_{\text{MHA}}$	23.4 $_{\text{HGT}}$	26.6 $_{\text{HGT}}$	24.4 $_{\text{MHA}}$	93.2 $_{\text{MHA}}$	68.0 $_{\text{GCN}}$	60.4 $_{\text{GCN}}$
2nd GraphToken	91.2 $_{\text{MHA}}$	36.8 $_{\text{MPNN}}$	96.4 $_{\text{GCN}}$	26.6 $_{\text{MHA}}$	55.2 $_{\text{MHA}}$	25.0 $_{\text{MPNN}}$	94.2 $_{\text{NS}}$	71.8 $_{\text{HGT}}$	60.8 $_{\text{MHA}}$
1st GraphToken	<u>99.6</u> $_{\text{NS}}$	42.6 $_{\text{ES}}$	96.4 $_{\text{ES}}$	**34.8** $_{\text{MPNN}}$	<u>96.2</u> $_{\text{MPNN}}$	26.4 $_{\text{GCN}}$	94.4 $_{\text{HGT}}$	73.8 $_{\text{MHA}}$	63.8 $_{\text{MPNN}}$
GTLM (Incidence)	99.1$\pm$0.1	**62.2**$\pm$2.0	**98.7**$\pm$0.3	<u>31.6</u>$\pm$0.5	91.8$\pm$1.0	**98.8**$\pm$0.3	<u>98.7</u>$\pm$0.1	<u>99.6</u>$\pm$0.2	**91.7**$\pm$1.8
GTLM (Standard)	**100**$\pm$0.0	<u>56.9</u>$\pm$1.7	<u>96.7</u>$\pm$0.6	30.9$\pm$0.6	**99.7**$\pm$0.1	<u>90.7</u>$\pm$0.3	**99.0**$\pm$0.0	**99.8**$\pm$0.2	<u>90.1</u>$\pm$1.7

4.2 TEXT-ATTRIBUTED GRAPH BENCHMARKS

The second suite of tests we evaluate GTLM with focuses on node classification in Text-Attributed Graphs. We evaluate it on the following four benchmarks: *Cora*, *Pubmed* (Yang et al., 2016), and *OGBN-Arxiv* (Hu et al., 2020) for citation networks, and *Reddit* (Huang et al., 2024) for a social network. We use Accuracy (↑) and F1 score (↑) to evaluate GTLM, as those are the widely adopted standards for these benchmarks. To establish a robust evaluation framework, we benchmark GTLM against the suite of baselines curated by Zhang et al. (2026) with matching data splits. We categorize these baselines into two primary groups: *Graph-native architectures* and *Multimodal LLMs*.

As detailed in Table 2, GTLM consistently achieves state-of-the-art or highly competitive performance across all four benchmarks. The model surpasses all baselines on *Pubmed* and *OGBN-Arxiv* across both accuracy and F1 score. On *Cora*, GTLM establishes a new best F1 score while maintaining top-tier accuracy, and it remains a strong runner-up on the *Reddit* dataset.

4.3 GRAPHRAG EVALUATION

Knowledge-graph question answering is highly relevant to industrial applications. This area has seen advancements in highly specific methods and the current state of the art is established by agentic approaches, such as **GraphWalker** (Xu et al., 2026) and **TRACE** (Wang et al., 2026), which lead on both benchmarks by querying the graph interactively, so no single subgraph bounds

Table 2: Text-Attributed Graph node classification results (**best**, second-best).

Method		Cora		Pubmed		OGBN-Arxiv		Reddit	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1
Graph-native architectures									
GCN (Kipf & Welling, 2016)		89.30	87.95	89.48	89.04	73.83	54.97	63.19	62.49
GraphSAGE (Hamilton et al., 2017)		87.82	86.77	90.54	90.51	74.47	56.41	58.51	58.41
GAT (Veličković et al., 2018)		87.08	86.32	88.11	87.60	74.39	57.71	61.78	65.38
NodeFormer (Wu et al., 2022)		87.30	85.03	89.92	89.63	68.69	50.64	68.06	67.89
GRACE (Zhu et al., 2020)		89.11	88.12	86.89	86.58	72.00	50.79	65.39	65.39
Multimodal LLMs									
GraphText-ICL (Zhao et al., 2024)		82.29	81.16	56.72	52.55	68.67	51.67	56.50	52.23
GraphText-SFT (Zhao et al., 2024)		84.69	83.77	87.80	87.68	49.88	19.46	62.20	62.07
GraphGPT (Tang et al., 2024)		83.92	82.72	81.76	80.99	71.59	54.58	62.11	62.39
LLaGA (Chen et al., 2024)		88.75	87.87	90.34	90.25	74.61	56.48	66.38	66.38
RGLM-Decoder (Zhang et al., 2026)		89.85	89.39	91.15	91.07	75.00	56.92	**68.64**	**68.54**
RGLM-Similarizer (Zhang et al., 2026)		89.67	88.96	91.61	91.51	75.14	57.74	66.79	66.77
RGLM-Denoiser (Zhang et al., 2026)		**90.22**	89.35	90.95	90.90	75.10	56.84	67.64	67.50
GTLM (Llama-3.2-1B)	*Best*	90.04	**92.14**	**94.70**	**94.84**	**76.53**	**70.02**	68.09	68.00
	Mean	88.99	91.31	94.57	94.76	76.07	69.40	67.80	67.71
	Std	1.12	0.90	0.11	0.10	0.54	0.86	0.37	0.32

them. Therefore, we leave them out of the headline comparison, as our aim is to focus on graph comprehension, rather than optimizing the subgraph retrieval method. We compare against systems that, like ours, extract once under a fixed budget—**RoG** (Luo et al., 2024), **GNN-RAG** (Mavromatis & Karypis, 2025) with its retrieval-augmented variant **GNN-RAG+RA** and the variant reading the exact same SR subgraphs we do (Zhang et al., 2022), and **SubgraphRAG** (Li et al., 2025)—on **WebQSP** (Yih et al., 2016) and **CWQ** (Talmor & Berant, 2018).

Table 3: Knowledge graph QA benchmarks WebQSP and CWQ (**best**, second-best).

Method	Reader	WebQSP			CWQ		
		Hit	H@1	F1	Hit	H@1	F1
Own retrieval							
RoG	Llama-2-7B	85.7	80.0	70.8	62.6	57.8	56.2
GNN-RAG	Llama-2-7B	85.7	80.6	71.3	66.8	61.7	59.4
GNN-RAG + RA	Llama-2-7B	**90.7**	**82.8**	73.5	**68.7**	62.8	60.4
SubgraphRAG	GPT-4o-mini	90.1	–	77.5	62.0	–	54.1
SR-retrieved (matched inputs)							
GNN-RAG (SR)	Llama-2-7B	83.4	78.9	69.8	60.6	55.6	53.3
GTLM (ours)	Llama-3.1-8B	85.9	82.5	**77.9**	67.3	**64.5**	**62.1**

On matched inputs GTLM improves on GNN-RAG's reasoner-plus-reader pipeline by 8.1 F1 on WebQSP (77.9 vs. 69.8) and 8.8 on CWQ (62.1 vs. 53.3). SR's subgraphs are far smaller than the baselines' two-hop pools (median 29 nodes vs. 1530 on WebQSP, 94 vs. 1481 on CWQ). On WebQSP the coverage costs 5.2 points of ceiling (89.1 vs. 94.3) and GTLM overtakes GNN-RAG on its own retrieval regardless (77.9 vs. 71.3); on CWQ the ceilings coincide (79.0 vs. 79.5), so GTLM's 2.7-point F1 lead over GNN-RAG (62.1 vs. 59.4) reflects stronger reasoning. Appendix G reports the chosen hyperparameters and the coverage ceiling measurements.

4.4 Molecule Benchmarks

Molecule processing tasks have been attempted by approaches centered around LLMs. These methods include those using purely textual SMILES encodings (Runcie et al., 2026), and others which take an approach more similar to ours, where they modify the LLM's architecture to directly process the molecule graphs. Following the evaluation setup of Cao et al. (2025), we compare GTLM to other existing methods on **MoleculeNet** (Wu et al., 2018) property prediction (Table 4) and **ChEBI-20** (Edwards et al., 2021) molecule captioning (Table 5).

Table 4: MoleculeNet property prediction, **ROC-AUC** (↑). **Best** and second-best marked among LLM-based methods.

Method	Input	BACE	BBBP	HIV
Chemistry-pretrained specialists				
Uni-Mol (Zhou et al., 2023)	3D conformers	85.7	72.9	80.8
MolCA (Liu et al., 2023)	SMILES + graph	79.8	70.0	–
GraphCL (You et al., 2020)	graph	75.3	69.7	78.5
ChemBERTa-2 (Ahmad et al., 2022)	SMILES	73.5	69.8	79.3
Fine-tuned LLM-based methods				
InstructMol-GS (Cao et al., 2025)	graph + SMILES	82.1	**72.4**	68.9
InstructMol-G (Cao et al., 2025)	graph	**84.3**	68.6	74.0
Llama-2-7B-chat (Touvron et al., 2023)	SMILES	74.8	65.6	62.3
Vicuna-v1.3-7B (Chiang et al., 2023)	SMILES	68.3	60.1	58.1
GTLM (Llama-3.2-1B)	graph	82.0±1.2	70.6±2.3	**76.9**±1.7

Table 5: ChEBI-20 molecule captioning. **Best** and second-best marked among LLM-based methods.

Method	Input	BLEU-2	BLEU-4	ROUGE-L	METEOR
Chemistry-pretrained sequence models					
MolT5-Large (Edwards et al., 2022)	SMILES	0.594	0.508	0.594	0.614
Text+Chem T5 (Christofidellis et al., 2023)	SMILES	0.625	0.542	0.622	0.648
MolCA (Liu et al., 2023)	SMILES + graph	0.620	0.531	0.618	0.651
Fine-tuned LLM-based methods					
Mol-Instructions (Fang et al., 2024)	SELFIES	0.249	0.171	0.289	0.271
BioMedGPT-10B (Luo et al., 2023)	graph	0.234	0.141	0.332	0.308
InstructMol-GS (Cao et al., 2025)	graph + SMILES	0.475	0.371	0.502	0.509
InstructMol-G (Cao et al., 2025)	graph	0.466	0.365	0.479	0.491
GTLM (Llama-3.2-1B)	graph	0.534±0.009	0.437±0.011	0.551±0.006	0.567±0.008
GTLM (Llama-3.1-8B)	graph	**0.572**±0.003	**0.482**±0.003	**0.583**±0.002	**0.602**±0.002

We find that GTLM significantly outperforms general-purpose LoRA-tuned LLMs, and performs on par with the 7B InstructMol models on property prediction, despite their chemistry-pretrained GNN encoder. Finally, GTLM demonstrates strong performance on ChEBI-20 molecule captioning, with a 0.097 BLEU-2 advantage over the best LLM-based method InstructMol-GS (0.572 vs 0.475), nearly matching chemistry-pretrained specialist models. Additional details about these experiments are provided in Appendix H.

4.5 Long-Context Retrieval

Finally, we compare GTLM against a standard LLM under a matching training budget on a synthetic **needle-in-a-graph** task, where the model must follow a k-hop path in a graph to retrieve the answer (Table 6). As graph size N and per-node token count T grow, GTLM maintains high accuracy. The LLM collapses to 8.5% at $(N{=}128, T{=}512)$ on 4-hop questions compared to GTLM's 98.0%.

This gap is a result of two properties. First, by **Property 1** (Prefix Node Permutation Equivariance), GTLM is invariant to node order and cannot get *lost in the middle*. On the other hand, the LLM suffers up to 57 pp fluctuations across node permutations (documented in Appendix I). Second, since GTLM resets RoPE indices for all nodes individually, token positions scale with node length T rather than total input sequence length $N \times T$, enabling seamless out-of-distribution (OOD) context length generalization ($*$ cells) where standard LLMs struggle.

4.6 Ablation Study

To justify the simultaneous use of three types of attention biases, we conduct a thorough ablation study on the GraphQA benchmark, which was chosen due to its abstract nature, where the models are evaluated purely on their ability to process and reason over graph topologies, rather than textual information. Furthermore, we chose to represent the graphs in their standard format, as it isolates GTLM's ability to infer connectivity information solely from the attention biases.

Table 6: **Needle-in-a-graph performance** (%) across graph size N and token count per node T for k-hop questions ($k \in \{2, 4\}$). * marks cells outside of the training distribution.

	$k=2$								$k=4$							
	GTLM				LLM				GTLM				LLM			
$N \setminus T$	64	128	256	512	64	128	256	512	64	128	256	512	64	128	256	512
16	100	100	100	100	95.5	94.0	95.0	95.5	92.5	91.0	93.5	89.0	27.5	31.0	35.0	31.0
32	100	100	100	100	94.0	93.0	92.0	91.0	97.0	95.0	90.5	94.0	17.0	20.0	19.0	20.0
64	100	100	100	100*	90.5	90.5	87.5	87.0*	95.5	95.5	95.0	94.0*	12.0	16.0	12.0	11.5*
128	100	100	100*	100*	88.0	83.0	84.0*	78.0*	98.0	96.5	97.5*	98.0*	8.5	10.5	6.5*	8.5*

Table 7: **Ablation study (GraphQA)** results, comparing the accuracies (%) of standard GTLM to its variants without each attention bias type individually.

	Graph Tasks				Node Tasks		Edge Tasks		
Method	Node Count	Edge Count	Cycle Check	Triangle Count	Node Degree	Conn. Nodes	Reach-ability	Edge Exist.	Shortest Path
Standard	99.7±0.5	57.2±2.7	97.7±0.8	31.5±0.8	99.9±0.1	94.3±0.1	98.4±0.0	99.9±0.1	92.7±1.7
w/o SPD	99.9±0.1	59.3±5.1	96.8±0.2	31.5±0.8	99.9±0.1	93.2±0.7	98.5±0.1	99.9±0.2	93.5±0.5
w/o RRWP	100±0.0	54.4±4.5	96.9±0.2	30.7±1.1	99.9±0.2	93.3±0.4	97.9±0.1	99.6±0.2	91.9±0.6
w/o Magnetic	100±0.0	8.9±0.5	94.5±0.4	21.0±2.1	99.9±0.1	94.4±1.0	97.5±0.2	99.3±0.3	87.5±3.9

As seen in Table 7, we find that the Magnetic Bias is the most crucial for strong performance, RRWP demonstrated more modest gains only in some specific tasks, and SPD's effect is no greater than the per-seed noise. However, since the SPD bias does not introduce many new parameters or any computational overhead, we still opted to use all three bias types across our other experiments. Additionally, in Appendix K, we show that the performance gains can be attributed to both: (1) fixing the compression bottleneck and (2) the graph-derived attention biases.

5 Discussion

In this section, we look beyond the raw performance metrics to analyze the underlying mechanisms and practical properties of GTLM. We demonstrate GTLM's ability to simulate message passing, and discuss its parameter efficiency, implementation, and future research directions.

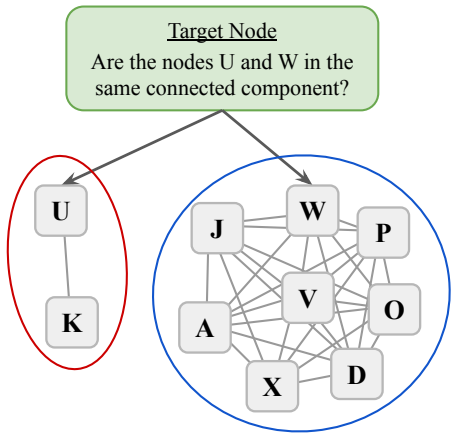
Simulated Message Passing

To illustrate the internal mechanisms of GTLM, we fine-tune the 1-billion-parameter model on a synthetic task where it must predict whether two nodes belong to the same connected component (Figure 2a). By analyzing the resulting attention weights, we observe that many attention heads learn to implicitly simulate message passing by leveraging our structural attention biases (Figure 2b).

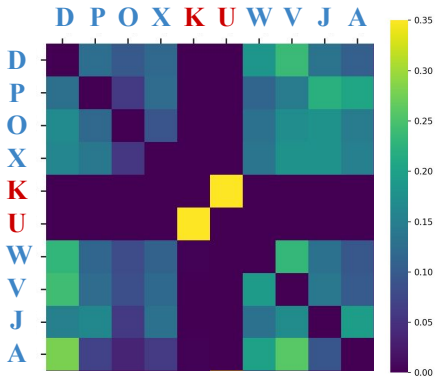
Note that the attention scores between nodes **K** and **U** and the remaining nodes are near zero, while the attention scores between same-clique nodes are higher. This experiment suggests that the GTLM architecture can natively process graphs akin to dedicated GNNs or GTs. This property explains the model's exceptionally strong performance on the GraphQA benchmark, as it tests the model's algorithmic reasoning abilities. Further details about this experiment are provided in Appendix L.

Efficiency

Parameter Efficiency. GTLM requires two types of parameters to be trained: (1) LoRA parameters, and (2) bias-related parameters. LoRA adapters are known to add only a few percent of the total base model parameters, specifically, around 2% when training a 1-billion-parameter model with rank 32. On the other hand, our bias-related parameter count is almost negligible in comparison to the total number of parameters in modern LLMs. In total, the 173,056 bias-related parameters—comprising 4,096 for SPD, 50,688 for RRWP, and 118,272 for the Magnetic Laplacian—account for only around 0.015% of the 1-billion-parameter base LLM.



(a) Visualization of the synthetic clique-detection task.



(b) Attention map on the input prefix (all nodes except for the **target node**) demonstrating learned message passing.

Figure 2: Qualitative analysis of GTLM's attention scores on a synthetic graph.

Implementation. Since the graph bias is an additive term on the attention logits, it renders the FlashAttention Dao et al. (2022) backend unusable for GTLM. Therefore, we implement the graph-bias-compatible attention backend using a PyTorch FlexAttention `score_mod` (Dong et al., 2025). Unlike the naive eager approach, our implementation does not materialize the full $L \times L$ attention matrix $\boldsymbol{A}$, nor the full token-level bias matrix; it only instantiates the node-level bias matrix $[b(u, v)]_{u,v} \in \mathbb{R}^{N \times N}$ per head and layer. We use a sequence-to-node index to perform a single lookup of the appropriate bias value inside the kernel. Since each node contributes many tokens, $N \ll L$, and this removes the $O(L^2)$ memory term the bias introduced. On our TAG benchmarks, a forward+backward step costs around $2\times$ that of a standard LLM using Flash Attention on exactly identical token sequences with no graph machinery, at exactly the same peak memory. Compared with the eager implementation, the same step time is 2-10$\times$ faster with substantially lower peak memory, removing the $O(L^2)$ bottleneck. Since the advantage only becomes meaningful at a scale beyond a few hundred tokens, we select the appropriate backend per experiment. More information and detailed scaling measurements are available in Appendix M.

Future Research

Generalist Foundation Models. Given our strong results across a wide variety of problem types related to graphs and text, alongside the theoretical guarantee that GTLM reduces to a standard LLM with text-only inputs, we believe GTLM to be a strong architecture for building a generalist foundation model for these combined modalities. This endeavor would require a large-scale corpus of various graph-related tasks for continual pretraining of the model, as well as an instruction-tuning set with graph-conditioned questions to facilitate zero-shot interactions. Ultimately, such a system could meaningfully accelerate progress in domains such as molecular discovery, logical reasoning, or enterprise knowledge retrieval, where data fundamentally is structured as a graph.

6 Conclusion

We introduced **GTLM**, a novel architecture that natively unifies graph and text modalities without relying on lossy node compression or disjointed multi-step pipelines. By mathematically aligning structural graph attention biases with the prefix-LM masking strategy, GTLM retains exact backward compatibility with its pretrained LLM foundation while achieving state-of-the-art performance on a variety of benchmarks with structured data. Beyond standard benchmark performance, we show that GTLM generalizes significantly better to longer out-of-distribution sequences than an LLM under a needle-in-a-graph multi-hop retrieval setting.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools for writing code and compiling experiment results. We have not used generative AI tools for [other tasks with required disclosure], and [the rest of the required disclosure tasks] are not applicable to this work. Additionally, we used generative AI tools for [tasks with recommended disclosure]. We have reviewed all AI-assisted work. [Elaborate. For example, "we checked LLM-generated research ideas for potential plagiarism through a manual literature survey", "LLM-generated code was verified and tested for correctness by 2 authors", etc.]. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All results presented in this paper are fully reproducible. We include detailed explanations in the Appendix sections and also provide the source code for the novel model architecture implementation and all experiments.

REFERENCES

Walid Ahmad, Elana Simon, Seyone Chithrananda, Gabriel Grand, and Bharath Ramsundar. ChemBERTa-2: Towards chemical foundation models, 2022.

He Cao, Zijing Liu, Xingyu Lu, Yuan Yao, and Yu Li. InstructMol: Multi-modal integration for building a versatile and reliable molecular assistant in drug discovery. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 354–379, Abu Dhabi, UAE, 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.25/.

Runjin Chen, Tong Zhao, Ajay Jaiswal, Neil Shah, and Zhangyang Wang. Llaga: Large language and graph assistant. *arXiv preprint arXiv:2402.08170*, 2024.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality. LMSYS blog, https://lmsys.org/blog/2023-03-30-vicuna/, 2023.

Dimitrios Christofidellis, Giorgio Giannone, Jannis Born, Ole Winther, Teodoro Laino, and Matteo Manica. Unifying molecular and textual representations via multi-task language modelling. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 6140–6157. PMLR, 2023.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flexattention: A programming model for generating fused attention variants. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025.

Carl Edwards, ChengXiang Zhai, and Heng Ji. Text2Mol: Cross-modal molecule retrieval with natural language queries. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 595–607. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.47.

Carl Edwards, Tuan Lai, Kevin Ros, Garrett Honke, Kyunghyun Cho, and Heng Ji. Translation between molecules and natural language. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 375–413. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.26.

Yin Fang, Xiaozhuan Liang, Ningyu Zhang, Kangwei Liu, Rui Huang, Zhuo Chen, Xiaohui Fan, and Huajun Chen. Mol-Instructions: A large-scale biomolecular instruction dataset for large language models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Tlsdsb6l9n.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 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.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025. URL https://arxiv.org/abs/2503.19786.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Xuanwen Huang, Kaiqiao Han, Yang Yang, Dezheng Bao, Quanjin Tao, Ziwei Chai, and Qi Zhu. Can gnn be good adapter for llms? In *Proceedings of the ACM web conference 2024*, pp. 893–904, 2024.

Yinan Huang, Haoyu Peter Wang, and Pan Li. What are good positional encodings for directed graphs? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=s4Wm71LFK4.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Mufei Li, Siqi Miao, and Pan Li. Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.20724.

Zhiyuan Liu, Sihang Li, Yanchen Luo, Hao Fei, Yixin Cao, Kenji Kawaguchi, Xiang Wang, and Tat-Seng Chua. MolCA: Molecular graph-language modeling with cross-modal projector and uni-modal adapter. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 15623–15638, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.966.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.01061.

Yizhen Luo, Jiahuan Zhang, Siqi Fan, Kai Yang, Yushuai Wu, Mu Qiao, and Zaiqing Nie. BioMedGPT: Open multimodal generative pre-trained transformer for biomedicine, 2023.

Liheng Ma, Chen Lin, Derek Lim, Adriana Romero-Soriano, Puneet K. Dokania, Mark Coates, Philip Torr, and Ser-Nam Lim. Graph inductive biases in transformers without message passing, 2023. URL https://arxiv.org/abs/2305.17589.

Costas Mavromatis and George Karypis. Gnn-rag: Graph neural retrieval for efficient large language model reasoning on knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2025*. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-acl.856/. Extended version: arXiv:2405.20139.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Meta AI. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

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

Ofir Press, Noah A. Smith, and Mike Lewis. 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.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), January 2020. ISSN 1532-4435.

Nicholas T. Runcie, Fergus Imrie, and Charlotte M. Deane. Molecular representations for large language models, 2026. URL https://arxiv.org/abs/2605.01822.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. BLOOM: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 641–651, New Orleans, Louisiana, 2018. Association for Computational Linguistics. URL https://aclanthology.org/N18-1059/.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 491–500, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Yingxu Wang, Jiaxin Huang, Mengzhu Wang, and Nan Yin. Trace: An experiential framework for coherent multi-hop knowledge graph question answering. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 1830–1851, San Diego, California, United States, 2026. Association for Computational Linguistics. URL https://aclanthology.org/2026.findings-acl.89/.

Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in neural information processing systems*, 35:27387–27401, 2022.

Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: A benchmark for molecular machine learning, 2018. URL https://arxiv.org/abs/1703.00564.

Shuwen Xu, Yao Xu, Jiaxiang Liu, Chenhao Yuan, Wenshuo Peng, Jun Zhao, and Kang Liu. Graphwalker: Agentic knowledge graph question answering via synthetic trajectory curriculum. In *Conference on Language Modeling (COLM)*, 2026. URL https://arxiv.org/abs/2603.28533.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 201–206, Berlin, Germany, 2016. Association for Computational Linguistics. URL https://aclanthology.org/P16-2033/.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? In *Advances in Neural Information Processing Systems*, volume 34, pp. 28877–28888, 2021.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems*, volume 33, pp. 5812–5823, 2020.

Chaohao Yuan, Kangfei Zhao, Ercan Engin Kuruoglu, Liang Wang, Tingyang Xu, Wenbing Huang, Deli Zhao, Hong Cheng, and Yu Rong. A survey of graph transformers: Architectures, theories and applications. *arXiv preprint arXiv:2502.16533*, 2025.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5773–5784, Dublin, Ireland, 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.acl-long.396/.

Xitong Zhang, Yixuan He, Nathan Brugnone, Michael Perlmutter, and Matthew Hirn. Magnet: a neural network for directed graphs. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Zhongjian Zhang, Xiao Wang, Mengmei Zhang, Jiarui Tan, and Chuan Shi. Toward graph-tokenizing large language models with reconstructive graph instruction tuning, 2026. URL https://arxiv.org/abs/2603.01385.

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

Gengmo Zhou, Zhifeng Gao, Qiankun Ding, Hang Zheng, Hongteng Xu, Zhewei Wei, Linfeng Zhang, and Guolin Ke. Uni-Mol: A universal 3D molecular representation learning framework. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=6K2RM6wVqKu.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep Graph Contrastive Representation Learning. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020. URL http://arxiv.org/abs/2006.04131.

A Derivations of Structural Features

This appendix provides the formal mathematical definitions for the structural features utilized by our Relative Positional Encodings.

A.1 Relative Random Walk Probabilities (RRWP)

To capture complex structural dynamics, we utilize Relative Random Walk Probabilities (Ma et al., 2023). For a given graph with an adjacency matrix $\mathbf{A}$ and a diagonal degree matrix $\mathbf{D}$, the transition matrix is defined as $\mathbf{M} := \mathbf{D}^{-1}\mathbf{A}$. The K-step walk probabilities between nodes u and v are formulated as:

$$\mathbf{P}_{u,v} = [\mathbf{I}, \mathbf{M}, \mathbf{M}^2, \ldots, \mathbf{M}^{K-1}]_{u,v} \in \mathbb{R}^K.$$

These features are subsequently processed via a learned MLP into dense scalar attention biases.

A.2 The Magnetic Laplacian

To capture directed topology, we utilize the Magnetic Laplacian (Zhang et al., 2021). We first define the symmetrized adjacency matrix as $\mathbf{A}_s(u,v) := \frac{1}{2}(\mathbf{A}(u,v)+\mathbf{A}(v,u))$. The corresponding degree matrix is defined as $\mathbf{D}_s(u,u) := \sum_{v\in\mathcal{V}} \mathbf{A}_s(u,v)$.

Directional information is captured via a phase matrix $\mathbf{\Theta}^{(q)}$, defined as $\mathbf{\Theta}^{(q)}(u,v) := 2\pi q(\mathbf{A}(u,v) - \mathbf{A}(v,u))$ for $q \geq 0$. The complex Hermitian adjacency matrix is then given by $\mathbf{H}^{(q)} := \mathbf{A}_s \odot \exp(i\mathbf{\Theta}^{(q)})$, where $\odot$ denotes componentwise multiplication. From this, the normalized Magnetic Laplacian is defined as:

$$\mathbf{L}_N^{(q)} := \mathbf{I} - (\mathbf{D}_s^{-1/2}\mathbf{A}_s\mathbf{D}_s^{-1/2}) \odot \exp(i\mathbf{\Theta}^{(q)}).$$

This matrix is Hermitian, positive-semidefinite, and thus diagonalized by an orthonormal basis of complex eigenvectors associated with real, nonnegative eigenvalues. Its spectral decomposition is given by $\mathbf{L}_N^{(q)} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\dagger$. To resolve the unitary basis ambiguity in the complex eigenvectors $\mathbf{V}$, we follow an approach similar to Huang et al. (2025), in order to construct a basis-invariant kernel:

$$\mathbf{K} = \mathbf{V}\mathrm{diag}(\phi(\lambda))\mathbf{V}^\dagger \in \mathbb{C}^{N\times N\times d_{\text{Mag}}},$$

where $\phi(\lambda)$ is a learned permutation-invariant transformation of the eigenvalues. This invariant kernel provides a mathematically robust foundation for extracting relative spectral distances between nodes.

B Theoretical Proofs and Empirical Verification of Architectural Properties

In this section, we provide the formal mathematical proofs for the architectural guarantees introduced in Section 3.3, followed by their corresponding empirical and numerical validations.

B.1 Formal Mathematical Proofs

Proof of Property 1: Prefix Node Permutation Equivariance

Claim: Let $\mathbf{X} = [\mathbf{x}_{v_1} \oplus \cdots \oplus \mathbf{x}_{v_N}]$ be the serialized sequence of token embeddings for the N prefix context nodes. GTLM is equivariant with respect to any permutation $\pi \in S_N$ applied to the order of these nodes, meaning $\text{GTLM}(\pi \cdot \mathbf{X}) = \pi \cdot \text{GTLM}(\mathbf{X})$, and strictly invariant with respect to the generated target sequence.

Proof: In a standard Transformer layer, the updated representation for a token i inside node u is computed as a weighted sum of value vectors V_j from tokens $j \in v$, weighted by the attention matrix $\text{softmax}(A_{i,j})$. Equivariance holds if and only if the set of attended tokens and their corresponding attention scores $A_{i,j}$ remain completely independent of the permutation π. We demonstrate this across the three relevant components of the attention mechanism:

1. **Attention Masking:** GTLM applies a fully-visible, non-causal mask over the entire prefix. Therefore, the set of tokens that token i attends to is exactly the entire prefix set $\bigcup_{k=1}^{N}\{t \in v_k\}$, which is invariant to the serialization order π.
2. **Positional Encodings (RoPE):** In GTLM, the 1D positional index for RoPE is reset to 0 at the start of every node. Consequently, the query Q_i and key K_j representations depend exclusively on their local token indices within their respective nodes u and v, strictly decoupling them from the global flattened sequence index imposed by π.
3. **Structural Biases:** The raw attention score is $A_{i,j}^{(l,h)} = \frac{Q_i \cdot K_j^\top}{\sqrt{d_{\text{head}}}} + b^{(l,h)}(u,v)$. The bias term $b^{(l,h)}(u,v) = b_{\text{SPD}} + b_{\text{RRWP}} + b_{\text{Mag}}$ is computed using static adjacency matrices, transition probabilities, and Laplacian spectra derived from the immutable graph topology $\mathcal{G}$. These graph-level properties are inherently permutation invariant.

Because the mask, the representations Q and K, and the structural biases $b(u,v)$ are all mathematically invariant to π, the computed attention scores $A_{i,j}$ for any given pair of tokens remain identical regardless of the node ordering. Thus, the resulting token representations are identical in value and simply undergo the same permutation π as the input, satisfying exact permutation equivariance for the prefix context.

Proof of Property 2: Pretraining Backward Compatibility

Claim: For any graph $\mathcal{G}$ consisting of a single node (representing a standard text sequence), the GTLM forward pass is mathematically identical to that of the unmodified base LLM.

Proof: Consider a single-node graph $\mathcal{G} = (\{u\}, \emptyset)$. For any two tokens i and j, we have $u = v$, and the attention score between those two tokens within the node u is defined as:

$$A_{i,j}^{(l,h)} = \frac{(Q_i^{(l,h)}) \cdot (K_j^{(l,h)})^\top}{\sqrt{d_{\text{head}}}} + b_{\text{SPD}}^{(l,h)}(u,u) + b_{\text{RRWP}}^{(l,h)}(u,u) + b_{\text{Mag}}^{(l,h)}(u,u).$$

By the architectural definitions established in Section 3.2, each structural bias natively evaluates to zero for intra-node interactions:

1. $b_{\text{SPD}}^{(l,h)}(u,u) = 0$, as the shortest path distance from a node to itself is exactly 0, which is explicitly mapped to a zero-bias vector.
2. $b_{\text{RRWP}}^{(l,h)}(u,u) = 0$, enforced by the piecewise formulation where $u = v$.
3. $b_{\text{Mag}}^{(l,h)}(u,u) = 0$, enforced by the piecewise formulation where $u = v$.

Summing these terms completely cancels the structural biases, reducing the computation to $A_{i,j}^{(l,h)} = \frac{Q_i \cdot K_j^\top}{\sqrt{d_{\text{head}}}}$. Furthermore, because the sequence consists of only one node, the per-node RoPE reset behaves identically to standard RoPE over a single continuous sequence. Causal masking constraints apply to the target sequence exactly as they do in the base model. Thus, all matrix operations in the attention mechanism exactly mirror those of standard scaled dot-product attention, guaranteeing zero degradation of pretrained capabilities.

B.2 Empirical Verification

To complement the theoretical proofs provided above, we numerically verify both architectural guarantees. Because these properties are fundamental to the model's forward pass and operate independently of the learned weights, we evaluate them using the base `Llama-3.2-1B` model with randomly initialized bias-related parameters.

Backward Compatibility. We construct a trivial graph consisting of a single **target node** with no edges, assigned a randomly generated textual attribute. We perform a forward pass on this input using both our GTLM architecture and the unmodified base Llama model. We monitor the maximum absolute difference between the output logits of the two models. Across five independent trials, we found this difference to be $2.1 \times 10^{-5} \pm 8.3 \times 10^{-6}$. This delta is entirely within the expected floating-point (FP32) accumulation error, numerically confirming exact backward compatibility.

Permutation Equivariance. We construct a multi-node test graph with a fixed topology. We then generate two flattened input sequences by randomly permuting the serialization order of the prefix nodes, while strictly preserving their textual attributes and the underlying graph structure. Both permuted sequences are processed by GTLM, and the outputs are inversely permuted to align with the canonical node order. We measure the maximum absolute difference in the final logits between the two permuted sequences. The result we get across five independent trials is $2.77 \times 10^{-5} \pm 2.87 \times 10^{-6}$. These results confirm that our architecture is strictly equivariant to the arbitrary ordering of nodes in the flattened sequence.

Code to reproduce these numerical verification checks is available in our codebase alongside the main training framework. Note that we report the numbers as mean $\pm$ sample standard deviation.

C General GTLM Training Setup

Attention Bias Hyperparameters. Unless stated otherwise, all GTLM experiments employ a fixed configuration for the structural attention biases. Specifically, we fix the maximum distance for Shortest Path Distance (SPD), the maximum steps for Relative Random Walk Probabilities (RRWP), and the q value and dimensionality for Magnetic Laplacian edge-level embeddings. The parameter values are shown in Table 8.

Table 8: Graph-aware attention bias parameter values.

Parameter	Value
SPD max dist.	8
max RRWP steps	16
Magnetic Lap. q	0.25
Magnetic Lap. dim.	32

Learning rates. We utilize a differential learning rate strategy, assigning a higher rate to the structural attention bias parameters than to the LoRA adapters. This approach is motivated by two factors: (1) the bias parameters are initialized randomly and must learn to map graph topology to the attention space, whereas the LLM and adapters start from highly refined, pretrained weights, and (2) despite their low parameter count, these biases are the primary mechanism for graph-awareness and require a stronger gradient signal to converge. This ensures that the essential structural features are learned effectively without being overshadowed by the larger, pretrained components during optimization.

D GTLM with ALiBi and dual RoPE

In order to demonstrate the generality of the GTLM architecture, specifically the per-node position index resetting idea, we show that it is possible to implement with **ALiBi** and **dual RoPE**. We use BLOOM (Scao et al., 2022) as base model for the former, and Gemma-3-1B (Gemma Team, 2025) for the latter.

ALiBi. ALiBi adds a fixed distance penalty per head to the attention logits before softmax. GTLM's structural mask and graph bias are added in the same slot, so BLOOM's attention requires no changes. The only adaptation is to calculate ALiBi distances from GTLM's node-position identifiers instead of the raw token order. This prevents the node order from acting as a spurious prior, and the single-node graph still accurately reproduces standard BLOOM.

Dual RoPE. GTLM only changes the position IDs provided to RoPE, not the rotation, so it does not depend on the underlying frequency of RoPE. The local (10k) and global (1M) Gemma-3 layers rotate the same position identifiers per node. Each layer therefore encodes the order of tokens within a node, and the graph bias, added identically in each layer, carries the structure between nodes. The 512-token sliding window is discarded because it follows the serialization order, not the graph. At GraphQA sequence lengths (maximum 126 tokens), this change has no effect.

We provide the results for both and compare them to the results obtained with Llama-3.2-1B as the base model (Table 9).

Table 9: GraphQA per-task accuracy (%) across GTLM base models with different positional encodings (**best**, second-best).

	Graph Tasks				Node Tasks		Edge Tasks			
GTLM Base Model	Node Count	Edge Count	Cycle Check	Triangle Count	Node Degree	Conn. Nodes	Reach-ability	Edge Exist.	Shortest Path	Avg.
Llama-3.2-1B (RoPE)	99.7±0.5	**57.2**±2.7	**97.7**±0.8	**31.5**±0.8	**99.9**±0.1	94.3±0.1	**98.4**±0.0	**99.9**±0.1	**92.7**±1.7	**85.7**
bloom-1b1 (ALiBi)	**100**±0.0	41.5±2.8	97.5±1.3	31.2±0.8	94.4±3.1	**98.5**±0.5	**98.4**±0.2	99.7±0.1	**92.7**±0.8	83.8
gemma-3-1b-pt (dual RoPE)	**100**±0.0	39.7±2.0	97.3±0.5	25.8±1.0	95.6±1.4	85.5±1.1	98.3±0.3	**99.9**±0.1	92.2±0.7	81.6

E GRAPHQA

The **incidence graph** is a transformation of a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ into a bipartite directed graph $\overline{\mathcal{G}} = (\overline{\mathcal{V}}, \overline{\mathcal{E}})$ as follows:

$$\overline{\mathcal{V}} = \mathcal{V} \cup \mathcal{E}$$
$$\overline{\mathcal{E}} = \{(u, e), (e, v) \mid e = (u, v) \in \mathcal{E}\}$$

In this bipartite formulation, every original directed edge $e = (u, v)$ is effectively represented as a node in $\overline{\mathcal{V}}$ and is replaced by a path of length two: $u \to e \to v$. This transformation ensures that both vertices and edges from the original graph are treated uniformly as nodes in the incidence graph. Consequently, any edge attributes or topological properties can be seamlessly processed by GTLM's structural attention biases alongside the node data.

Graph Construction. We use the graphs from the publicly available GraphQA dataset and set the textual attributes of each node to its own unique ID. When using incidence graphs, the original edges are given the text attributes with format "(i, j)", where i and j are the nodes the given edge connects.

Training Hyperparameters used to train GTLM on the *standard* and *incidence* graphs are the same and shown in Table 10.

Table 10: Hyperparameter values used to train GTLM on the GraphQA benchmark.

Parameter	**Value**
Epochs	20
Learning Rate (η)	3×10^{-5}
Bias Learning Rate (η_{bias})	5×10^{-3}
LoRA Rank (r)	16
LoRA Alpha (α)	32

F TAG BENCHMARKS

Dataset splits. The dataset splits for these four benchmarks were taken from Zhang et al. (2026) to ensure a fair comparison to the RGLM models and their baselines.

Neighborhood sampling. To construct the inputs for GTLM, we construct ego subgraphs for each node with up to N (30 or 60) 2-hop neighbors. In cases when there were more than N 2-hop neighbors, they were chosen based on closeness, while equidistant nodes were sampled randomly.

Text attributes. For the three benchmarks with paper citation graphs we employ the following strategy: the target paper contains the full title and abstract, while all others contain only the title of the paper. On the other hand, after some experimenting, we found that simply truncating the post contents in the *Reddit* dataset to around 25 tokens per node yielded strong results while maintaining manageable context lengths.

Question + Answer injection. In order to train the model for autoregressive generation solely on the `label` textual sequence, we append the following string to the end of the target node:
`\n\n{task_question}\n A: {label}`

Training Hyperparameters. We conducted a search over multiple parameters with all options shown and the chosen setup highlighted, as seen in Table 11. For the neighborhood sizes we chose between 30 and 60 based on the context length such that the average total sequence length did not exceed 1000 tokens, which was a heuristic choice to balance training speeds and performance.

Table 11: **Hyperparameter Configurations.** Final values and search spaces for each dataset. For learning rates and LoRA rank, the search set is provided with the selected optimal value **bolded**.

Parameter	Cora	PubMed	ogbn-arxiv	Reddit
Neighbors (k)	60	30	60	30
Epochs	20	4	1	15
Learning Rate (η)	{6e-5, 2e-4, **3e-4**}	{6e-5, **2e-4**, 3e-4}	{6e-5, **2e-4**, 3e-4}	{**6e-5**, 2e-4, 3e-4}
Bias LR (η_{bias})	{0.01, **0.04**}	{0.01, **0.04**}	{0.01, **0.04**}	{**0.01**, 0.04}
LoRA Rank (r)	{32, **64**}	{**32**, 64}	{32, **64**}	{**32**, 64}
LoRA Alpha (α)	$2r$	$2r$	$2r$	$2r$

G GRAPHRAG

As input to GTLM we use the SR subgraphs (Zhang et al., 2022), which we downloaded from GNN-RAG's public Google Drive folder[1]. We convert each subgraph into a directed Levi graph, in which every triple (h, r, t) becomes a path $h \to r \to t$. Entity nodes are verbalized by their Freebase names from `entities_names.json`, which we extend with Freebase name aliases, and relation nodes by the last segment of the relation identifier (e.g. `people.person.place_of_birth` becomes "place of birth"); graphs are capped at 512 nodes for WebQSP and 1024 for CWQ. The question is placed in its own node in the prefix with no edges to the rest of the graph, so the structural encodings treat it as a separate component while every graph token can still attend to it, and the answer is generated from a separate prompt (target) node connected to the topic entities. We use the standard train, dev and test splits of both benchmarks: training keeps only questions whose answer appears in the retrieved subgraph, while dev and test keep every question and score unanswerable ones as 0, so our evaluation denominators match those of RoG and GNN-RAG.

Table 12 lists the hyperparameters required to reproduce our results with the 1B (Llama-3.2-1B) and 8B (Llama-3.1-8B) base models on WebQSP and CWQ. For hyperparameters that were tuned, the table gives every value we tried and marks the one we selected.

Table 12: **KGQA Hyperparameter Configurations.** Final values and search spaces for WebQSP and CWQ across 1B and 8B backbones. Where a search set is provided, the selected optimal value is **bolded**.

	Llama-3.2-1B		Llama-3.1-8B	
Parameter	**WebQSP**	**CWQ**	**WebQSP**	**CWQ**
Max Nodes	512	1024	512	1024
Max Answers ($n_{\max}$)	50	50	50	50
Epochs	15	8	15	8
Learning Rate (η)	{5e-5, **1e-4**}	{5e-5, **1e-4**}	{**5e-5**, 1e-4}	{**5e-5**, 1e-4}
Bias LR (η_{bias})	{**5e-3**, 1e-2}	5e-3	5e-3	5e-3
LoRA Rank (r)	{16, **64**}	64	{16, **64**}	64
LoRA Dropout	{**0.15**, 0.25}	0.15	0.15	0.15

Table 13 gives the upper bound on the response coverage for each input condition on the test split, i.e. the best result that any reader can achieve: the Hits@1 ceiling is the fraction of questions with at least one gold answer in the input graph, and the F1 ceiling is the macro F1 of a reader that outputs exactly the gold answers present. Our node cap costs at most 0.8 points. On WebQSP, the 2-hop

[1] `data.zip` and `entities_names.json` files from: `https://drive.google.com/drive/folders/1ifgVHQDnvFEunP9hmVYT07Y3rvcpIfQp`

pool that RoG and GNN-RAG retrieve from has a 5.2-point higher F1 ceiling than our SR inputs, while on CWQ the two ceilings are almost equal.

Table 13: **KGQA Answer-Coverage Ceilings** on the test split (%).

	WebQSP		CWQ	
Input condition	Hits@1	F1	Hits@1	F1
RoG / GNN-RAG 2-hop pool	95.6	94.3	80.7	79.5
SR, uncapped	91.1	89.6	80.7	79.8
SR, as used (512 / 1024 nodes)	90.9	89.1	79.9	79.0

H MOLECULE BENCHMARKS

For property prediction, we use the BACE, BBBP, and HIV classification sets from MoleculeNet (Wu et al., 2018), split using DeepChem's deterministic Bemis–Murcko scaffold splitter (80/10/10), the split used by the baselines we compare against. For captioning, we use ChEBI-20 (Edwards et al., 2021) with its own splits for training, validation, and testing[2]. We parse each SMILES string using RDKit into a heavy-atom graph and convert it into a Levi graph, where atoms and bonds are nodes, and each atom and bond incidence is an edge in both directions. Atom nodes are verbalized by the features a standard GNN featuriser receives, namely element, aromaticity, ring membership, heavy-atom degree, hydrogen count, formal charge, and chiral tag (e.g., "carbon aromatic ring deg3 H1"), and bond nodes by their bond order and ring membership (e.g., "double bond in ring"), so the model never sees a SMILES string. Similar to the GraphRAG experiments, the question is placed in its own node in the graph's prefix with no edges to the rest of the graph, and the answer is generated from a separate prompt (target) node, which here is connected to every atom node because the questions concern the molecule as a whole. Property prediction is posed as a yes/no question (e.g. "Does this molecule penetrate the blood-brain barrier?"), and we score ROC-AUC from the margin between the "Yes" and "No" logits at the answer position; captioning answers "Describe this molecule." with the reference caption.

Tables 14 and 15 list the hyperparameters required to reproduce our results with the 1B (Llama-3.2-1B) and 8B (Llama-3.1-8B) base models. All runs use a one-epoch linear warm-up, followed by a cosine decay to one-tenth of the peak learning rate. For property prediction, we evaluate on the validation split at a fixed step interval and report the test result of the checkpoint with the best validation ROC-AUC; for captioning, we report the final checkpoint. For the hyperparameters that are tuned, the tables give each value we tried and indicate the one we chose.

Table 14: **Property Prediction Hyperparameter Configurations.** Final values and search spaces for BACE, BBBP and HIV across 1B and 8B backbones. Where a search set is provided, the selected value is **bolded**.

	Llama-3.2-1B			Llama-3.1-8B		
Parameter	**BACE**	**BBBP**	**HIV**	**BACE**	**BBBP**	**HIV**
Epochs	40	40	10	40	40	10
Global Batch Size	32	32	32	32	32	32
Learning Rate (η)	{3e-5, **3e-4**}	{3e-5, **3e-4**}	{3e-5, **1e-4**, 3e-4}	{1.5e-4, **3e-4**}	{**1.5e-4**, 3e-4}	{**5e-5**, 1e-4}
Bias LR (η_{bias})	1e-2	1e-2	1e-2	1e-2	1e-2	1e-2
LoRA Rank (r)	{**16**, 32}	{**16**, 32}	{**16**, 32}	16	16	16
LoRA Dropout	0.05	0.05	0.05	0.05	0.05	0.05
Max SPD Distance	{**32**, 64}	{**32**, 64}	{**32**, 64}	32	32	32

[2] `ChEBI-20_data` directory `https://github.com/blender-nlp/MolT5`

Table 15: **ChEBI-20 Hyperparameter Configurations.** Final values and search spaces for molecule captioning across 1B and 8B backbones. Where a search set is provided, the selected value is **bolded**.

Parameter	**Llama-3.2-1B**	**Llama-3.1-8B**
Max Heavy Atoms	128	128
Epochs	12	12
Batch Size	64	64
Learning Rate (η)	{5e-5, 2e-4, 3e-4, 5e-4, **1e-3**}	{1e-4, 3e-4, **5e-4**}
Bias LR (η_{bias})	{**5e-3**, 2e-2}	5e-3
LoRA Rank (r)	16	16
LoRA Dropout	0.05	0.05
Max SPD	32	32

I NEEDLE-IN-A-GRAPH EXPERIMENT

Task. Each graph has N nodes: a question node, $N-2$ content nodes, and a prompt (target) node where the answer is generated. Each content node v holds exactly T tokens of WikiText-103 (Merity et al., 2017) filler text with one record sentence inserted at a random offset:

> `The access code for` $\mathrm{id}(v)$ `is` $\mathrm{code}(v)$`. Continue at` $\mathrm{id}(\mathrm{succ}(v))$`. Decoy reference:` $\mathrm{id}(\mathrm{dec}(v))$`.`

The question asks "*Starting at* $\mathrm{id}(c_0)$*, follow* k *'Continue at' references, ignoring any decoy references. What is the access code there?*", and the target is $\mathrm{code}(c_k)$ with $c_{i+1} = \mathrm{succ}(c_i)$. Both pointers (the correct and decoy) become untyped directed edges $v \to \mathrm{succ}(v)$ and $v \to \mathrm{dec}(v)$, so about 2^k nodes lie at distance k from c_0 and the path can only be resolved by reading the node texts. An oracle using only shortest-path distances scores 15.6% at N=32, k=3, where GTLM scores 99%, proving GTLM could not achieve those results with just an $O(1)$ retrieval by decoding structural features. Furthermore, we ensure that the full graph topology is fully derivable from the text the LLM receives, making the comparison fair. The question node is connected to every content node, and the prompt node has no edges.

Data. Identifiers (e.g. `NODE-0728`) are sampled without replacement from a pool of $4{,}096$. Codes follow the pattern `LLDLL` (e.g. `AB1CD`) and tokenize to exactly three tokens. To make the cells comparable, each test graph is generated once and reused across the grid. A larger N adds nodes around the same path, a larger k extends the same path from the same starting node, and a different T only changes the amount of padding around each record. Node order is shuffled. The $16{,}000$ training graphs sample $k \in \{1, 2, 3, 4\}$, then N, then T uniformly among the cells whose input fits into $16{,}384$ tokens. The cells $(N, T) \in \{(64, 512), (128, 256), (128, 512)\}$ (32k–65k tokens) are never seen in training (* in Table 6). Each (N, T, k) cell is tested on 200 graphs.

Models. Both models fine-tune Llama-3.2-1B with LoRA under the same budget (Table 16). The LLM receives the node texts in random order, each preceded by `## Article` i, followed by the question and `Answer:`. We report exact match on the three code tokens, taking the greedy prediction at each answer position given the prefix, for the seed with the median validation accuracy out of three. Over all three seeds, the LLM averages 81.1% (k=2) and 21.1% (k=4) across the cells of Table 6, against 89.9% and 17.9% for the median seed.

Node-order sensitivity. We re-evaluate the checkpoints of Table 6 on the same test graphs, changing only the node order. The $k+1$ path nodes form a grouped block, in path order (fwd) or reversed (rev), inserted at one of five relative positions among the other nodes, whose relative order is unchanged. Together with the original random order, this gives 11 orderings per graph. For GTLM, reordering permutes the node indices together with the structural encodings and edges.

We use all 200 test graphs, all three LLM seeds, and the reported GTLM seed. Because every graph is scored under every ordering, comparisons are paired. We report 95% CIs from a bootstrap that

Table 16: **Needle-in-a-Graph Hyperparameters.** Both models use Llama-3.2-1B, a 10% linear warm-up and cosine decay to one-tenth of the peak learning rate. We validate every 500 steps and keep the best checkpoint.

Parameter	**GTLM**	**LLM**
Training Graphs / Epochs	16,000 / 2	16,000 / 2
Max Training Length (tokens)	16,384	16,384
Global Batch Size	8	8
Learning Rate (η)	1e-4	1e-4
Bias LR (η_{bias})	5e-3	–
Weight Decay	0.1	0.1
LoRA Rank / α / Dropout	64 / 128 / 0.15	64 / 128 / 0.15

resamples graphs (keeping a graph's seeds together), and p-values from exact McNemar tests on the graphs whose correctness changes between two orderings.

Table 17: **Node-order sensitivity.** Accuracy (%) on the same 200 test graphs per cell when only the node order changes. The path block is in path order (fwd) or reversed (rev); *End* is adjacent to the question, and *Random* is the order of Table 6. LLM values are means over three seeds. GTLM scores identically under all 11 orderings. * marks cells outside the training distribution.

			LLM						**GTLM**
k	(N, T)	Block	Start	0.25	Middle	0.75	End	Random	All orders
2	$(32, 512)$	fwd	91.8	**94.0**	90.2	83.7	80.8	84.2	100
		rev	75.7	73.7	69.5	64.3	66.5		
	$(128, 128)$	fwd	**94.0**	88.8	80.0	70.7	63.5	74.2	100
		rev	87.3	52.8	45.0	37.3	40.8		
	$(128, 512)$*	fwd	**81.3**	77.3	68.3	65.0	62.3	68.8	100
		rev	76.8	32.2	29.3	38.3	45.0		
4	$(32, 512)$	fwd	**25.0**	20.8	22.3	18.5	17.7	23.8	94.0
		rev	5.8	5.8	5.7	7.0	10.2		
	$(128, 128)$	fwd	10.5	**11.5**	8.0	5.2	5.7	12.0	96.5
		rev	5.3	1.3	0.7	1.3	1.3		
	$(128, 512)$*	fwd	4.7	6.0	5.7	6.0	**8.7**	7.3	97.5–98.0
		rev	1.8	0.5	0.8	1.8	4.2		

With the checkpoint and inputs fixed, node order alone changes the LLM's accuracy by up to 56.7 percentage points (pp; Table 17, k=2, $(128, 128)$). Reversing the path costs between 4 and 27 pp, averaged over positions. Accuracy is generally highest with the path near the start of the context and lowest with it next to the question (up to 30.5 pp lower), so the effect is not U-shaped as it would be in the *lost in the middle* phenomenon. These differences are significant in every cell: the 95% CI of the reversal cost excludes zero everywhere, and start vs. end has $p < 0.004$ everywhere ($p < 10^{-7}$ in all k=2 cells).

At k=2, placing the path first recovers much of the loss from larger graphs: $(128, 128)$ reaches 94.0%, above the LLM's 88.2% average on the N=16 cells. At k=4, no ordering exceeds 25%. GTLM's prediction is unchanged in $11{,}997$ of $12{,}000$ (graph, ordering) pairs, and its mean gold log-probability varies by at most 0.01 nats, against 0.5–3.6 for the LLM. This is the invariance of Property 1, up to floating-point summation order.

J Scaling GTLM to Larger Base Models

In the process of optimizing GTLM's performance on the various benchmarks mentioned in this paper, we examined the dynamics of scaling the base model beyond 1B parameters. As seen in Table 18, we find that GTLM benefits the most from scaling in cases when the task involves open-

ended generation (tasks (2) and (4)), while it did not contribute in a statistically significant way with the classification tasks (1) and (3).

Table 18: GTLM scaling performance across Llama-3.2-1B, Llama-3.2-3B, and Llama-3.1-8B backbones. Δ_{rel} denotes the relative percentage improvement from 1B to 8B.

Task / Metric	1B	3B	8B	Δ_{rel}
(1) TAG Node Classification				
Cora (Acc)	88.99±1.12	89.17±0.70	89.25±0.36	+0.3%
Cora (F1)	91.31±0.90	91.58±0.97	91.60±0.53	+0.3%
Pubmed (Acc)	94.57±0.11	94.52±0.28	94.67±0.08	+0.1%
Pubmed (F1)	94.76±0.10	94.72±0.18	94.93±0.11	+0.2%
(2) Knowledge Graph QA				
WebQSP (Hit)	83.25±1.00	–	85.9	+3.2%
WebQSP (H@1)	78.03±1.15	–	82.5	+5.7%
WebQSP (F1)	73.51±0.76	–	77.9	+6.0%
CWQ (Hit)	60.75±0.68	–	67.3	+10.8%
CWQ (H@1)	57.37±0.23	–	64.5	+12.4%
CWQ (F1)	55.23±0.45	–	62.1	+12.4%
(3) Molecule Property Prediction				
BACE (ROC-AUC)	82.0±1.2	–	79.8	−2.7%
BBBP (ROC-AUC)	70.6±2.3	–	71.6	+1.4%
HIV (ROC-AUC)	76.9±1.7	–	75.5	−1.8%
(4) ChEBI-20 Molecule Captioning				
BLEU-2	0.534±0.009	–	0.572±0.003	+7.1%
BLEU-4	0.437±0.011	–	0.482±0.003	+10.3%
ROUGE-L	0.551±0.006	–	0.583±0.002	+5.8%
METEOR	0.567±0.008	–	0.602±0.002	+6.2%

The task-specific hyperparameters are highlighted in their appropriate Appendix sections.

K Performance Attribution

In order to isolate where the performance gains come from, we conduct a smaller experiment where we compare the standard GTLM architecture to a variant without structural attention biases ($\forall u, v : b(u, v) = 0$). Table 19 outlines the results of this experiment, which helps us attribute the performance to the two mechanisms GTLM innovates with:

1) fixing the **compression bottleneck** – most impactful in the TAG benchmarks, as the difference between text-only GTLM and standard GTLM is not as large;
2) structural **attention biases** – most impactful in tasks where the question asks about structure, as demonstrated with WebQSP F1 score and GraphQA.

Table 19: Comparison between standard GTLM and its text-only variant (without structural attention biases).

	Cora		PubMed		WebQSP	GraphQA
Method	Acc	F1	Acc	F1	F1	Task-mean Acc
GTLM (text-only)	86.78±0.57	90.17±0.35	93.98±0.12	94.26±0.12	48.5	33.2
GTLM (standard)	88.99±1.12	91.31±0.90	94.57±0.11	94.76±0.10	73.5	85.7

L Simulated Message Passing—Experiment Details

To investigate GTLM's internal learned mechanisms (as discussed in Section 5), we designed a controlled synthetic connectivity task. This section outlines the high-level methodology used to

isolate this emergent behavior, while full implementation details and hyperparameter configurations are available in our project codebase.

L.1 Dataset Generation

We programmatically generated a dataset of random graphs to evaluate the model's ability to deduce whether two arbitrary nodes belong to the same clique. Each graph is generated with two or more isolated connected components. To prevent the model from relying on language priors, node attributes are restricted to randomly assigned single letters.

To query the model, we introduce the *prompt node* containing the textual question (e.g., "`Are the nodes {x} and {y} connected? [Yes/No]`"). This prompt node is explicitly linked via directed edges to the two nodes in question. The dataset is strictly balanced to contain an equal number of positive examples (targets share a connected component) and negative examples (targets belong to disjoint components).

L.2 Training and Evaluation Setup

Model Configuration. We utilize `Llama-3.2-1B` as the base model, augmented with our full suite of structural attention biases (SPD, RRWP, and Magnetic Laplacian). To ensure parameter efficiency, the base model is frozen, and we **do not** apply any Low-Rank Adapters (LoRA).

Optimization Strategy. We only train the randomly initialized bias-related parameters in our model. This setup allows us to only observe the effects of our newly introduced structure-aware attention biases on the model's behavior.

Findings. We find that the model learns to solve this task with close to 100% accuracy. We further investigate the internal attention scores on one synthetic example and notice that multiple attention heads have learned to simulate message passing. For the visualization in Section 5 we used one specific attention head which best illustrated the point, while many other attention heads also exhibited a clear awareness of the graph structure—either through a similar "message passing" mechanism or its inverse where disconnected nodes have high attention scores, etc.

In summary, this experiment suggests that our GTLM architecture has the capacity for simulating the functionalities of dedicated GNNs and conventional GTs.

M Flex Attention with GTLM

M.1 Implementation

The fused FlashAttention kernels are ineligible due to the additive per-head and per-layer graph bias. The eager implementation materializes the bias at token level, (B, H, L, L), and this is what would bound the sequence lengths GTLM could train on—not the architecture. However, our backend keeps the bias at node level instead, storing only $[b(u, v)]_{u,v} \in \mathbb{R}^{N\times N}$ per head and layer and gathering the relevant entry inside the kernel through a sequence-to-node index $\pi : \{1, \dots, L\} \to \{1, \dots, N\}$, so the `score_mod` is a single lookup:

```
def score_mod(score, b, h, q, kv):
    return score + node_bias[b, h, node_id[q], node_id[kv]]
```

Since the kernel never depends on how the `node_bias` was produced, any bias type we define is compatible with these kernels, making the model's perception of graphs very easy to modify to experiment with different bias types. The backend requires L to be padded to a multiple of the block size. Further, to overcome excessive kernel re-compiling, we pad the sequences and node features into a few dozen predefined (N, L) buckets. Decoding uses a hybrid approach—due to the aforementioned requirement of FlexAttention to have discrete shape buckets—we route the prefill phase through our FlexAttention kernel, while each of the next autoregressive generation steps falls back to eager attention.

M.2 COST ON THE TEXT-ATTRIBUTED-GRAPH BENCHMARKS

Table 20 shows training-step latency over 24 randomly sampled real batches per benchmark. All arms receive identical tokens through the collator and the same padding method is used. We compare our own FLEX implementation to the naive eager backend for GTLM, as well as a standard LLM without any graph machinery. We find that FLEX is much faster and significantly more stable than the EAGER counterpart, while only $2\times$ slower than the LLM floor.

Table 20: Training-step latency on the TAG benchmarks (H100 80GB, bf16, $B = 1$, LoRA, gradient checkpointing enabled). L is the mean unpadded token count per graph.

		step latency (ms)			ratio	
dataset	L	EAGER	FLEX	FLASH	eager/flex	flex/flash
cora	663	836.9	226.8	118.6	3.69×	1.91×
pubmed	950	1,255.3	215.9	112.1	5.81×	1.93×
reddit	976	406.3	201.3	103.6	2.02×	1.94×
ogbn-arxiv	1,065	2,006.2	208.3	103.7	9.63×	2.01×

The consistency arguably matters even more than some multiplier: FLEX lands at 201–227 ms and FLASH at 104–119 ms on every dataset, while only EAGER varies significantly, from 406 to 2,006 ms. Peak memory falls with it ($6.33 \rightarrow 4.85$ GB on cora, matching FLASH), and one epoch of ogbn-arxiv falls from 50.7 to 5.3 hours.

M.3 SCALING

Figure 3 measures one layer of attention in isolation, where there is an $O(L^2)$ bias materialization. Relative to the dense path—the only baseline that computes the same function—our advantage scales with length: 3.1×, 6.2×, and 8.5× less time, and 4.6×, 9.6×, and 14.9× less memory, at 256, 1k, and 4k tokens, respectively. Above $L \approx 4.5$k the dense path completely exhausts the 80 GB device, while ours continues to $L \approx 70$k at less than 5 GB. Eager attention pays $O(L^2)$ memory per layer for the token-level bias; ours pays $O(N^2)$ on top of flash attention's $O(L)$, and $N \ll L$ since each node contributes many tokens.

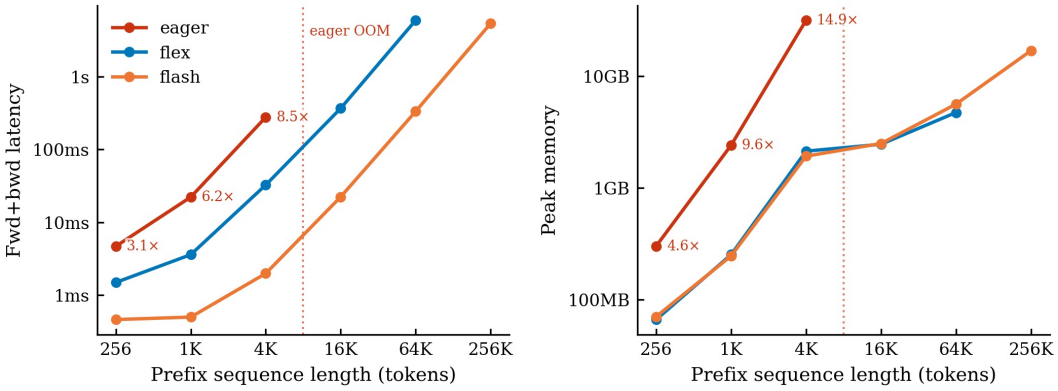


Figure 3: Forward+backward latency (left) and peak memory (right) for one attention layer against packed sequence length, on synthetic graph batches with real topology and shortest-path distances. Annotations on the dense curve give its cost relative to ours, and the dotted line marks where it exhausts memory. Flash attention is shown as a bias-free lower bound; it cannot express the graph bias.

Compilation is the only additional cost: one autotuned kernel per (L, N) bin, 176 s on first encounter and 12 s from disk cache, over the four to ten bins that the TAG dataset produces. Amortized over tens of thousands of steps, it is negligible compared to the 2.0-9.6× faster step per fraction of memory, at lengths that dense traversal cannot reach at all.

N Computational Resources and Hardware Setup

To ensure full reproducibility and provide transparency regarding the computational requirements of our architecture, we detail the hardware setup and empirical training times below.

Although some of our experiments used higher end graphics cards such as Nvidia's B200 or B300, all experiments can be conducted using a single NVIDIA A100 or H100 GPU (80GB VRAM). When fine-tuning the base `Llama-3.2-1B` model equipped with our structural LoRA adapters, training times on the individual GraphQA tasks ranged from 10 minutes (for standard graphs) to 40 minutes (for incidence graphs). The more demanding Text-Attributed Graph (TAG) benchmarks required between 6 and 8 hours of continuous training on an H100 GPU (with the exception of *OGBN-Arxiv*, which required around 20 hours). Other experiments were carried out on those higher-end graphics cards, where training times ranged between 1 and 10 hours, depending on the task in question and the training configuration. Finally, training the 3B and 8B model variants introduced approximately $2\times$ and $3\times$ slowdowns, respectively.