A Graph Talks, But Who's Listening? Rethinking Evaluations for Graph-Language Models

Soham Petkar[1,3]* **Hari Aakash K**[1]*

Anirudh Vempati[1] **Akshit Sinha**[1,2] **Ponnurangam Kumaraguru**[1]

Chirag Agarwal[2]

[1]IIIT Hyderabad, India [2]University of Virginia, USA [3]Plaksha University, India

CLEGR Dataset GitHub

Abstract

Recent developments in Graph-Language Models (GLMs) aim to combine the structural reasoning capabilities of Graph Neural Networks (GNNs) with the semantic understanding of Large Language Models (LLMs). In this work, we show that current benchmarks for graph-language tasks are inadequate for evaluating multimodal reasoning. We perform a diagnostic study, which reveals that high accuracy can be achieved using unimodal information without necessitating graph–language integration. To address this, we introduce CLEGR (Compositional Language Graph Reasoning), a benchmark that tests multimodal reasoning via synthetic graph generation paired with questions requiring joint structural and semantic reasoning. Evaluating representative GLM architectures on CLEGR, we find that soft-prompted LLM baselines rival GLMs with full GNN backbones. Moreover, GLMs degrade significantly in tasks that emphasize structural reasoning, highlighting the need for more advanced methods to integrate graph and language inputs. Our findings highlight key limitations in current GLMs and establish CLEGR as a step towards a rigorous evaluation of explicit graph–language reasoning.

1 Introduction

The success of Vision-Language Models (VLMs) such as GPT-4V [31] and LLaVa [20] demonstrates the transformative potential of integrating different data modalities to improve complex reasoning capabilities through visual question answering, image captioning, and multimodal instruction following [5]. Inspired by this paradigm, recently there has been an increasing interest in using LLMs for graph-based applications [17]. Depending on the role of LLMs and their interaction with graph neural networks (GNNs), such techniques can be classified into treating LLMs as the final component for prediction (LLM as Predictor) [3, 9, 10, 27], treating LLMs as the feature extractor for GNNs (LLM as Encoder) [4, 8, 19], or aligning the latent space of LLMs with GNNs (LLM as Aligner) [12, 36].

Similarly, according to the relationship between graph and text presented in the application, the scenarios can be categorized into pure graphs, Text-Paired graphs (entire graphs paired with a descriptive text summary) [13], and Text-Attributed Graphs (TAGs). TAGs are prevalent in real-world applications where nodes represent textual entities like documents or papers, and edges capture relationships between them [11, 30]. The combination of textual attributes with the graph structure has

*Equal contribution. soham.petkar@plaksha.edu.in, hariaakash.k@research.iiit.ac.in.

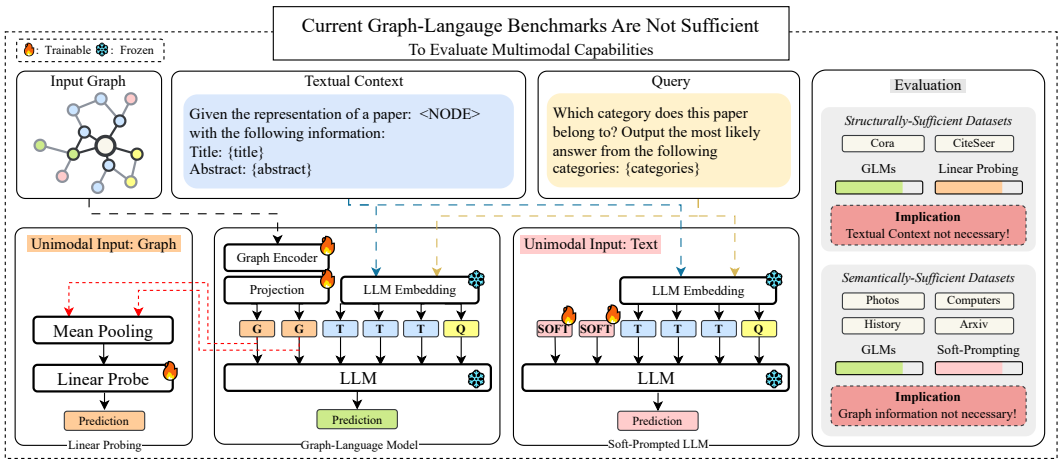

Figure 1: Current graph-language benchmarks are insufficient: Node Classification datasets can be solved with unimodal signals alone. Linear probing matches GLMs on structurally-sufficient datasets (text unnecessary), while soft-prompted LLMs match GLMs on semantically-sufficient datasets (graph unnecessary), showing that true multimodal reasoning is not evaluated.

significantly improved representation learning for various applications ranging from recommendation systems to social networks [37], with recent work further demonstrating that integrating textual semantics is critical for generalization in foundational models [2].

To effectively exploit the rich semantic information within TAGs, the LLM-as-predictor framework has emerged as a particularly promising direction [24]. Notably, LLM-as-predictor approaches have demonstrated zero-shot transfer capabilities across datasets, allowing models trained on a single dataset to generalize effectively to unseen graphs with different features and graph structure [27].

In this work, we focus primarily on LLM-as-predictor approaches, as they represent a direct application of language models to graph question-answering tasks and align with our goal of assessing graph-language multimodal capabilities. Henceforth, we refer to this approach as *Graph-Language Models (GLMs)* throughout the rest of the manuscript.

We first show that including node classification datasets in graph-language benchmarks does not provide a good proxy for tasks requiring a combination of structural and semantic knowledge. By training unimodal baselines that use either the graph structure or the text attributes associated with the nodes, we achieve performance comparable to GLMs, questioning the utility of these datasets for assessing multimodal capabilities. To accurately assess these capabilities, we further introduce a new Graph Question-Answering benchmark, CLEGR (Compositional Graph-Language Reasoning), specifically designed so that both semantic and structural understanding are necessary to accurately answer the questions. Our contributions are summarized as follows:

1. We demonstrate that current GLMs can achieve strong performance on existing benchmarks by relying solely on graph or language modalities: On *semantically-sufficient* datasets, we show that using a graph encoder as a backbone does not provide any advantage over using only text attributes for classification. On *structurally-sufficient* datasets, we use linear probing [1] to show that graph-encoder representations are sufficient to achieve performance comparable to the full GLM setup, questioning the utility of these datasets in evaluating the multimodal interplay of graphs and language.

2. We introduce CLEGR, a synthetic benchmark explicitly constructed to assess multimodal graph-language reasoning. CLEGR spans over 1,000 diverse graphs and **54,000** questions designed across multiple structural and reasoning levels. We also show that while GLMs saturate performance on CLEGR's retrieval tasks, their performance drops significantly on tasks requiring reasoning over graph structure, and is on par with soft-prompted LLM baselines.

Details of our model formulations, dataset descriptions, and experimental setup are deferred to Appendix A, Appendix B, and Appendix B.8 respectively. For additional context on notation, GLM variants, dataset construction, and training protocols, we refer the reader to these sections.

Table 1: Accuracy on Node Classification tasks reveals two dataset categories: *semantically-sufficient* datasets (Computers, Photo, History, Arxiv) where soft-prompted LLMs approximate GLM performance, indicating textual content alone suffices for classification; and *structurally-sufficient* datasets (Cora, CiteSeer) where GNNs dominate and soft-prompted LLMs fail, suggesting graph structure is critical while semantic reasoning capabilities remain underutilized in current GLM evaluations.

Model	Computers	Photo	History	Arxiv	Cora	CiteSeer
GNNs						
GAT	$93.67_{\pm 0.28}$	$96.51_{\pm 0.20}$	$82.81_{\pm 0.74}$	$73.30_{\pm 0.18}$	$86.05_{\pm 1.37}$	$71.12_{\pm 0.84}$
GCN	$\mathbf{93.94_{\pm 0.13}}$	$95.74_{\pm 0.10}$	$82.91_{\pm 0.45}$	$73.53_{\pm 0.12}$	$86.98_{\pm 0.95}$	$72.14_{\pm 0.67}$
GraphSAGE	$93.11_{\pm 0.23}$	$\mathbf{96.54_{\pm 0.15}}$	$83.24_{\pm 0.82}$	$73.00_{\pm 0.28}$	$\mathbf{87.31_{\pm 0.81}}$	$\mathbf{72.26_{\pm 0.70}}$
Graph–Language Models						
TEA-GLM Llama3-8B	$73.10_{\pm 1.07}$	$70.51_{\pm 1.33}$	$81.56_{\pm 5.39}$	$73.08_{\pm 0.00}$	$82.26_{\pm 1.23}$	$48.05_{\pm 1.28}$
TEA-GLM Phi3-3.5B	$69.88_{\pm 0.61}$	$64.81_{\pm 4.13}$	$81.63_{\pm 0.58}$	$67.38_{\pm 1.31}$	$82.49_{\pm 1.17}$	$42.08_{\pm 3.36}$
G-Token (GSAGE) Llama3-8B	$76.13_{\pm 0.58}$	$76.61_{\pm 0.92}$	$\mathbf{85.91_{\pm 0.27}}$	$75.49_{\pm 0.28}$	$86.72_{\pm 0.93}$	$53.23_{\pm 0.53}$
G-Token (GSAGE) Phi3-3.5B	$72.38_{\pm 0.90}$	$73.50_{\pm 2.55}$	$85.15_{\pm 0.31}$	$71.38_{\pm 0.19}$	$86.60_{\pm 1.12}$	$44.69_{\pm 1.22}$
Soft-Prompted LLMs						
Llama3-8B-SPT	$74.34_{\pm 0.63}$	$74.90_{\pm 0.57}$	$84.99_{\pm 0.66}$	$\mathbf{76.03_{\pm 0.45}}$	$28.69_{\pm 2.70}$	$18.21_{\pm 0.26}$
Phi3-3.5B-SPT	$69.74_{\pm 0.26}$	$70.71_{\pm 3.81}$	$84.55_{\pm 0.43}$	$72.04_{\pm 1.15}$	$29.74_{\pm 1.12}$	$18.28_{\pm 1.43}$

2 Evaluating Current Benchmarks for Graph-Language Tasks

In this section, we primarily investigate the following questions: **RQ1:** Are node classification datasets a sufficient test for graph-language multimodality? **RQ2:** Do both GNN and LLM components of GLMs contribute to their strong performance on these datasets?

2.1 Experimental Setup

We evaluate all models on six widely-used TAG datasets: Cora [23], CiteSeer [32], Computers (Amazon Computers), Photo (Amazon Photos), History [25], and Arxiv [11], spanning diverse domains and graph structures. These datasets are specifically selected because they form the backbone of current graph-language evaluation practices, appearing across multiple prominent benchmarks including GLBench [18], GraphFM [28], TAG [29], and Planetoid [32].

Graph-Language Models. We evaluate two prominent GLM architectures: (1) TEA-GLM [27], which performs zero-shot graph learning by encoding graph structure through textual descriptions and leveraging LLMs for reasoning, and (2) GraphToken [24], which learns discrete graph tokens to represent structural information and integrates them with language model processing. Both GLMs are equipped with GraphSAGE as the backbone, and each GLM is paired with both Llama3-8B and Phi3-3.5B backbones to assess consistency across different language model scales.

Graph-Only Baselines. To isolate the contribution of structural information, we employ strong traditional GNNs that rely exclusively on graph topology: (1) GAT [26], which uses attention mechanisms to weigh the importance of neighboring nodes, (2) GCN [14], which performs localized first-order approximations of spectral convolutions, and (3) GraphSAGE [6], which generates node embeddings by sampling and aggregating features from node neighborhoods.

Language Only Baselines. To isolate the contribution of textual information, we use soft-prompted LLMs [16]. These models use identical Llama3-8B and Phi3-3.5B backbones but operate only on the node text, augmented with trainable prompt vectors.

We employ identical training procedures across all models and report results across five random seeds. Additional implementation details are deferred to Appendix Sections B.6, B.7, B.9.

2.2 Analysis of Modality Contribution

Informed by the performance of unimodal (graph-only vs language-only) baselines, we categorize the datasets into two groups: *(a) semantically-sufficient* and *(b) structurally-sufficient*.

First, on the Computer, Photo, History, and Arxiv datasets, which we term *semantically-sufficient*, we observe in Table 1 that soft-prompted LLMs achieve results that are highly competitive with

GLMs across all datasets. GNNs outperform both GLMs and soft-prompted LLMs significantly on Computers and Photos, but are equally as good on History and Arxiv. These results suggest that, for these datasets, the semantic content present in the textual node attributes is sufficient to achieve performance equivalent to a multimodal model (incorporating the graph structure with the semantic content), making a graph encoder not strictly necessary to achieve high performance.

Our results on Cora and CiteSeer follow a different trend. In these datasets, we find that structural information alone can saturate performance. We term these datasets to be *structurally-sufficient*. Simply providing the textual attributes related to a node is not enough to classify it accurately, as is shown by the low accuracies achieved by soft-prompted LLMs in Table 1. These results suggest that the addition of textual attributes does not provide any extra gains in accuracy on top of what is already learned by the graph structure. Since LLMs alone cannot achieve strong performance in these datasets, we hypothesize that the large accuracy gains that GLMs achieve over LLMs can be attributed to their graph encoder. To test this hypothesis, we perform a probing analysis.

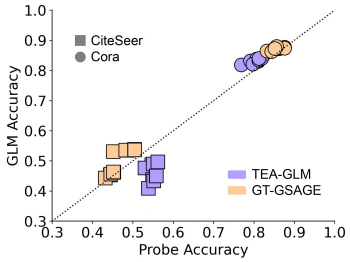

Figure 2: Linear probe accuracy closely matches full GLM performance on structurally-sufficient datasets, with Pearson correlation $r = 0.9643$, showing that graph encoders capture all task-relevant information while LLMs act merely as expensive decoder heads.

First, we take a fully trained and frozen GLM and pass the graph data through its graph encoder to extract the final node representations (*i.e.,* the graph tokens). Next, we pass these graph tokens through the projection module. A simple linear classifier is then trained on top of these frozen representations to perform the node classification task (See Equation 8 in Appendix B.5).

Our probing results (shown in Figure 2) on Cora and CiteSeer using TEA-GLM and G-Token (GSAGE) reveal that a simple linear classifier applied directly to the projector outputs achieves accuracy that nearly matches the full GLM performance. This finding provides strong evidence that on *structurally-sufficient* datasets, the graph encoder captures all the critical information required for the task, suggesting that the LLM is functionally similar to a very large decoder network. Therefore, the textual-semantic reasoning capabilities of LLMs remain underutilized for these datasets.

Takeaway #1

Our analysis highlights a significant gap in existing benchmarks: a lack of datasets that are neither purely *semantically-sufficient* nor *structurally-sufficient*, but instead require the integration of both modalities to test true graph-language multimodality.

3 CLEGR: Compositional Language-Graph Reasoning

Having identified limitations in the current evaluation setup for graph-language models, we now introduce a framework that is neither *semantically-sufficient* nor *structurally-sufficient*. To accurately assess multimodal capabilities, we construct **CLEGR**, a benchmark suite to enable reasoning over graph structures with explicit node and edge textual attributes.

3.1 Overview of CLEGR

CLEGR's design is guided by three core principles to preclude unimodal solutions: i) *Structural Dependency*: Tasks require multi-hop reasoning over graph topology that cannot be inferred from the node text alone, making vanilla language models insufficient as they lack access to structural relationships beyond immediate textual context [34]; ii) *Semantic grounding:* Questions that require natural language understanding capabilities, making traditional GNNs inadequate as they operate on numerical representations without semantic comprehension [7, 35]; and iii) *Compositional complexity:* Tasks combine multiple reasoning steps that blend property lookup with logical inference, creating challenges that benefit from integrating both structural and semantic information throughout the reasoning [33].

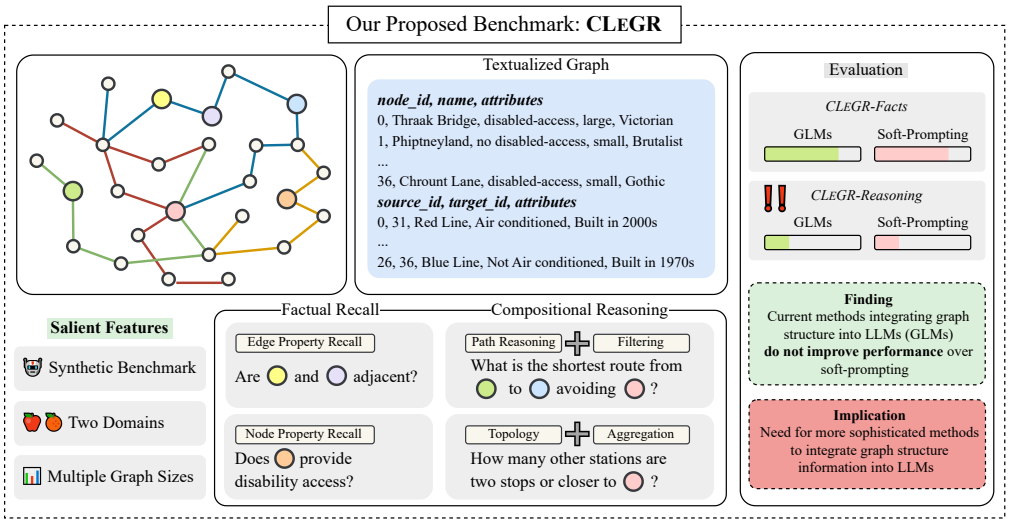

Figure 3: Illustration of our evaluation framework and the CLEGR benchmark. CLEGR addresses limitations of existing benchmarks using synthetic graphs with explicit node and edge attributes, covering tasks from factual recall (CLEGR-Facts) to compositional reasoning (CLEGR-Reasoning) across filtering, aggregation, path reasoning, and topology. Evaluation shows that GLMs match baselines on fact-based tasks but fail to outperform soft-prompted LLMs on reasoning, indicating insufficient graph-language integration.

CLEGR is constructed as an extension of the foundational CLEVR-Graph dataset [22], which introduced synthetic graph question answering tasks using fictional subway system graphs. The synthetic design of stations, lines, and connections helps to eliminate pre-training confounds, ensuring language models cannot rely on memorized knowledge.

3.2 CLEGR Design and Structure

The CLEGR benchmark (see Fig. 3) contains 1000 synthetic subway graphs split into two subsets: CLEGR-Facts (500 graphs) and CLEGR-Reasoning (500 graphs), each using a 3:1:1 train/validation/test split.

CLEGR-Facts is a purely retrieval-based benchmark, requiring node/edge property lookup without graph structure reasoning (*What music is played on Station X?*). This subset generates exactly 22,000 questions (44 per graph from 22 templates) and serves as a baseline to assess LLM performance disparity when reasoning over graph structure is not required.

CLEGR-Reasoning extends beyond current TAG benchmarks by adding compositional reasoning over graph structure. It contains 32,248 questions generated from 34 templates, with natural filtering removing structurally invalid instances. Questions demand multi-step inference and graph traversal (e.g., *What is the shortest path between Station A and B using only air-conditioned lines?*). CLEGR-Reasoning systematically covers four reasoning types (Filtering, Aggregation, Path Reasoning, Topology) across three scopes (node, edge, subgraph-level). The compositional design combines multiple reasoning types to create complex multi-step problems that probe different facets of graph-language reasoning capabilities. Detailed structural statistics, template examples, and dataset construction process are present in Appendix C and Appendix D.

4 Evaluating GLMs on CLEGR

Equipped with a benchmark that necessitates the incorporation of graph structure and text semantics, we now evaluate GLMs on CLEGR. Specifically, we answer the following question: **RQ3:** Does incorporating structural information into LLMs provide performance gains over soft-prompting LLMs on tasks requiring multimodal reasoning?

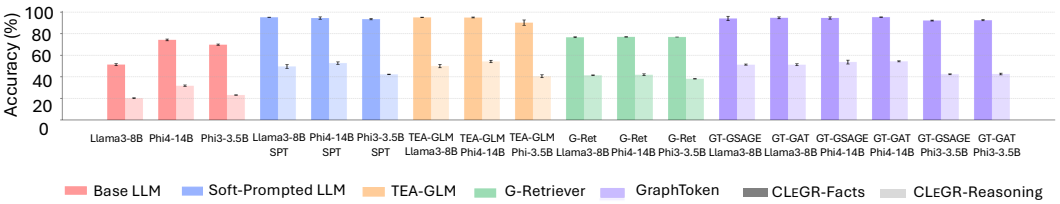

Figure 4: CLEGR Results. Accuracy (%) (y-axis) of models (GLMs and baselines) (x-axis) on CLEGR. Results are shown separately for CLEGR-Facts and CLEGR-Reasoning, demonstrating that while GLMs saturate fact-based tasks, they fail to outperform soft-prompted baselines on reasoning tasks requiring structural understanding.

4.1 Experimental Setup

We evaluate several model categories to comprehensively assess graph reasoning capabilities. First, we include the GLMs introduced in Section 2.1. Second, we add GLMs with retrieval-augmented approaches, specifically G-Retriever [9], which enhances task performance through subgraph and textual retrieval to extract relevant nodes and edges for answering questions. Following G-Retriever's setup, we adapt all GLMs to our tasks by including graphs in textualized format, ensuring consistent input prompts across models. To evaluate GLM performance across different backbones, we also evaluate GraphToken with a GAT backbone. We also include LLM-only baselines, adding Phi4-14B to our initial set of LLMs from Section 3.1 to analyze scaling effects.

Performance is evaluated using overall accuracy across all question types. A prediction is considered correct only when it precisely matches the ground truth. We provide detailed performance breakdowns across our two major subsets (Facts vs. Reasoning). Exact training and evaluation details are deferred to Appendix B.6 and Appendix E.

4.2 Results

Our results, presented in Figure 4, show that despite using diverse architectural strategies for encoding structural information into LLMs, GraphToken and TEA-GLM provide negligible performance gains, if any, over purely language-based soft-prompted baselines. Surprisingly, G-Retriever, which uses a sophisticated subgraph retrieval mechanism intended to selectively provide relevant graph substructures to the model, suffers from a degradation rather than an improvement in performance. We hypothesize that G-Retriever's performance degradation stems from potentially retrieving incorrect subgraphs, and not having sufficient context to answer questions correctly. These findings collectively indicate that current GLMs do not utilize multimodal inputs effectively; instead, they revert to powerful but ultimately unimodal textual processing, failing to integrate the structural data in a meaningful way. Additional experiments and extended results that further support these observations are provided in Appendix G and Appendix H.

Takeaway #2

GLMs fail to leverage their structural encoders for graph reasoning tasks and do not offer significant advantages over soft-prompted baselines. Our findings suggest GLMs' capabilites are primarily driven by the LLM's textual processing capabilities rather than an interplay of graph and text modalities.

5 Discussion

In this work, we first demonstrate that on current graph-language benchmarks that incorporate Node Classification datasets to evaluate multimodality, performance equivalent to GLMs can be achieved by using unimodal baselines, highlighting the need for new evaluation paradigms. To fill this gap, we introduce CLEGR, a synthetic graph-language reasoning benchmark explicitly designed to assess multimodal integration. Evaluations on CLEGR highlight limitations of current GLMs, showcasing that they provide negligible gains over soft-prompting LLMs, emphasizing the need for more sophisticated methods to integrate graph information into LLMs. While CLEGR introduces

a framework to evaluate multimodal capabilities, several limitations and avenues for future work remain. As GLMs continue to develop, their capacity to represent graphs are likely to increase, and our current evaluation may not encompass all their capabilities. We aim to continually refine and update CLEGR to more effectively assess the graph-language understanding and reasoning abilities of emerging GLMs.

6 Acknowledgements

We would like to thank the PreCog Lab at IIIT-Hyderabad for their valuable guidance and discussions throughout this work. In particular, we thank Sumit Kumar, Anish R. Joishy, Shiven Sinha, Vedanta S. P., Sriharini Margapuri, and Shashwat Singh for their detailed feedback and thoughtful suggestions during multiple stages of the project.

We are grateful to IIIT-Hyderabad for funding Soham Petkar's six-month research internship, and to Plaksha University for their continued support of his research endeavors. We also thank the anonymous reviewers for their insightful feedback.

C.A. is supported, in part, by grants from Capital One, the LaCross Institute for Ethical AI in Business, the UVA Environmental Institute, the OpenAI Researcher Program, and Cohere.

The views expressed are those of the authors and do not necessarily reflect the official policies or positions of the supporting organizations.

References

[1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2018. URL https://arxiv.org/abs/1610.01644.

[2] Arvindh Arun, Sumit Kumar, Mojtaba Nayyeri, Bo Xiong, Ponnurangam Kumaraguru, Antonio Vergari, and Steffen Staab. Semma: A semantic aware knowledge graph foundation model, 2025. URL https://arxiv.org/abs/2505.20422.

[3] Runjin Chen, Tong Zhao, Ajay Jaiswal, Neil Shah, and Zhangyang Wang. Llaga: Large language and graph assistant, 2024. URL https://arxiv.org/abs/2402.08170.

[4] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S Dhillon. Node feature extraction by self-supervised multi-scale neighborhood prediction, 2022. URL https://arxiv.org/abs/2111.00064.

[5] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning, 2023. URL https://arxiv.org/abs/2305.06500.

[6] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018. URL https://arxiv.org/abs/1706.02216.

[7] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications, 2018. URL https://arxiv.org/abs/1709.05584.

[8] Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning, 2024. URL https://arxiv.org/abs/2305.19523.

[9] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering, 2024. URL https://arxiv.org/abs/2402.07630.

[10] Zhongmou He, Jing Zhu, Shengyi Qian, Joyce Chai, and Danai Koutra. Linkgpt: Teaching large language models to predict missing links, 2024. URL https://arxiv.org/abs/2406.04640.

[11] 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, 2021. URL https://arxiv.org/abs/2005.00687.

[12] Bowen Jin, Wentao Zhang, Yu Zhang, Yu Meng, Xinyang Zhang, Qi Zhu, and Jiawei Han. Patton: Language model pretraining on text-rich networks, 2023. URL https://arxiv.org/abs/2305.12268.

[13] Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. Large language models on graphs: A comprehensive survey, 2024. URL https://arxiv.org/abs/2312.02783.

[14] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. URL https://arxiv.org/abs/1609.02907.

[15] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited, 2019. URL https://arxiv.org/abs/1905.00414.

[16] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021. URL https://arxiv.org/abs/2104.08691.

[17] Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. A survey of graph meets large language model: Progress and future directions, 2024. URL https://arxiv.org/abs/2311.12399.

[18] Yuhan Li, Peisong Wang, Xiao Zhu, Aochuan Chen, Haiyun Jiang, Deng Cai, Victor Wai Kin Chan, and Jia Li. Glbench: A comprehensive benchmark for graph with large language models, 2024. URL https://arxiv.org/abs/2407.07457.

[19] Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. One for all: Towards training one graph model for all classification tasks, 2024. URL https://arxiv.org/abs/2310.00149.

[20] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023. URL https://arxiv.org/abs/2304.08485.

[21] Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Classic gnns are strong baselines: Reassessing gnns for node classification, 2024. URL https://arxiv.org/abs/2406.08993.

[22] D. Mack and A. Jefferson. Clevr graph: A dataset for graph question answering. https://github.com/Octavian-ai/clevr-graph, 2018.

[23] Andrew Kachites McCallum et al. Automating the construction of internet portals with machine learning. In *AAAI Spring Symposium on Mining Answers from Textual Data*, 2000.

[24] 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.

[25] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation, 2019. URL https://arxiv.org/abs/1811.05868.

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. URL https://arxiv.org/abs/1710.10903.

[27] Duo Wang, Yuan Zuo, Fengzhi Li, and Junjie Wu. Llms as zero-shot graph learners: Alignment of gnn representations with llm token embeddings, 2024. URL https://arxiv.org/abs/2408.14512.

[28] Yuhao Xu, Xinqi Liu, Keyu Duan, Yi Fang, Yu-Neng Chuang, Daochen Zha, and Qiaoyu Tan. Graphfm: A comprehensive benchmark for graph foundation model, 2024. URL https://arxiv.org/abs/2406.08310.

[29] Hao Yan, Chaozhuo Li, Ruosong Long, Chao Yan, Jianan Zhao, Wenwen Zhuang, Jun Yin, Peiyan Zhang, Weihao Han, Hao Sun, Weiwei Deng, Qi Zhang, Lichao Sun, Xing Xie, and Senzhang Wang. A comprehensive study on text-attributed graphs: Benchmarking and rethinking. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=m2mbfoSuJ1.

[30] Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhuo Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. Graphformers: Gnn-nested transformers for representation learning on textual graph, 2023. URL https://arxiv.org/abs/2105.02605.

[31] Zhengyuan Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. The dawn of lmms: Preliminary explorations with gpt-4v(ision), 2023. URL https://arxiv.org/abs/2309.17421.

[32] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings, 2016. URL https://arxiv.org/abs/1603.08861.

[33] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

[34] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. QA-GNN: Reasoning with language models and knowledge graphs for question answering. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.45. URL https://aclanthology.org/2021.naacl-main.45/.

[35] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks, 2019. URL https://arxiv.org/abs/1903.03894.

[36] Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. Learning on large-scale text-attributed graphs via variational inference, 2023. URL https://arxiv.org/abs/2210.14709.

[37] Jason Zhu, Yanling Cui, Yuming Liu, Hao Sun, Xue Li, Markus Pelger, Tianqi Yang, Liangjie Zhang, Ruofei Zhang, and Huasha Zhao. Textgnn: Improving text encoder via graph neural network in sponsored search. In *Proceedings of the Web Conference 2021*, WWW '21, page 2848–2857. ACM, April 2021. doi: 10.1145/3442381.3449842. URL http://dx.doi.org/10.1145/3442381.3449842.

A Preliminaries

A.1 Graph-Language Models

Formally, a graph-language model GLM $= (M_l, M_g, M_P)$ comprises three main components: M_l is an LLM, M_g is the graph encoder, and M_P is the linear projector that aligns graph and text representations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, F^{\mathcal{V}}, F^{\mathcal{E}})$ be a Text Attributed Graph (TAG), where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges between nodes, $N = |\mathcal{V}|$ is the number of nodes in the graph, $M = |\mathcal{E}|$ is the number of edges, $F^{\mathcal{V}} = \{f_1^{\mathcal{V}}, f_2^{\mathcal{V}}, \ldots, f_N^{\mathcal{V}}\}$ is the set of textual features of each node, and $F^{\mathcal{E}} = \{f_1^{\mathcal{E}}, f_2^{\mathcal{E}}, \ldots, f_M^{\mathcal{E}}\}$ is the set of textual features of each edge.

For node-level question-answering tasks, given a graph $\mathcal{G}$, a node $n_i \in \mathcal{V}$, its textual features $f_i \in F$, and a textual question q, the prediction made by the GLM is:

$$\begin{aligned} \hat{y} &= \text{Predict}_{\text{GLM}}(\mathcal{G}, n_i, f_i, q) \\ &= M_l\left(M_P(M_g(\mathcal{G}, n_i)) \| W(f_i) \| W(q), \right) \end{aligned} \tag{1}$$

where $W \in \mathbb{R}^{|L| \times d}$ is the word embedding matrix of the LLM (L denotes the token vocabulary and d the LLM's hidden dimension), and $\|$ denotes concatenation of token sequences. Here, $\hat{y}, q, f_i \in L^*$ where L^* represents sequences of tokens from the vocabulary. The ground truth g also belongs to L^*.

For graph-level question-answering tasks, given a graph $\mathcal{G}$ and a textual question q, the GLM makes a prediction as:

$$\begin{aligned} \hat{y} &= \text{Predict}_{\text{GLM}}(\mathcal{G}, q) \\ &= M_l \left(M_P(\text{Pool}(M_g(\mathcal{G}))) \| W(f) \| W(q) \right) \end{aligned} \tag{2}$$

where $\text{Pool}(\cdot)$ aggregates node embeddings from $M_g(\mathcal{G}, 0), \ldots, M_g(\mathcal{G}, N-1)$ to produce a single graph-level representation, and f represents the concatenated textual features of the entire graph.

A.2 Soft Prompting

To isolate the contribution of graph encoders in GLMs, we use soft prompting as a baseline. Soft Prompting [16] introduces learnable prompt tokens which are concatenated with the embeddings of the input given to the LLM. We note here that a GLM can be viewed as a soft-prompt where a graph encoder is learnt instead of token embeddings. A soft-prompted LLM consists of components $(M_l, \mathbf{s})$, where M_l is the same LLM used in GLMs and $\mathbf{s} \in \mathbb{R}^d$ is the trainable soft-prompt vector. For a given task, the soft-prompted LLM prediction is:

$$\hat{y}_{\text{soft}} = M_l \left(\mathbf{s} \| W(f_i) \| W(q) \right) \tag{3}$$

The soft-prompt vector $\mathbf{s}$ is trained using the same objective and training procedure as the GLM, effectively learning to encode task-relevant information without access to graph structure. This baseline allows us to determine whether a sophisticated graph encoder is required to achieve performance gains or it can be achieved through simple parameter optimization in the language model space.

B Models

B.1 TEA-GLM

The TEA-GLM (Token Embedding-Aligned Graph Language Model) [27] methodology is a novel framework designed to enhance zero-shot graph machine learning by integrating Graph Neural Networks (GNNs) with instruction-fine-tuned Large Language Models (LLMs). It consists of two main stages: first, pretraining a GNN using enhanced self-supervised learning with feature-wise contrastive learning to align its node representations with LLM token embeddings, enabling the GNN to leverage the LLM's pretrained knowledge; second, training a linear projector to transform these GNN representations into a fixed number of graph token embeddings, which are incorporated into a unified instruction for various graph tasks, without tuning the LLM.

B.2 G-Retriever

G-Retriever [9] is a framework for question answering on textual graphs, integrating Retrieval-Augmented Generation (RAG) with Graph Neural Networks (GNNs) and Large Language Models (LLMs) to enable users to "chat with their graph." It addresses complex queries on real-world textual graphs by first developing a Graph Question Answering (GraphQA) benchmark with diverse datasets like ExplaGraphs, Scenegraphs, and WebQSP. G-Retriever employs a novel RAG approach, formulating subgraph retrieval as a Prize-Collecting Steiner Tree optimization problem to efficiently select relevant graph parts, mitigating scalability issues and LLM hallucination. The retrieved subgraph is textualized and combined with the query, then processed by a frozen LLM with soft prompting for fine-tuned, contextually accurate responses across applications like scene graph understanding, common sense reasoning, and knowledge graph reasoning.

B.3 Graph-Token (G-Token)

In the Graph-Token methodology [24] G-Token (GSAGE) employs GraphSAGE as the GNN encoder to process the graph's structure, generating representations through neighborhood aggregation, while

G-Token (GAT) uses GAT (Graph Attention Network) as the GNN encoder, leveraging attention mechanisms to weigh node connections. The representations are then mapped by a trained linear projector into token embeddings. These tokens are prepended to the prompt of a frozen LLM.

B.4 GNN Architectures

Let a graph be $G = (\mathcal{V}, \mathcal{E}, X, Y)$, where $\mathcal{V}$ denotes the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents the set of edges, $X \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the node feature matrix, with $|\mathcal{V}|$ representing the number of nodes and d the dimension of the node features, and $Y \in \mathbb{R}^{|\mathcal{V}| \times C}$ is the one-hot encoded label matrix, with C being the number of classes. Let $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ denote the adjacency matrix of G.

B.4.1 Graph Convolutional Networks (GCN)

GCNs [14] update node embeddings using a normalized sum over neighboring features:

$$h_v^{(l)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\hat{d}_v \hat{d}_u}} h_u^{(l-1)} W^{(l)}\right), \tag{4}$$

where $\hat{d}_v$ is the degree of node v (including self-loops), $W^{(l)}$ is the trainable weight matrix at layer l, and $\sigma(\cdot)$ is an activation function such as ReLU.

B.4.2 GraphSAGE

GraphSAGE [6] aggregates neighborhood information using a fixed aggregation function (e.g., mean):

$$h_v^{(l)} = \sigma\left(h_v^{(l-1)} W_1^{(l)} + \left(\text{mean}_{u \in \mathcal{N}(v)} h_u^{(l-1)}\right) W_2^{(l)}\right), \tag{5}$$

where $W_1^{(l)}$ and $W_2^{(l)}$ are trainable matrices and the neighbor embeddings are averaged.

B.4.3 Graph Attention Networks (GAT)

GATs [26] apply masked self-attention over neighbors. The attention coefficient between nodes v and u is:

$$\frac{\exp\left(\text{LeakyReLU}\left(a^\top [W h_v^{(l-1)} \parallel W h_u^{(l-1)}]\right)\right)}{\sum_{r \in \mathcal{N}(v)} \exp\left(\text{LeakyReLU}\left(a^\top [W h_v^{(l-1)} \parallel W h_r^{(l-1)}]\right)\right)}, \tag{6}$$

and the node update is:

$$h_v^{(l)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(l)} W h_u^{(l-1)}\right), \tag{7}$$

where W is the shared weight matrix, a is a learnable attention vector, and $\parallel$ denotes concatenation and $\alpha_{vu}^{(l)}$ is the attention coefficient between nodes v and u.

B.5 Linear Probing

Mathematically, the prediction made by the probe can be represented as:

$$\hat{y} = \arg\max_i \left(W \cdot \text{flatten}\left(M_P\left(M_g(\mathcal{G})\right)\right) + b\right)_i \tag{8}$$

where $W \in \mathbb{R}^{c \times d'}$ and $b \in \mathbb{R}^c$ denote the classifier weights and bias, c is the number of output classes, and d' is the dimensionality of the flattened embedding representation. This minimal setup isolates the graph encoder and projection module to quantify linearly separable task-specific information without the influence of the LLM (M_l).

B.6 Training Details for GLMs and Soft-Prompt Baselines

All experiments are conducted using five random seeds: `0`, `42`, `1918`, `2004`, and `2024`. We use a fixed number of `10` graph tokens (projector output tokens) across all GLMs and soft-prompt baselines. The input format to the LLM follows the structure:

```
[<BOS> + 10x<G> + Context + Question + <EOS>]
```

where `<G>` represents either the trainable graph tokens (in GLMs) or the fixed soft-prompt tokens (in soft-prompt baselines).

For both the CLEGR and node classification tasks, all Graph-Language Models (GLMs) and soft-prompt variants are trained for a single epoch using the AdamW optimizer with a constant learning rate of `0.001` and a batch size of `1` during both training and evaluation.

All GLMs are implemented using a GraphSAGE [6] backbone, with the exception of G-Token (GAT), which uses a GAT-based encoder. Furthermore, the TEA-GLM models are pre-trained using 1000 PCA-projected features extracted from each LLM, following the protocol described in [27]. (Table 3 & Table 4)

B.7 GNN Training

We follow the works of [21] for GNN baselines; we follow the same training setup, adopting their recommended hyperparameters and optimization configurations for each dataset. (Table 2)

B.8 Node Classification Evaluation

For the **GNN baselines**, we adopt standard evaluation pipelines as used in prior work, including accuracy-based evaluation over ground-truth node labels. Specifically, following [21], models are trained and evaluated on fixed data splits, and performance is reported as the mean and standard deviation across five random seeds.

For the **Graph-Language Models (GLMs)** and **soft-prompt models**, we adopt a flexible string-matching protocol to map generated textual responses to discrete class labels. Each dataset-specific evaluator contains a fixed list of candidate class names and defines a 'match_prediction' function that attempts to align the raw LLM output with one of the ground-truth labels. A prediction is considered correct if it begins with the correct class name (e.g., `"Asia"` matches `"Asia in the 20th century"`). Unmatched predictions are mapped to a special `None` class index.

This string-based matching enables compatibility between natural language generation from LLMs and traditional classification metrics like accuracy. Final accuracy scores are computed by comparing matched predictions to the ground-truth class label.

B.9 Computing Infrastructure

All experiments are conducted on machines equipped with NVIDIA A100 GPUs with 80GB of memory.

C CLEGR Benchmark Construction

C.1 CLEGR

C.1.1 Overview

The CLEGR dataset is a graph-based question answering benchmark built on synthetic subway networks extending on the work by [22]. Each graph represents a fictional subway system with randomly generated nodes and attributes. Questions are categorized into two types: *Fact-Based* and *Reasoning*.

Table 2: Hyperparameter configurations for GCN, GAT, and GraphSAGE across five benchmark datasets. 'Norm' abbreviates normalization (LN: LayerNorm, BN: BatchNorm), and 'LR' is the learning rate.

Model	Dataset	ResNet	Norm	Dropout	#Layers	Hidden Dim	LR	Epochs
GCN	Cora	False	None	0.7	3	512	0.001	500
	Citeseer	False	None	0.5	2	512	0.001	500
	Computer	False	LN	0.5	3	512	0.001	1000
	Photo	True	LN	0.5	6	256	0.001	1000
	History	True	LN	0.5	6	256	0.001	1000
	Arxiv	True	BN	0.5	5	512	0.0005	2000
GAT	Cora	True	None	0.2	3	512	0.001	500
	Citeseer	True	None	0.5	3	256	0.001	500
	Computer	False	LN	0.5	2	64	0.001	1000
	Photo	True	LN	0.5	3	64	0.001	1000
	History	True	LN	0.5	3	64	0.001	1000
	Arxiv	True	BN	0.5	5	256	0.0005	2000
GraphSAGE	Cora	False	None	0.7	3	256	0.001	500
	Citeseer	False	None	0.2	3	512	0.001	500
	Computer	False	LN	0.3	4	64	0.001	1000
	Photo	True	LN	0.2	6	64	0.001	1000
	History	True	LN	0.2	6	64	0.001	1000
	Arxiv	True	BN	0.5	4	256	0.0005	2000

Table 3: GLM hyperparameter settings for using the G-Retriever & G-Token (GSAGE) models. All configurations use consistent hidden/project dimensions and dropout.

Dataset	GNN backbone	Task	In Dim	Hidden Dim	Out Dim	Proj Dim	Layers	Dropout
Cora	GraphSAGE	Node	500	1024	1024	1024	3	0.5
Citeseer	GraphSAGE	Node	500	1024	1024	1024	3	0.5
Arxiv	GraphSAGE	Node	128	1024	1024	1024	3	0.5
Computers	GraphSAGE	Node	768	1024	1024	1024	3	0.5
History	GraphSAGE	Node	768	1024	1024	1024	3	0.5
Photo	GraphSAGE	Node	768	1024	1024	1024	3	0.5
CLEGR-Facts	GraphSAGE	Graph	768	1024	1024	1024	3	0.5
CLEGR-Reasoning	GraphSAGE	Graph	768	1024	1024	1024	3	0.5

C.1.2 Graph Generation

Graphs in CLEGR are primarily constructed using *lines*. Each graph begins with a predefined number of lines, where each line is assigned a unique name and the following attributes:

- `has_aircon`
- `color`
- `stroke`
- `built`

Lines may intersect with each other. Along each line, a sequence of *stations* is generated. Stations near line intersections may belong to two lines. Each station is assigned a unique name and the following attributes:

- `disabled_access`
- `has_rail`
- `music`

Table 4: GLM hyperparameter settings for using the GAT models. All configurations use consistent hidden/project dimensions and dropout.

Dataset	GNN backbone	Task	In Dim	Hidden Dim	Out Dim	Proj Dim	Layers	Dropout
Cora	GAT	Node	500	1024	1024	1024	3	0.5
Citeseer	GAT	Node	500	1024	1024	1024	3	0.5
Arxiv	GAT	Node	128	1024	1024	1024	3	0.5
Computers	GAT	Node	768	1024	1024	1024	3	0.5
History	GAT	Node	768	1024	1024	1024	3	0.5
Photo	GAT	Node	768	1024	1024	1024	3	0.5
CLEGR-Facts	GAT	Graph	768	1024	1024	1024	3	0.5
CLEGR-Reasoning	GAT	Graph	768	1024	1024	1024	3	0.5

- `architecture`
- `size`
- `cleanliness`

Edges connect every pair of adjacent stations on a line, and inherit properties from the parent line. The overall size of the graph is controlled by the number of lines and the number of stations per line.

C.2 Graph Statistics

Table 5 mentions the graph statistics for CLEGR Subway Networks.

Table 5: Graph statistics for CLEGR Subway Networks with different graph sizes.

Metric	CLEGR	CLEGR-Large
Average Number of Nodes	26.54 ± 5.41	73.17 ± 13.55
Average Number of Edges	28.32 ± 6.37	78.87 ± 16.41
Average Number of Lines	6.02 ± 1.23	9.97 ± 2.02

C.3 Dataset Generation Pipeline

C.3.1 Questions

The dataset contains 22 question templates for Fact Based questions and 34 question templates for Reasoning Based questions. Each generated graph will have two instances of each question template. The question templates are of the following form:

"How many stations playing {} does {} pass through?"

The empty {} in the above template will be filled by a randomly picked Music and Line.

C.3.2 Fact Based Questions

Contains 22 questions which require retrieving information about nodes, edges or the graph itself. (All the templates can be viewed in Table 6)

C.3.3 Reasoning Based Questions

Contains 34 questions which require the model to do some amount of reasoning on the information it retrieves from the graph. The different types of reasoning subgroups we try to incorporate in our dataset are listed here: **Aggregation**, **Filtering**, **PathReasoning**, **Topology**. (All the templates as per the question scope can be viewed at Table 7, Table 8, and Table 9.)

D Dataset Generation and Schema

The dataset consists of procedurally generated transit (metro system) graphs, each accompanied by a set of questions and answers derived from its structure and attributes.

D.1 Graph Generation Pipeline

Each graph, representing a unique transit map, is generated through a multi-stage pipeline designed to create complex and semi-realistic structures. The core generation logic is implemented in `generate_graph.py`.

1. **Line Generation:** A set of metro lines is created. Each line is assigned a unique ID, a name (e.g., "Blue Line", "Circle Express"), and a set of properties such as color, stroke style (solid, dashed, dotted), year of construction, and whether it has air conditioning. To ensure visual distinctiveness, combinations of color and stroke style are unique across the graph.
2. **Station Placement along Curves:** For each line, a cubic Bézier curve is generated with random control points within a predefined map radius. A specified number of initial station locations are then calculated by evaluating points along this curve. This method produces smooth, winding paths for metro lines rather than simple straight lines. A small amount of Gaussian noise is added to each station's coordinates for organic variation. Each station is initialized with a unique, programmatically generated name and a set of properties (e.g., architecture, cleanliness, disabled access).
3. **Station Coalescing:** A critical step to create realistic interchanges. A KD-Tree is used to efficiently find all stations across all lines that are within a minimum distance threshold of each other. These nearby stations are merged into a single node using a Disjoint Set Union (DSU) algorithm. The resulting merged node, representing an interchange station, inherits the ID and properties of one of its constituent pre-merge stations. This process transforms a simple collection of lines into a more complex, interconnected network.
4. **Edge Generation:** After stations are coalesced, edges are created to connect consecutive stations along each line's path. If a sequence of stations on a line was A $\rightarrow$ B $\rightarrow$ C and stations B and C were coalesced into a new station D, the resulting edges would connect A $\rightarrow$ D. Edges store the IDs of the two stations they connect and inherit properties from their parent line, such as its name, color, and stroke style.
5. **Connectivity Assurance:** The graph is checked for connectivity using NetworkX. If the graph consists of multiple disconnected components, new "connector" edges are added to link them. A random node is chosen from each of two components, and a new edge is created between them. This edge is styled as a dotted line to be visually distinct and is assigned to one of the existing lines. This ensures the entire graph is a single connected component, which is a prerequisite for many graph algorithms (e.g., shortest path calculations between any two nodes).
6. **Integer Naming (Optional):** For certain model training regimes, all human-readable station and line names can be replaced with unique integer strings. This prevents models from learning spurious correlations from the names themselves and forces them to rely solely on the graph's topology and categorical features. When this option is enabled, an entity's ID is also updated to match its new integer name for consistency.

The entire generation process is configurable via command-line arguments, allowing for the creation of graphs of varying sizes (small, medium, large, or a random mix), controlled by parameters like the number of lines, stations per line, and the map radius.

D.2 Feature Schema

The dataset contains three primary entities: Nodes (Stations), Edges (Tracks), and Lines. Their attributes are detailed below.

D.2.1 Node Features (Stations)

Each node represents a station and has the following attributes, which are one-hot encoded to form the node feature tensor `x`.

`id` (String): A unique identifier for the station. If integer names are used, this is the integer as a string.

`name` (String): The human-readable or integer name of the station.

`x, y` (Float): The 2D coordinates of the station on the map.

`disabled_access` (Boolean): Whether the station has disabled access.

`has_rail` (Boolean): A categorical property, e.g., for distinguishing train stations from bus stations in a mixed-modal system.

`music` (String): The genre of ambient music played (e.g., `'classical'`, `'rock'`, `'none'`).

`architecture` (String): The architectural style of the station (e.g., `'victorian'`, `'modernist'`).

`size` (String): The relative size of the station (e.g., `'small'`, `'large'`).

`cleanliness` (String): A binary property (`'clean'` or `'dirty'`).

D.2.2 Edge Features (Tracks)

Each edge represents a track segment between two stations on a specific line. Edges are directed in the PyG representation (i.e., an edge from A to B is distinct from B to A), but represent an undirected physical connection. Their attributes form the edge feature tensor `edge_attr`.

`station1, station2` (String): The IDs of the nodes connected by the edge.

`line_id` (String): The ID of the line this track segment belongs to.

`line_name` (String): The name of the line.

`line_color` (String): The color of the line.

`line_stroke` (String): The stroke style of the line (e.g., `'solid'`, `'dotted'`).

`properties` (Dict): A dictionary containing properties inherited from the line, such as `'line_has_aircon'` (Boolean) and `'line_built'` (String, e.g., '1990').

D.3 Question-Answer Generation

With a graph fully generated, the `generate.py` script produces a set of question-answer pairs.

1. **Question Templates:** A predefined set of `QuestionForm` objects encapsulate different types of questions. These are organized by group (e.g., lookup, comparison) and type (e.g., existence, counting).
2. **Instantiation:** For each graph, the script iterates a specified number of times to generate questions. In each iteration, it randomly selects a `QuestionForm` and attempts to instantiate it using the current graph. This involves sampling nodes, lines, or properties from the graph to fill in the template's parameters.
3. **Answer Derivation:** The ground truth answer is derived by programmatically executing a functional representation of the question on the `GraphSpec` object. For example, to answer "How many stations on the Red Line are large?", the program iterates through the nodes on the Red Line and counts how many have their `size` attribute set to `'large'`. If a question cannot be instantiated (e.g., a question about interchanges in a graph with none), the attempt is discarded, and another form is tried.

This process yields a diverse set of questions, ranging from simple property lookups ("Does Red Station have disabled access?") to complex multi-hop reasoning involving counting, comparison, and logical operations ("Which line has more modernist stations, the Blue Line or the Green Line?").

D.4 Final Data Format

The complete dataset is saved as a list of PyTorch Geometric `Data` objects in a single `.pt` file. Crucially, **each `Data` object represents a single graph-question-answer triplet**.

A companion `_mappers.pkl` file is also saved, containing dictionaries that map the raw string values of all categorical features to their integer indices used in the feature tensors.

Each `torch_geometric.data.Data` object has the following key attributes:

`x` (Tensor): Node feature matrix of shape $[N, F_{node}]$, where N is the number of nodes and F_{node} is the size of the embedding of the sentence representing the node features encoded by BERT.

`edge_index` (Tensor): Graph connectivity in COO format, a tensor of shape $[2, E]$, where E is the number of directed edges.

`edge_attr` (Tensor): Edge feature matrix of shape $[E, F_{edge}]$, where F_{edge} is the size of the embedding of the sentence representing the edge features encoded by BERT.

`question` (String): The natural language question, e.g., "How many stations are on the Cyan Line?".

`label` (String): The ground truth answer, serialized to a string (e.g., `'12'`, `'True'`, `'Red Line'`).

`question_type` (String): A unique string identifying the question template used, e.g., `'CountStationsOnLine'`.

`question_group` (String): The general category of the question, e.g., `'count'`.

D.4.1 Node Sentence Representation

The textual attributes of nodes are used to generate a sentence of the following form describing the node:

{NodeName} {has/does not have} disabled access and {has/does not have} rail. It features {Architecture} architecture, has {Cleanliness} cleanliness, {Music} music and is {Size} in size.

D.4.2 Edge Sentence Representation

Similarly, the textual attributes of edges are used to generate a sentence of the following form:

There is a {LineStroke} {LineColour} line from {SourceStation} to {DestinationStation}. It {has/does not have} air conditioning and was built in {BuiltYear}.

D.4.3 Graph Embeddings

The sentence representation of each node/edge is encoded into a 768 dimensional embedding using `bert-base-uncased`. These embeddings are passed to the GNN backbone.

D.4.4 Data Tuples

Each Data example in CLEGR is a tuple of the form (Graph, Question, Answer).

D.4.5 Dataset Statistics

The total default number of graphs generated is 500. The Small CLEGR dataset contains 22000 Fact Based questions(44 Fact Based questions per graph) and 32248 (after natural filteration of invalid questions from a total of 34000 questions) (To check and update) Reasoning questions(68 Reasoning Based questions per graph). The Train, Validation and Test set contain 300, 100 and 100 of the graphs(and their corresponding questions) respectively.

D.4.6 Output Format

The model's output is of one of the following formats: List, Boolean, String, Numeric. The prompt passed to the model will be suffixed with text describing the output format.

tabularx

Table 6: CLEGR Fact-Based Question Template Definitions for the Subway Networks

Template Name	Question Template	Output Type	Scope
StationPropertyCleanliness	How clean is {Station}?	String	Node
StationPropertyCleanliness2	What is the cleanliness level of {Station} station?	String	Node
StationPropertySize	How big is {Station}?	String	Node
StationPropertySize2	What size is {Station}?	String	Node
StationPropertyMusic	What music plays at {Station}?	String	Node
StationPropertyMusic2	Which type of music is played at {Station}?	String	Node
StationPropertyArchitecture	What architectural style is {Station}?	String	Node
StationPropertyArchitecture2	Describe {Station} station's architectural style.	String	Node
StationPropertyDisabled Access	Does {Station} have disabled access?	Boolean	Node
StationPropertyDisabled Access2	Is there disabled access at {Station}?	Boolean	Node
StationPropertyHasRail	Does {Station} have rail connections?	Boolean	Node
StationPropertyHasRail2	Can you get rail connections at {Station}?	Boolean	Node
StationExistence1	Is there a station called {Station}?	Boolean	Node
StationExistence2	Is there a station called {FakeStationName}?	Boolean	Node
StationLine	Which lines is {Station} on?	List	Edge
StationLineCount	How many lines is {Station} on?	Numeric	Edge
StationAdjacentAlwaysTrue	Are {Station} and {Station} adjacent?	Boolean	Edge
StationAdjacent	Are {Station} and {Station} adjacent?	Boolean	Edge
EdgePropertyColor	What color is the line between {Station} and {Station}?	String	Edge
EdgePropertyAircon	Does the line between {Station} and {Station} have air conditioning?	Boolean	Edge
EdgePropertyStroke	What stroke style is the line between {Station} and {Station}?	String	Edge
EdgePropertyBuilt	When was the line between {Station} and {Station} built?	String	Edge

Table 7: CLEGR Reasoning-Based Question Templates for the Subway Networks (Scope: Node)

Template Name	Question Template	Output Type
StationPairAdjacent	Which station is adjacent to both {Station} and {Station}?	String
StationArchitectureAdjacent	Which {Architecture} station is adjacent to {Station}?	String
StationTwoHops	How many other stations are two stops or closer to {Station}?	Numeric
HasCycle	Is {Station} part of a cycle?	Boolean
StationOneApartTrue	Are {Station} and {Station} connected by the same station?	Boolean
StationOneApart	Are {Station} and {Station} connected by the same station?	Boolean
TopologyMostCommonArch	What is the most common architectural style of stations within 2 hops of {Station}?	String
CountIntersectionProperties	How many stations are both large and have disabled access?	Numeric
CompareArchitectureCount	Which architectural style has more stations, {Architecture} or {Architecture}?	String

D.5 CLEGR Computer-Networks

The CLEGR Computer-Networks is a dataset containing graphs representing fictional computer networks. These graphs are generated very similar to the generation used in the Subway Networks dataset. There are some differences in the graph generation primarily due to the fact that "lines" do not exist in this dataset. The differences are highlighted below:

1. **Node-Centric Foundation (vs. Line-Centric):** The process begins by generating a target number of nodes, as specified by the `nodes` parameter. Unlike the transit map's structured

Table 8: CLEGR Reasoning-Based Question Templates for the Subway Networks (Scope: Edge)

Template Name	Question Template	Output Type
StationSameLineTrue	Are {Station} and {Station} on the same line?	Boolean
EdgeFilterAirconCount	How many air-conditioned lines is {Station} connected to?	Numeric
EdgeFilterColorCount	How many {Color} lines is {Station} connected to?	Numeric
PathYearSpan	How many years newer is the newest line between {Station} and {Station} compared to the oldest?	Numeric
PathOptimalColor	What is the most common line color on the shortest path between {Station} and {Station}?	String
PathEarliestBuilt	What is the earliest year a line was built on the shortest path between {Station} and {Station}?	String

Table 9: CLEGR Reasoning-Based Question Templates for the Subway Networks (Scope: Sub-graph)

Template Name	Question Template	Output Type
LineTotalArchitectureCount	How many architectural styles does {Line} pass through?	Numeric
LineTotalMusicCount	How many music styles does {Line} pass through?	Numeric
LineTotalSizeCount	How many sizes of station does {Line} pass through?	Numeric
LineFilterMusicCount	How many stations playing {Music} does {Line} pass through?	Numeric
LineFilterCleanlinessCount	How many {Cleanliness} stations does {Line} pass through?	Numeric
LineFilterSizeCount	How many {Size} stations does {Line} pass through?	Numeric
LineFilterDisabledAccessCount	How many stations with disabled access does {Line} pass through?	Numeric
LineFilterHasRailCount	How many stations with rail connections does {Line} pass through?	Numeric
LineStations	Which stations does {Line} pass through?	List
StationShortestCount	How many stations are between {Station} and {Station}?	Numeric
StationShortestAvoidingCount	How many stations are on the shortest path between {Station} and {Station} avoiding {Cleanliness} stations?	Numeric
StationShortestAvoiding ArchitectureCount	How many stations are on the shortest path between {Station} and {Station} avoiding {Architecture} architecture stations?	Numeric
DistinctRoutes	How many distinct routes are there between {Station} and {Station}?	Numeric
CountEqualSizeStation	How many stations in {Line} are of the same size as {Station}?	Numeric
LineIntersectionStations	How many stations are shared between the {Line} and the {Line}?	Numeric
NodeOnPath	Is {Station} on the shortest path between {Station} and {Station}?	Boolean
PathMostCommonMusic	What is the most common music style on the shortest path between {Station} and {Station}?	String
CompareLineDisabledAccess	Which line has more stations with disabled access, {Line} or {Line}?	String

placement along Bézier curves, these system nodes are scattered with **random (x, y) coordinates** across the map space. The concept of a 'LineSpec' is entirely absent; nodes are the primary, independent entities from the outset.

2. **Emergent Topology via Proximity (vs. Pre-Defined Paths):** Edge creation is not determined by a sequential path. Instead, the `gen_edges` function implements a k-nearest neighbor algorithm. After the final node positions are set, a `KDTree` is used to find the 'k' closest neighbors for each node (where 'k' is derived from the `avg_degree` parameter). Edges are then created between a node and its neighbors.

3. **Hub Formation (vs. Interchange Creation):** The function `coalesce_nearby_nodes` serves a different conceptual purpose here. In the transit model, it created interchanges by merging nodes from different lines. Here, it addresses the potential for random placement to create unrealistic clusters of nodes. By merging nodes that are too close, such node clusters are reduced.
4. **Intrinsic Edge Properties (vs. Inherited):** In the transit model, edge properties (like color and stroke) were inherited from their parent line. In this dataset, every edge is assigned its own set of properties directly and randomly from the `EdgeProperties` dictionary. Attributes like `bandwidth_units`, `latency_ms`, and `encryption_status` are intrinsic to the connection itself.

D.6 Feature Schema

The feature schema is completely redesigned to reflect the computer network domain.

D.6.1 Node Features (System Nodes)

Each node represents a computer system, server, or device. Its attributes are one-hot encoded into the node feature tensor `x`.

`id` (String): A unique identifier for the system node.

`name` (String): The programmatically generated name of the node.

`x, y` (Float): The 2D coordinates of the node in the grid space.

`status` (String): The operational status of the node (e.g., `'Operational'`, `'Offline'`, `'Overloaded'`).

`security_level` (String): An assigned security clearance level (e.g., `'Public'`, `'Internal'`, `'Restricted'`).

`location_sector` (String): The logical or physical sector where the node is located (e.g., `'Sector_Red'`).

`firmware_version` (String): The version of the node's firmware (e.g., `'v1.1'`, `'v2.0'`).

`power_consumption_units` (Integer): A measure of the node's power draw.

D.6.2 Edge Features (Connections)

Each edge represents a network connection between two system nodes. There is no 'LineSpec' entity. Edge attributes are one-hot encoded into the edge feature tensor `edge_attr`.

`node1_id, node2_id` (String): The IDs of the two nodes being connected. The class attribute is named station1, station2 to be compatible with the other domain's codebase.

`bandwidth_units` (Integer): A measure of the connection's data throughput capacity.

`latency_ms` (Integer): The latency of the connection in milliseconds.

`encryption_status` (String): The encryption state of the connection (e.g., `'Encrypted'`, `'Unencrypted'`).

D.6.3 Node Sentence Representation

Similar to the Subway Netowrks dataset, the textual attributes of nodes are used to generate a sentence of the following form describing the node:

System node {NodeName} is in {LocationSector} with status {Status}. It has security level {SecurityLevel}, firmware {FirmwareVersion}, and consumes {PowerConsumptionUnits} power units.

D.6.4 Edge Sentence Representation

Similarly, the textual attributes of edges are used to generate a sentence of the following form:

A link connects {SourceNodeName} and {DestinationNodeName}. It has {BandwidthUnits} bandwidth units, {Latency}ms latency, and its encryption status is {EncryptionStatus}.

Table 11: Fact-Based Question Template Definitions for the CLEGR Computer-Networks Dataset

Template Name	Question Template	Output Type	Scope
NodePropertyStatus	What is the status of node {Node}?	String	Node
NodePropertySecurity	What is the security level of node {Node}?	String	Node
NodePropertyLocation	Which sector is node {Node} located in?	String	Node
NodePropertyFirmware	What firmware version runs on {Node}?	String	Node
NodePropertyPower	How many power units does {Node} consume?	Numeric	Node
NodeExistence1	Is there a node named {Node} in the grid?	Boolean	Node
NodeExistence2	Is there a node named {FakeNodeName} in the grid?	Boolean	Node
NodeAdjacentTrue	Are nodes {Node} and {Node} directly linked?	Boolean	Edge
NodeAdjacent	Are nodes {Node} and {Node} directly linked?	Boolean	Edge

Table 12: Reasoning-Based Question Template Definitions for the CLEGR Computer-Networks Dataset

Template Name	Question Template	Output Type	Scope
CountNodesWithStatus	How many nodes have status {Status}?	Numeric	Sub-graph
ListNodesInSector	List all nodes in {Sector}.	List	—
MostCommonFirmware	What is the most common firmware version?	String	Sub-graph
CountNodesWithTwoProps	How many nodes in {Sector} have security level {Security Level}?	Numeric	Sub-graph
CountNeighborsOperational	How many neighbors of {Node} are 'Operational'?	Numeric	Sub-graph
ShortestPathLen	How many nodes are on shortest path between {Node} and {Node}?	Numeric	Sub-graph
NodesBetween	How many nodes lie between {Node} and {Node} on that path?	Numeric	Sub-graph
PathAvoidingStatus	Is there a path from {Node} to {Node} avoiding status {Status}?	Boolean	Sub-graph
WithinHops	How many other nodes are within 3 hops of {Node}?	Numeric	Node
HasCycle	Is {Node} part of a cycle?	Boolean	Node
OneIntermediary	Are {Node} and {Node} connected via exactly one intermediary?	Boolean	Edge

D.7 Graph Statistics

Table 10 shows the graph statistics for CLEGR Computer Networks, which is also similar to that of CLEGR Subway Networks.

Table 10: Graph statistics for CLEGR Computer Networks with different graph sizes.

Metric	CLEGR Computer-Networks
Average Number of Nodes	21.64 ± 3.93
Average Number of Edges	28.61 ± 5.30

All the question templates for the CLEGR Computer Networks are present at Table 11 & Table 12

E CLEGR Evaluation

E.1 Evaluation Metrics by Answer Type

CLEGR includes four distinct answer formats, each evaluated with tailored methods:

- **Categorical Answers** (e.g., station names, architecture types):
  Evaluated using *exact match* after normalization (lowercasing and punctuation removal).

- **Boolean Answers** (e.g., yes/no questions):
  Text responses such as "yes", "no", "true", or "false" are mapped to binary values. We compute *accuracy*, *F1-score*, and *Matthews Correlation Coefficient (MCC)*.

- **Numeric Answers** (e.g., distances, years):
  Evaluated using approximate equality via `numpy.isclose`. If direct parsing fails, we extract numbers using regex. Metrics include *accuracy*, *Mean Absolute Error (MAE)*, and *Root Mean Square Error (RMSE)*.

- **Set-Valued Answers** (e.g., lists of stations):
  Scored using set-based *precision*, *recall*, and *F1-score*, based on overlap with the ground truth set.

E.2 Overall Evaluation

Each question is scored using the appropriate method for its answer type. We report **overall accuracy** as the primary evaluation metric, defined as the proportion of correctly answered questions across the full test set. This ensures a fair and consistent comparison across models and tasks.

F Prompt Templates

F.1 Example Prompts for NC, each dataset

F.1.1 Arxiv

Textual Prompt Format

<s>[INST] Title: {Title}\nAbstract: {Abstract}\nAnswer the following question: Which subcategory does this paper belong to? Please only output the most likely answer from the following subcategories and nothing else: {Comma seperated Category List}. \nAnswer: [/INST] {Label} [/s]

F.1.2 Cora

Textual Prompt Format

<s>[INST] Title: {Title}\nAbstract: {Abstract}\nAnswer the following question: Which subcategory does this paper belong to? Please only output the most likely answer from the following subcategories and nothing else: theory, reinforcement learning, genetic algorithms, neural networks, probabilistic methods, case based, rule learning. \nAnswer: [/INST] {Label} [/s]

F.1.3 CiteSeer

Textual Prompt Format

<s>[INST] Text: {Text}\nAnswer the following question: Which category does this paper belong to? Please only output the most likely answer from the following categories directly and nothing else: Agents, AI, DB, IR, ML, HCI. \nAnswer: [/INST] {Label} [/s]

F.1.4 Computers

Textual Prompt Format

<s>[INST] {Context}\nAnswer the following question: Which computer product subcategory does this review belong to? Please only output the most likely answer from the following subcategories and nothing else: computer accessories and peripherals, tablet accessories, laptop accessories, computers and tablets, computer components, data storage, networking products, monitors, servers, tablet replacement parts \n\n Answer: [/INST] {Label} [/s]

F.1.5 Photo

Textual Prompt Format

<s>[INST] {Context}\nAnswer the following question: Which photography related subcategory does this description belong to? Please only output the most likely answer from the following subcategories and nothing else: Film Photography, Video, Digital Cameras, Accessories, Binoculars & Scopes, Lenses, Bags & Cases, Lighting & Studio, Flashes, Tripods & Monopods, Underwater Photography, Video Surveillance \n\n Answer: [/INST] {Label} [/s]

F.1.6 History

Textual Prompt Format

<s>[INST] {Context}\nAnswer the following question: Which history related subcategory does this description belong to? Please only output the most likely answer from the following subcategories and nothing else: World, Americas, Asia, Military, Europe, Russia, Africa, Ancient Civilizations, Middle East, Historical Study & Educational Resources, Australia & Oceania, Arctic & Antarctica \n\n Answer: [/INST] {Label} [/s]

F.1.7 CLEGR Facts and CLEGR Reasoning

Textual Prompt Format

<s>[INST] --- Nodes ---\n{CSV String describing all the Nodes}\n--- Edges ---\n{CSV String describing all the Edges}Above is the representation of a synthetic subway network. All stations and lines are completely fictional. Keep in mind that the subway network is not real. All information necessary to answer the question is present in the above representation. The question is: {Question}\n\n{Answer Format Suffix} [/INST] {Label} [/s]

CSV String describing all the Nodes: A string describing all the nodes formatted like a CSV files with rows representing each node and columns describing different attributes of the nodes following the G-Retriever [9] pipeline.

CSV Header

"id", "name", "disabled_access", "has_rail", "architecture", "cleanliness", "music", "size"

CSV String describing all the Edges: A string describing all the nodes formatted like a CSV files with rows representing each edge and columns describing different attributes of the edges.

CSV Header

"source_id", "target_id", "line_color", "line_stroke", "has_aircon", "built"

Answer Format Suffix: Depending on the question and the output format expected by the question, one of the following is suffixed to the question.

Table 13: Zero-shot transfer accuracy (%) from Computers to History and Photo.

Model	Computers → History	Computers → Photo
Graph-Language Models		
TEA-GLM (Llama 8B)	57.94 ± 4.97	5.44 ± 2.50
TEA-GLM (Phi 3.5B)	18.16 ± 5.70	3.71 ± 0.16
G-Token (GSAGE) (Llama 8B)	25.65 ± 17.15	5.01 ± 1.65
G-Token (GSAGE) (Phi 3.5B)	19.62 ± 5.07	2.52 ± 0.22
Soft-Prompted LLMs		
Llama3-8B-SPT	43.56 ± 12.50	6.00 ± 1.19
Phi3-3.5B-SPT	32.85 ± 13.79	2.85 ± 0.42
Random Baseline (1/num_classes)		
Uniform Guessing	8.33	8.33

Table 14: Zero-shot transfer accuracy (%) from Arxiv to Cora.

Model	Arxiv → Cora
Graph-Language Models	
TEA-GLM (Llama 8B)	10.02 ± 2.73
TEA-GLM (Phi 3.5B)	5.80 ± 4.73
G-Token (GSAGE) (Llama 8B)	5.12 ± 4.18
G-Token (GSAGE) (Phi 3.5B)	3.76 ± 4.32
Soft-Prompted LLMs	
Llama3-8B-SPT	16.57 ± 1.15
Phi3-3.5B-SPT	17.19 ± 0.68
Random Baseline (1/num_classes)	
Uniform Guessing	14.29

Output Format Suffixes

String Output: Answer directly:
Bool Output: Answer with 'True' or 'False': \n\nAnswer:
List Output: Output a comma-separated list:
Count Output: Answer with a number: \n\nAnswer:
Cycle Detection Question: Answer with 'True' if it is in a cycle, otherwise 'False': \n\nAnswer:

G Extended Experimental Results

G.1 Node Classification (Zero-Shot Transfer)

Effective zero-shot transfer learning is a key advantage claimed by GLMs like TEA-GLM. To evaluate this, we perform zero-shot transfer experiments across distinct semantic domains (Computers to History and Photo, and Arxiv to Cora). Surprisingly, our results (Table 14 Table 13) indicate negligible gains for GLMs over soft-prompted LLM baselines, and in some cases, GLMs even perform worse. This finding challenges the claimed superior generalization of GLMs, highlighting that even in tasks considered advantageous for GLMs, simple soft-prompted LLMs exhibit comparable or superior performance, raising questions about the practical value of current GLM designs in zero-shot contexts of Node classification.

To achieve cross-dataset transfer capability, where models trained on one graph dataset can perform reasoning on different target datasets, we design instructions that include both the task description and the complete set of alternative answers from both source and target domains. This approach follows the methodology established by TEA-GLM.

For the Arxiv → Cora transfer scenario, the training instruction is structured as: "Which research area does this paper belong to? Please directly give the most likely answer from the following categories: ans", where ans contains class labels from both the Arxiv dataset (computer science subcategories such as "cs.AI", "cs.LG") and the Cora dataset (machine learning areas such as "Neural Networks", "Reinforcement Learning").

Similarly, for the Computers → History and Computers → Photo transfers, the instruction follows: "Which product category does this item belong to? Please select from the following options: ans", where ans includes classes from the Computers dataset alongside the respective target dataset classes (History product types or Photo equipment categories). Including alternative answers from both source and target datasets enables the model to learn the task of "selecting the correct answer from a given set according to the reasoning requirements" rather than memorizing dataset-specific label mappings, thus facilitating effective knowledge transfer across different graph domains.

Table 15: Zero-shot Transfer Accuracy: Trained on CLEGR Subway Reasoning, Tested on CLEGR Computer Network Reasoning

Model	Trained → Tested	Accuracy
GraphSAGE Llama3-8B	CLEGR Subway → CLEGR Computer	33.16 ± 0.97
GraphSAGE Phi3-3.5B	CLEGR Subway → CLEGR Computer	27.33 ± 2.51
GraphSAGE Phi4-14B	CLEGR Subway → CLEGR Computer	39.3 ± 2.34
TEA-GLM Llama3-8B	CLEGR Subway → CLEGR Computer	32.92 ± 2.27
TEA-GLM Phi3-3.5B	CLEGR Subway → CLEGR Computer	27.67 ± 2.52
TEA-GLM Phi4-14B	CLEGR Subway → CLEGR Computer	40.72 ± 2.87
Llama3-8B-SPT	CLEGR Subway → CLEGR Computer	30.89 ± 0.66
Phi3-3.5B-SPT	CLEGR Subway → CLEGR Computer	26.81 ± 1.45
Phi4-14B-SPT	CLEGR Subway → CLEGR Computer	36.79 ± 1.17

Table 16: In-Domain Accuracy: Trained and Tested on CLEGR Computer Network Reasoning

Model	Trained → Tested	Accuracy
GraphSAGE Llama3-8B	CLEGR Computer → CLEGR Computer	52.33 ± 1.34
GraphSAGE Phi3-3.5B	CLEGR Computer → CLEGR Computer	47.53 ± 1.46
TEA-GLM Llama3-8B	CLEGR Computer → CLEGR Computer	47.94 ± 4.32
TEA-GLM Phi3-3.5B	CLEGR Computer → CLEGR Computer	39.67 ± 4.39
Llama3-8B-SPT	CLEGR Computer → CLEGR Computer	48.22 ± 2.07
Phi3-3.5B-SPT	CLEGR Computer → CLEGR Computer	46.38 ± 2.48

Table 17: Train on CLEGR Subway Facts, Test on CLEGR Computer (Facts / Reasoning).

Model (Method + LLM)	Train → Test	Accuracy
GraphSAGE Llama3-8B	CLEGR Subway Facts → CLEGR Computer Facts	87.99 ± 1.55
GraphSAGE Phi3-3.5B	CLEGR Subway Facts → CLEGR Computer Facts	87.96 ± 1.63
GraphSAGE Phi4-14B	CLEGR Subway Facts → CLEGR Computer Facts	99.49 ± 0.30
TEA-GLM Llama3-8B	CLEGR Subway Facts → CLEGR Computer Facts	91.68 ± 1.67
TEA-GLM Phi3-3.5B	CLEGR Subway Facts → CLEGR Computer Facts	82.65 ± 2.53
GraphSAGE Llama3-8B	CLEGR Subway Facts → CLEGR Computer Reasoning	34.34 ± 2.76
GraphSAGE Phi3-3.5B	CLEGR Subway Facts → CLEGR Computer Reasoning	31.10 ± 0.90
GraphSAGE Phi4-14B	CLEGR Subway Facts → CLEGR Computer Reasoning	40.27 ± 2.24
TEA-GLM Llama3-8B	CLEGR Subway Facts → CLEGR Computer Reasoning	34.06 ± 2.09
TEA-GLM Phi3-3.5B	CLEGR Subway Facts → CLEGR Computer Reasoning	28.17 ± 2.98

G.2 CLEGR (Zero Shot Transfer)

This section presents the accuracy results of GLMs on different transfer learning scenarios across domains and knowledge types (Facts vs. Reasoning). We present 5 tables (Table 15, Table 16,

Table 18: Train on CLEGR Subway Reasoning, Test on CLEGR Computer (Facts / Reasoning).

Model (Method + LLM)	Train → Test	Accuracy
GraphSAGE Llama3-8B	CLEGR Subway Reasoning → CLEGR Computer Facts	45.26 ± 35.78
GraphSAGE Phi3-3.5B	CLEGR Subway Reasoning → CLEGR Computer Facts	79.50 ± 5.98
GraphSAGE Phi4-14B	CLEGR Subway Reasoning → CLEGR Computer Facts	90.53 ± 4.83
TEA-GLM Llama3-8B	CLEGR Subway Reasoning → CLEGR Computer Facts	71.74 ± 5.81
TEA-GLM Phi3-3.5B	CLEGR Subway Reasoning → CLEGR Computer Facts	76.71 ± 2.50
GraphSAGE Llama3-8B	CLEGR Subway Reasoning → CLEGR Computer Reasoning	26.06 ± 11.86
GraphSAGE Phi3-3.5B	CLEGR Subway Reasoning → CLEGR Computer Reasoning	28.51 ± 2.45
GraphSAGE Phi4-14B	CLEGR Subway Reasoning → CLEGR Computer Reasoning	36.07 ± 2.12
TEA-GLM Llama3-8B	CLEGR Subway Reasoning → CLEGR Computer Reasoning	34.37 ± 0.94
TEA-GLM Phi3-3.5B	CLEGR Subway Reasoning → CLEGR Computer Reasoning	28.34 ± 1.40

Table 19: Cross-Modality Transfer Between CLEGR Subway Facts and CLEGR Subway Reasoning.

Model (Method + LLM)	Train → Test	Accuracy
GraphSAGE Llama3-8B	CLEGR Subway Facts → CLEGR Subway Reasoning	35.66 ± 0.31
GraphSAGE Phi3-3.5B	CLEGR Subway Facts → CLEGR Subway Reasoning	30.30 ± 2.33
GraphSAGE Phi4-14B	CLEGR Subway Facts → CLEGR Subway Reasoning	36.09 ± 2.13
TEA-GLM Llama3-8B	CLEGR Subway Facts → CLEGR Subway Reasoning	33.93 ± 0.76
TEA-GLM Phi3-3.5B	CLEGR Subway Facts → CLEGR Subway Reasoning	29.19 ± 0.12
GraphSAGE Llama3-8B	CLEGR Subway Reasoning → CLEGR Subway Facts	42.38 ± 33.81
GraphSAGE Phi3-3.5B	CLEGR Subway Reasoning → CLEGR Subway Facts	62.86 ± 5.34
GraphSAGE Phi4-14B	CLEGR Subway Reasoning → CLEGR Subway Facts	72.28 ± 1.86
TEA-GLM Llama3-8B	CLEGR Subway Reasoning → CLEGR Subway Facts	69.52 ± 5.88
TEA-GLM Phi3-3.5B	CLEGR Subway Reasoning → CLEGR Subway Facts	52.86 ± 6.82

Table 17, Table 18, and Table 19) testing various scenarios like Zero-Shot domain transfer (e.g. CLEGR Subway Networks to CLEGR Computer Networks) and Zero-Shot in-domain but question-type transfer (e.g. CLEGR Facts to CLEGR Reasoning and vice versa).

Now, we further answer the following questions:

RQ4: Do GLMs provide better zero-shot generalization to other domains? To assess if the structural encoding in GLMs leads to better zero-shot performance, we evaluate their transfer capability from the original subway domain presented in CLEGR-Reasoning to a new, structurally analogous Computer-Networks domain. The results are presented in Figure 5. Neither TEA-GLM nor GraphToken achieve significant performance improvements compared to soft prompted LLMs, showcasing that the injection of structural information by GLMs does not have a significant impact on zero-shot transfer.

RQ5: How does GLM performance scale with increasing graph size? We test whether increasing the size of graphs present in CLEGR show a scenario where GLMs outperform language-only baselines. For evaluation purposes, we introduce CLEGR-Large, a modification of CLEGR that contains graphs approximately three times larger. As graph complexity increased, both approaches exhibited nearly identical performance degradation on reasoning and factual tasks (see Figure 6). This parallel decline demonstrates that GLMs offer no inherent advantage for handling large graph structures, as their performance scales just like simpler, text-based methods. All these results corroborate our findings, highlighting the negligible advantages produced by GLMs.

H Analyzing Representation Alignment

Our findings in Section 2 and Section 4 empirically demonstrate that simply soft-prompting LLMs matches or exceeds GLMs across all our evaluations (Table 1, Figure 4). In this section, we experimentally investigate if the cause behind similar performance metrics for different models is similar internal representations. To verify this, we employ Centered Kernel Alignment (CKA) [15]

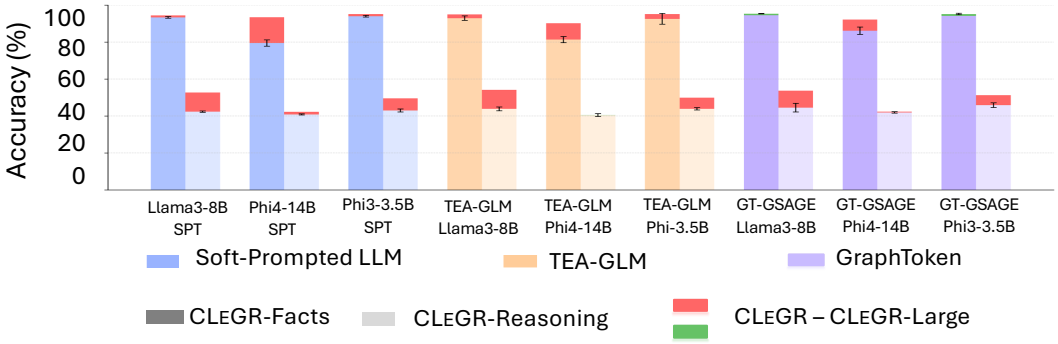

Figure 5: Results on graphs larger than standard CLEGR demonstrate that increased structural complexity provides no advantage to GLMs over soft-prompted baselines, with both approaches showing comparable performance degradation.

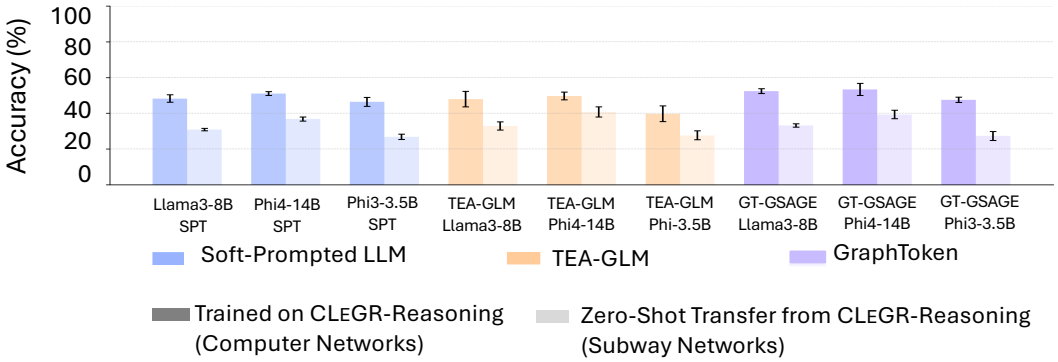

Figure 6: Zero-shot generalization from subway to computer network domains shows GLMs provide no transfer benefits compared to soft prompted approaches, indicating structural encoders do not enhance cross-domain reasoning capabilities.

to measure representational overlap between GLMs and soft-prompts. As shown in Figure 7, we analyze the relationship between the performance gap $|A_{\text{GLM}} - A_{\text{soft}}|$ (x-axis), where where A denotes accuracy and representational similarity $\text{CKA}(\mathbf{H}_{\text{GLM}}, \mathbf{H}_{\text{soft}})$ (y-axis), where $\mathbf{H}$ represents the activation over graph and soft-prompt tokens respectively.

We observe that most datasets cluster in the upper-left region of Figure 7 (low performance gap, high CKA). *Semantically-sufficient* datasets and CLEGR tasks maintain high CKA across all layers. *Structurally-sufficient* datasets show lower CKA in middle layers (9-24), correlating with our observed low performance by soft-prompted baselines on these datasets. Our analysis highlights that GLMs learn different representations only when datasets are *structurally-sufficient*, i.e., when the LLM's textual reasoning capabilities are underutilized.

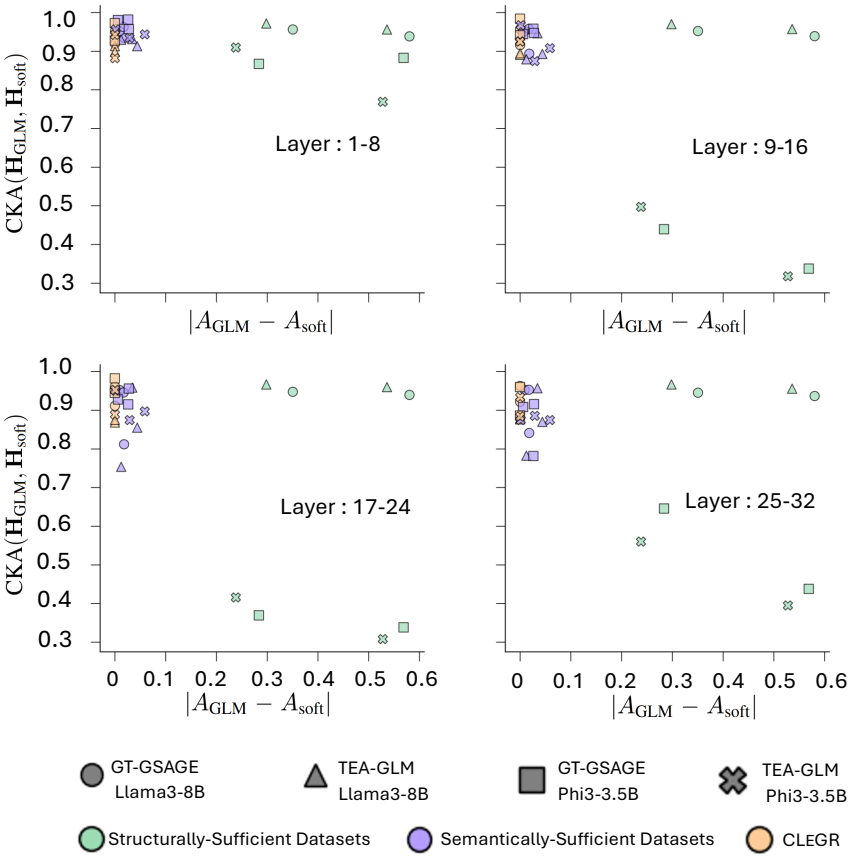

Figure 7: CKA aligns strongly when performance is similar. *Semantically-sufficient* datasets show near-identical representations; *Structurally-sufficient* datasets diverge in mid layers, aligning with soft-prompt failures. Each point denotes a (dataset, model, layer-group) triplet.