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

Anonymous ACL submission

Abstract

Recent research has extensively explored the graph-reasoning capabilities of Large Language Models (LLMs) through textual descriptions. However, benchmarks specifically designed for Graph-Language Models (GLMs), which integrate Graph Neural Networks (GNNs) with LLMs, remain significantly underdeveloped. In this work, we first demonstrate that existing GLM evaluations, largely repurposed from unimodal node and edge level tasks, fail to assess true multimodal integration. Our analysis reveals that strong performance on these benchmarks is achievable using textual or structural features in isolation, bypassing the need for joint reasoning. To bridge this gap, we introduce CLEGR (Compositional Language-Graph Reasoning), a benchmark explicitly designed to evaluate multimodal reasoning over graph topology and textual semantics. Evaluation of representative GLMs on CLEGR shows that they exhibit significant performance degradation on CLEGR tasks and unimodal soft-prompted LLMs perform on par with complex multimodal GLMs. These findings collectively highlight limitations in the graph reasoning capabilities of existing GLMs and provide a foundation for advancing the community toward explicit multimodal reasoning involving graph structure and language.

1 Introduction & Related Work

The success of Vision-Language Models (VLMs) like GPT-4V (Yang et al., 2023b) and LlaVa (Liu et al., 2023) demonstrates the transformative potential of integrating different data modalities to enhance complex reasoning capabilities through visual question answering, image captioning, and multimodal instruction following (Dai et al., 2023). Inspired by this paradigm, there has been an increasing interest in using LLMs for graph-based applications (Li et al., 2024a; Chen et al., 2024). Depending on the role of LLMs and their interaction with graph neural networks (GNNs), the LLM on graphs techniques can be classified into treating LLMs as the final component for prediction (LLM as Predictor) (Wang et al., 2024; He et al., 2024b; Liu et al., 2024b), treating LLMs as the feature extractor for GNNs (LLM as Encoder) (Chien et al., 2022; He et al., 2024a; Liu et al., 2024a), or align the latent space of LLMs with GNNs (LLM as Aligner) (Zhao et al., 2023; Jin et al., 2023).

Beyond the role of the LLM, the relationship between graph structure and text defines the learning objective. While traditional graph learning often operates on pure graphs, many real-world scenarios involve Text-Attributed Graphs (TAGs), where nodes represent textual entities, such as academic papers or social media profiles, and edges capture their relational dependencies (Yang et al., 2023a; Hu et al., 2021). Integrating these textual semantics is critical for the generalization of foundational models (Arun et al., 2025); however, the challenge lies in effectively fusing these distinct modalities.

To exploit this rich information, the LLM-as-predictor framework has emerged as a particularly promising direction (Perozzi et al., 2024; Wang et al., 2024; He et al., 2024b; Liu et al., 2024b). In this work, we focus on evaluating models that utilize a GNN-based backbone to encode structure and an LLM for reasoning and prediction. This architecture is essential for our study as it allows for a clear decomposition of modality-specific contributions, and analysis of evaluations which have both graph and language component as input. Throughout this manuscript, we refer to this integrated class of models as Graph-Language Models (GLMs) and investigate their performance as multimodal systems (see Fig. 1).

As GLMs combine structural and textual information, a natural question arises: *how should we evaluate their ability to jointly utilize both modalities?* Ideally, benchmarks should require genuine

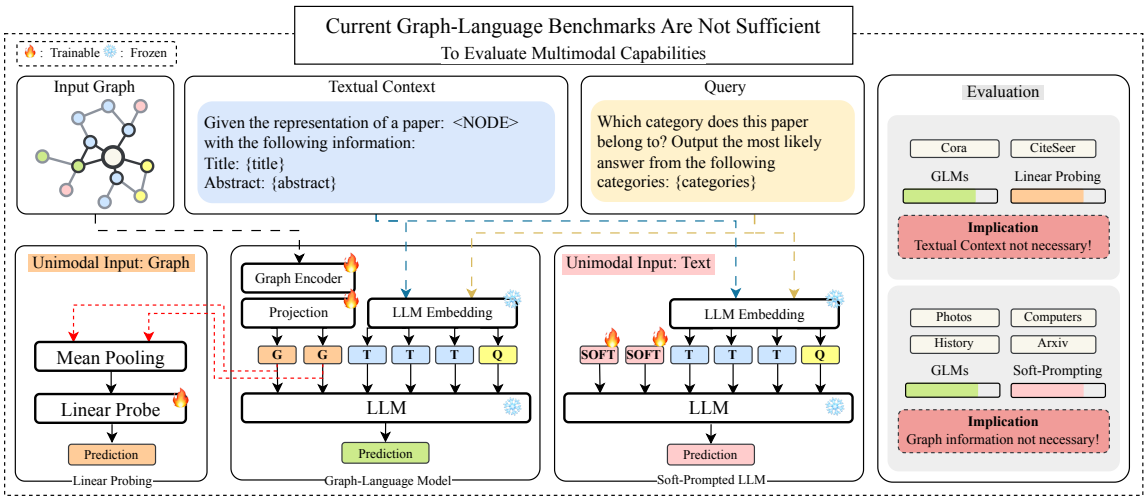


Figure 1: Existing datasets can be solved using unimodal information alone: linear probing on graph encoders achieves strong performance matching GLMs in some datasets, while soft-prompted LLMs match GLMs in others. This demonstrates that current benchmarks fail to evaluate true multimodal capabilities.

integration of graph structure and language semantics, ensuring that neither modality alone is sufficient for strong performance. However, despite the rapid development of GLM architectures, current evaluation strategies fall short in three key ways: i) graph reasoning benchmarks such as GraphEval36k (Wu et al., 2025), GraphArena (Tang et al., 2025), and GraCore (Yuan et al., 2025) focus on the algorithmic capabilities of LLMs alone, using text-only descriptions of graph problems. While these test logical reasoning, they entirely bypass the structural modality, requiring no graph encoder and thus being inadequate to evaluate multimodal integration; ii) Knowledge Graph (KGQA) benchmarks such as ExplaGraphs (Saha et al., 2021) and WebQSP (Yih et al., 2016) are designed to evaluate reasoning over rich text but often rely on trivial structural patterns that allow models to succeed through memorization rather than genuine graph reasoning (Xu et al., 2024a) and use real-world knowledge graphs rather than synthetic data, making it nearly impossible to disentangle whether strong performance stems from the LLM's pretraining knowledge or from true graph-language understanding; and iii) benchmarks like GLBench (Li et al., 2024b) augment traditional GNN datasets (*e.g.,* Cora (McCallum et al., 2000), CiteSeer (Yang et al., 2016), PubMed (Sen et al., 2008)) with node text attributes for node classification tasks. However, these evaluations do not systematically isolate or measure the individual contribution of each modality, making it unclear whether strong performance stems from true multimodal synergy or from exploiting *shortcuts* in a single modality.

Recent work has raised concerns that inadequate evaluation practices in graph learning may undermine the field's relevance (Bechler-Speicher et al., 2025). This problem extends to multimodal graph-language models, which continue to rely on the traditional GNN benchmarks and evaluation practices. In this work, we demonstrate that node and edge classification datasets commonly used to evaluate GLMs (Perozzi et al., 2024; Wang et al., 2024; Liu et al., 2024b) fail to assess multimodal integration, as unimodal baselines achieve comparable and sometimes even superior performance. To address this, we introduce CLEGR (Compositional Graph-Language Reasoning), a question-answering benchmark comprising **500** diverse graphs and **32,000** questions explicitly designed to quantify both structural and semantic understanding. Our evaluation reveals that current GLMs perform poorly on CLEGR, achieving results comparable to unimodal LLM baselines and exposing fundamental limitations in their integration strategies.

Our contributions are: (1) We demonstrate that widely-used benchmarks fail to evaluate multimodal capabilities across diverse tasks (node classification and link prediction) and evaluation settings (fully supervised and zero-shot), as strong performance can be achieved using a single modality alone; (2) We introduce CLEGR, to the best of our knowledge, the first graph-language reasoning benchmark designed to assess multimodal ca-

pabilities of GLMs. CLEGR spans 500 diverse graphs and 32,000 questions across multiple structural levels (node, edge, subgraph) and reasoning types (aggregation, filtering, path reasoning, topology analysis), systematically preventing unimodal shortcuts; and (3) Through extensive experiments on CLEGR, we show that **current GLMs do not deliver the multimodal value they claim**. Despite multimodal architectures, they perform poorly on structural reasoning tasks at levels comparable to unimodal LLM baselines, emphasizing the need for more sophisticated integration architectures and advancing multimodal graph-language research.

2 Preliminaries

Here, we detail the notations and formulations for the graph-language and soft-prompted large language models.

2.1 Graph-Language Models

Formally, a graph-language model GLM $= (M_l, M_g, M_P)$ comprises three main components: an LLM M_l, a graph encoder M_g, and a linear projector M_P 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 total number of nodes in the graph, $M = |\mathcal{E}|$ is the total 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:

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

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^*.

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

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

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.

2.2 Soft Prompting

To isolate the contribution of graph encoders in GLMs, we use soft prompting as a baseline. Soft Prompting (Lester et al., 2021) introduces learnable prompt tokens which are concatenated with the embeddings of the input given to the LLM. We note that a GLM can be viewed as a soft-prompt LLM where a graph encoder is learned 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 $\mathbf{s} \in \mathbb{R}^{k \times d}$ is the trainable soft-prompt vector. For a given task, the soft-prompted LLM prediction is:

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

where 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. Throughout the manuscript, we refer to soft-prompted models using the suffix SPT/SOFT.

3 Are Current Benchmarks Evaluating Multimodal Integration?

With the increasing development of GLMs, it is important to ask: *do current benchmarks effectively evaluate if GLMs use both graph and text information together?* To study this, we analyze common GLM evaluation practices.

We design our investigation around two core research questions: **RQ❶:** Are current evaluation datasets *single-modality-sufficient*, *i.e.,* can strong performance be achieved using graph or language information alone? **RQ❷:** If unimodal baselines match GLM performance, what does this reveal about the adequacy of these benchmarks for assessing true multimodal capabilities?

3.1 Experimental Setup

For reproducibility and maintaining standard evaluation practices, we use the same datasets and tasks that prior works utilized in their experiments.

Model	Computers	Photo	History	Arxiv	Cora	CiteSeer
GNNs						
GAT	93.67±0.28	96.51±0.20	82.81±0.74	73.30±0.18	86.05±1.37	71.12±0.84
GCN	**93.94±0.13**	95.74±0.10	82.91±0.45	73.53±0.12	86.98±0.95	72.14±0.67
GraphSAGE	93.11±0.23	**96.54±0.15**	83.24±0.82	73.00±0.28	**87.31±0.81**	**72.26±0.70**
Graph–Language Models						
TEA-GLM Llama3-8B	73.10±1.07	70.51±1.33	81.56±5.39	73.08±0.00	82.26±1.23	48.05±1.28
TEA-GLM Phi3-3.5B	69.88±0.61	64.81±4.13	81.63±0.58	67.38±1.31	82.49±1.17	42.08±3.36
G-Token (GSAGE) Llama3-8B	76.13±0.58	76.61±0.92	**85.91±0.27**	75.49±0.28	86.72±0.93	53.23±0.53
G-Token (GSAGE) Phi3-3.5B	72.38±0.90	73.50±2.55	85.15±0.31	71.38±0.19	86.60±1.12	44.69±1.22
GraphPrompter Llama3-8B	72.03±1.82	73.83±2.47	85.72±0.86	69.80±3.91	80.53±1.76	49.50±1.05
GraphPrompter Phi3-3.5B	64.10±0.39	66.13±1.96	84.32±0.95	61.34±1.40	81.47±2.85	42.12±1.88
Soft-Prompted LLMs						
Llama3-8B-SPT	74.34±0.63	74.90±0.57	84.99±0.66	**76.03±0.45**	28.69±2.70	18.21±0.26
Phi3-3.5B-SPT	69.74±0.26	70.71±3.81	84.55±0.43	72.04±1.15	29.74±1.12	18.28±1.43

Table 1: **Node classification accuracy** (%) in fully supervised setting. Unimodal baselines (either language-only or graph-only) achieve competitive performance across all datasets, indicating these evaluations may not effectively assess multimodal integration. Bold indicates best overall, underline indicates second best.

Datasets. We evaluate on six widely-used TAG datasets: Cora (McCallum et al., 2000), CiteSeer (Yang et al., 2016), Computers, Photo, History (Shchur et al., 2019), and Arxiv (Hu et al., 2021). These datasets form the backbone of current graph-language evaluation practices, appearing in GLBench (Li et al., 2024b), GraphFM (Xu et al., 2024b), TAG (Yan et al., 2023), and Planetoid.

Graph-Language Models. We evaluate three prominent GLM architectures: (1) TEA-GLM (Wang et al., 2024), which performs graph learning by encoding graph structure through textual descriptions and leveraging LLMs for reasoning, (2) GraphToken (Perozzi et al., 2024), which learns discrete graph tokens to represent structural information and integrates them with language model processing, and (3) GraphPrompter (Liu et al., 2024b), which leverages graph prompting. GLMs are evaluated with both Llama3-8B and Phi3-3.5B backbones, ensuring robustness across different LLMs.

Unimodal Baselines. The key to our investigation lies in systematically isolating the contribution of each modality. We employ unimodal baselines: (1) *Graph-only*: GAT (Veličković et al., 2018), GCN (Kipf and Welling, 2017), and GraphSAGE (Hamilton et al., 2018), which operate exclusively on graph structure without access to text tokens, and (2) *Language-only*: Soft-prompted LLMs (Eq. 3) using Llama3-8B and Phi3-3.5B backbones, which process only node text attributes without any structural information. Critically, if either baseline achieves a high and comparable performance to GLMs, it indicates that the evaluation is *single-modality-sufficient* and does not require genuine multimodal integration. We use identical training procedures across all models and report mean accuracy over five random seeds. Implementation details are in the Appendix A.6

3.2 Analysis of Modality Contribution

Table 1 reveals that on Computers, Photo, History, and Arxiv, soft-prompted LLMs achieve performance on-par or superior to GLMs. For instance, on Arxiv, Llama3-8B-SPT achieves 76.03%, outperforming all GLMs. This indicates that semantic content in node text is sufficient for classification, rendering graph encoders unnecessary.

On Cora and CiteSeer, GNNs achieve 87.31% and 72.26% respectively, while soft-prompted LLMs achieve only 28-30%, revealing that structural information dominates, suggesting 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 on these datasets, we hypothesize that the improvement that GLMs achieve over LLMs can be attributed to their graph encoder. To test this hypothesis, we perform a detailed probing analysis in Sec. 3.3.

Zero-Shot Transfer. To further assess generalization, we evaluate zero-shot transfer where models are trained on one dataset and tested on another with different semantic domains. Following TEA-GLM's methodology, instructions include class labels from both source and target datasets. Table 2 shows that GLMs provide negligible gains over soft-prompted LLMs in zero-shot settings. On Arxiv → Cora, soft-prompted LLMs outperform

Model	Comp → Hist	Comp → Photo	Arxiv → Cora
Graph-Language Models			
TEA-GLM Llama3-8B	**57.94**±4.97	5.44±2.50	10.02±2.73
TEA-GLM Phi3-3.5B	18.16±5.70	3.71±0.16	5.80±4.73
G-Token (GSAGE) Llama3-8B	25.65±17.15	5.01±1.65	5.12±4.18
G-Token (GSAGE) Phi3-3.5B	19.62±5.07	2.52±0.22	3.76±4.32
Soft-Prompted LLMs			
Llama3-8B-SPT	43.56±12.50	**6.00**±1.19	16.57±1.15
Phi3-3.5B-SPT	32.85±13.79	2.85±0.42	**17.19**±0.68
Random Guessing	8.33	8.33	14.29

Table 2: **Zero-shot node classification accuracy across domain shifts.** Despite access to both modalities, GLMs provide no advantage over language-only baselines. Best results are in bold for each dataset.

Model	Computers	Photo	History
GNNs (Graph-only)			
GAT	91.65±0.42	94.66±0.37	84.63±0.55
GCN	**96.88**±0.31	**97.46**±0.29	84.91±0.48
GraphSAGE	85.96±0.64	91.67±0.58	84.27±0.61
Graph-Language Models			
TEA-GLM Llama3-8B	74.05±0.13	78.32±0.47	87.65±0.20
TEA-GLM Phi3-3.5B	69.75±0.71	72.38±1.76	77.16±0.28
G-Token (GSAGE) Llama3-8B	57.36±3.86	83.22±1.24	84.25±1.55
G-Token (GSAGE) Phi3-3.5B	54.41±4.43	80.05±0.59	82.15±2.42
GraphPrompter Llama3-8B	62.44±6.96	75.65±2.30	91.65±1.93
GraphPrompter Phi3-3.5B	59.65±3.59	73.24±0.68	92.28±1.15
Soft-Prompted LLMs (Language-only)			
Llama3-8B-SPT	80.20±2.20	86.48±4.88	**93.88**±0.12
Phi3-3.5B-SPT	78.27±2.86	84.30±2.53	92.06±1.03

Table 3: **Link prediction results on TAG benchmarks.** Our results show that GNNs achieve strong performance across all datasets and GLMs do not consistently outperform language-only soft-prompted baselines. Best results are in bold for each dataset.

all GLMs, achieving 16-17% compared to 3-10% for GLMs. These results are consistent with the patterns observed in Table 1, where unimodal baselines matched or exceeded GLM performance.

Link Prediction. For completeness, we also evaluate link prediction, an arguably easier binary classification task, on Computers, Photo, and History datasets. Table 3 reveals that soft-prompted LLMs achieve strong performance across all datasets. On History, Llama3-8B-SPT achieves 93.76%, outperforming GraphPrompter and G-Token (GSAGE). We observe the same trend on Computers and Photo, suggesting that graph structure may not be necessary for link prediction on these benchmarks.

3.3 Probing Graph Tokens

To quantify the contribution of the graph encoder on graph datasets, we perform a linear probing analysis. We take a fully trained and frozen GLM and pass the graph data through its graph encoder and the projector to extract the final node representations (*i.e.,* the graph tokens). A simple linear classifier is then trained on these representations to

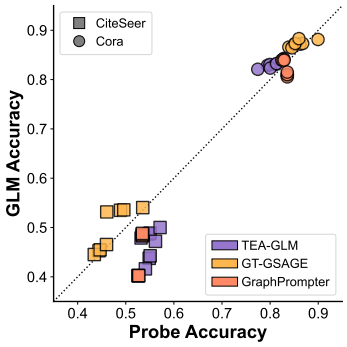


Figure 2: Linear probe accuracy on matches full GLM performance on Cora and CiteSeer (Pearson **r=0.96**), showing that graph encoders capture all task-relevant information while LLMs function as expensive decoders.

perform the node classification task. Mathematically, the probe prediction can be represented as:

$$\hat{y} = \arg\max_i \left(W \cdot \text{flatten}\left(M_P\left(M_g(\mathcal{G})\right)\right) + b\right)_i \tag{4}$$

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).

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

Takeaway #1

Our evaluation across diverse tasks (node classification and link prediction) and settings (supervised and zero-shot) reveals a critical inadequacy: widely-used benchmarks fail to assess multimodal integration and models achieve high accuracy using a single modality, indicating lack of datasets that require genuine integration of structural and semantic information.

4 CLEGR: Compositional Language-Graph Reasoning

Having demonstrated that current benchmarks allow unimodal shortcuts, we introduce CLEGR, a question-answering benchmark that, unlike existing evaluations, enforces genuine multimodal integration through compositional reasoning tasks over synthetic graphs with rich textual attributes (see Fig. 3).

4.1 Overview of CLEGR

CLEGR is a synthetic subway network comprising 500 graphs, 34 question templates, and approximately 32,000 questions. Building on CLEVR-Graph dataset (Mack and Jefferson, 2018), we extend it with a structured evaluation framework to systematically assess graph-language integration. The synthetic nature of the graphs, featuring fictional station names, metro lines, and connections, eliminates the possibility of LLMs exploiting pretraining knowledge, ensuring that performance reflects genuine reasoning capabilities rather than just recall. 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 node text alone (*e.g., "Is Station X part of a cycle?"*). Vanilla language models are insufficient as they lack access to structural relationships beyond immediate textual context (Yasunaga et al., 2021). This design prevents language-only shortcuts by demanding explicit graph traversal.

ii) Semantic Grounding: CLEGR's questions require natural language understanding of node and edge attributes (*e.g., "How many stations with Victorian architecture are adjacent to Station Y?"*). Traditional GNNs are inadequate as they operate on numerical representations without semantic comprehension. Critically, the question-answering format, natural language input to natural language output, *fundamentally prevents graph-only solutions*, as GNNs cannot be trained on this task format.

iii) Compositional Complexity: Tasks combine multiple reasoning steps that blend property lookup with logical inference (*e.g., "What is the shortest path between A and B avoiding dirty stations?"*). This creates challenges that benefit from integrating structural and semantic information throughout the reasoning process (Yang et al., 2018), making neither modality sufficient alone.

Concrete Example. Consider the question: *"How many stations are on the shortest path between Ashford and Belmont Station avoiding dirty stations?"* Models receive node and edge textual descriptions.

```
node_id, name, attributes
0, Thrak Bridge, disabled-access, large, Victorian
1, Phiptland, no disabled-access, small, Brutalist
...
36, Crount Lane, disabled-access, small, Gothic
source_id, target_id, attributes
0, 31, Red Line, Air-conditioned, Built in 2000s
...
26, 36, Blue Line, Not Air-conditioned, Built in 80s
```

The textual representation provides only local, pairwise connectivity, but computing shortest paths requires enumerating all possible routes, which is computationally intractable from text alone (Saparov et al., 2025). LLMs lack the algorithmic capabilities to perform graph traversal operations from textual descriptions (Agrawal et al., 2024). Conversely, GNNs face fundamental incompatibility: CLEGR requires natural language input→natural language output, but GNNs are designed for fixed output spaces (node and link prediction), not open-ended text generation. Even if adapted, GNNs would need semantic understanding of "dirty," "shortest path avoiding X", capabilities absent from numeric embeddings. This question necessitates both: graph structure for efficient structure-understanding via message passing and language-understanding to interpret the question constraint ("avoiding dirty stations") and generate the answer.

4.2 Benchmark Structure

Graph Generation. Each graph is a synthetic subway network composed of metro lines and stations with realistic, randomly generated attributes. On average, graphs contain $26.54_{\pm 5.41}$ nodes, $28.32_{\pm 6.37}$ edges, and $6.02_{\pm 1.23}$ distinct metro lines. Each station node is annotated with six different attributes and edge is associated with four distinct attributes.

Dataset Composition. CLEGR contains 500 synthetic subway graphs with 34 question templates spanning four reasoning types: Filtering (conditional selection), Aggregation (counting/comparison), Path Reasoning (multi-hop traversal), and Topology (structural analysis). After generating 2 questions per template for each graph and natural filtering to remove structurally invalid instances, the benchmark comprises approximately 32,248 questions across three structural scopes: node, edge, and subgraph levels. The dataset uses a 3:1:1 train/validation/test split (300/100/100 graphs). Questions require diverse answer types,

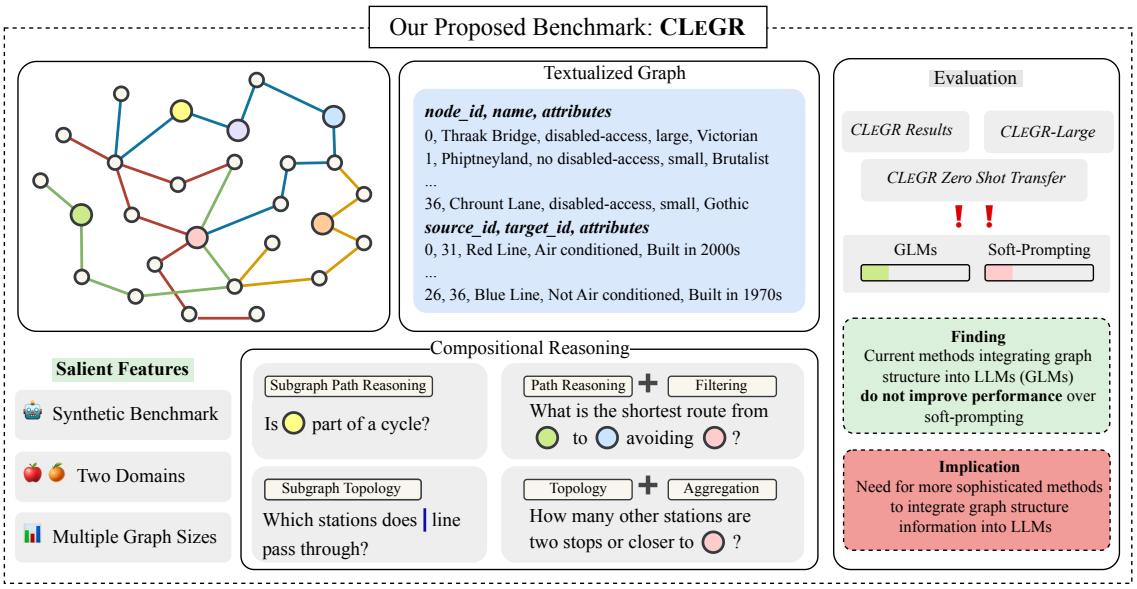


Figure 3: CLEGR addresses evaluation limitations through a multimodal approach that requires compositional reasoning over attributed synthetic graphs. It assesses model performance across node, edge, and subgraph levels, specifically targeting the integration of graph structure into language-based reasoning.

Boolean, Numeric, String, and List, necessitating flexible natural language generation rather than simple classification. Complete template definitions are in Table 8, 9 and 10.

5 Evaluating GLMs on CLEGR

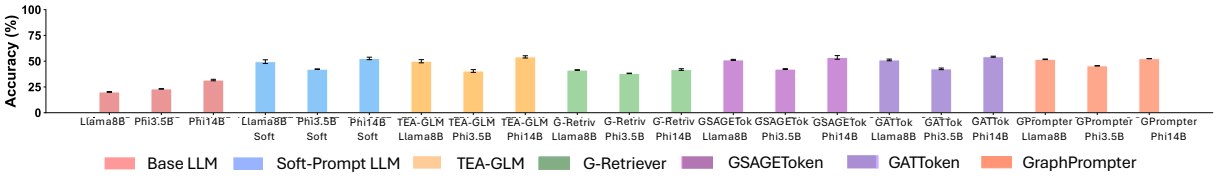
Equipped with a benchmark that necessitates the incorporation of graph structure and text semantics, we now evaluate GLMs on CLEGR. Since CLEGR is designed to require multimodal reasoning where structural encoders should provide an advantage, we investigate whether GLMs can exploit their architectural capabilities. Specifically, we answer the following questions: **RQ❸:** Does incorporating structural information into LLMs provide performance gains over soft-prompting LLMs on tasks requiring multimodal reasoning? **RQ❹:** Do GLMs provide better zero-shot generalization to other domains? **RQ❺:** How does GLM performance scale with increasing graph size?

5.1 Experimental Setup

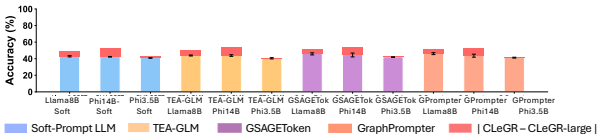
We evaluate several models to assess graph reasoning capabilities: the GLMs introduced in Sec. 3.1, G-Retriever (He et al., 2024b) that enhances performance through subgraph and textual retrieval, and LLM-only baselines including Phi4-14B to analyze scaling effects. Notably, G-Retriever and GraphToken were originally trained on question-answering tasks, aligning with CLEGR's QA formulation. All experiments use identical hardware configurations with consistent batch sizes and learning rates within model categories, employing greedy decoding and averaging results across five random seeds. Unlike prevalent node- and edge-level benchmarks, CLEGR is a graph-level task where questions may target any combination of nodes, edges, and subgraphs. We employ the pooling operation from Eq. 2 to pass the complete graph representation to all models. Performance is evaluated using accuracy, where a prediction is correct only when it precisely matches the ground truth. See Appendix D for more details on the experiment setup.

5.2 Results

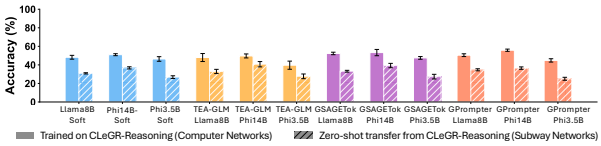
RQ❸: Multimodal reasoning effectiveness. Fig. 4a shows that despite access to structural encoders designed to aid multimodal reasoning, TEA-GLM, GraphToken, and GraphPrompter provide negligible performance gains over purely language-based soft-prompted baselines. Surprisingly, G-Retriever, which retrieves subgraphs intended to provide structural context, exhibits degraded performance. We hypothesize this results from incorrect subgraph retrieval and insufficient contextual coverage. The low and comparable performance between GLMs and unimodal baselines on a structurally grounded benchmark indicates that GLMs do not effectively exploit multimodal capabilities.



(a) CLEGR Results: GLMs fail to outperform soft-prompted baselines on reasoning tasks requiring structural understanding, revealing a reliance on surface-level language patterns over structural graph understanding.



(b) 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.



(c) 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.

Figure 4: Detailed Evaluation Results on CLEGR. (a) shows performance on CLEGR reasoning tasks, (b) shows the impact of increasing graph size, and (c) shows zero-shot transfer to a new semantic domain.

RQ❹: Zero-shot structural transfer. To evaluate whether structural encoding improves zero-shot transfer, we assess generalization from the subway domain in CLEGR to a structurally analogous Computer-Networks domain of CLEGR (see Appendix C.5) (Fig. 4c). Neither TEA-GLM nor GraphToken outperform soft-prompted LLMs. If GLMs captured transferable structural patterns, improved cross-domain performance would be expected; instead, results remain low and comparable to language-only baselines.

RQ❺: Scaling graph complexity. We examine whether increasing graph size reveals advantages for GLMs. We introduce CLEGR-Large, containing graphs approximately three times larger (Table 7). As structural complexity increases, both GLMs and soft-prompted baselines experience nearly identical performance degradation. This decline indicates that even under large structural demands, GLMs do not leverage graph encoders more effectively, corroborating findings from RQ3-RQ4.

Takeaway #2

On CLEGR, a benchmark designed to require multimodal integration, GLMs achieve low performance, comparable to unimodal soft-prompted baselines. This parity persists as graph size increases and during zero-shot transfer, indicating that GLMs do not effectively exploit their structural encoders even when tasks demand it. Under current formulations, the multimodal capabilities of GLMs remain unrealized.

6 Conclusion

In this work, we demonstrate that prevailing graph-language evaluation methodologies often fail to test true multimodal integration: existing benchmarks, regardless of the underlying graph or task, permit unimodal baselines (relying only on language or only on structure) to match the performance of graph-language models (GLMs). This exposes a fundamental limitation in current multimodal evaluation paradigms, as they do not rigorously assess a model's ability to reason jointly over both graph structure and language semantics. To directly address this gap, we introduce CLEGR, a multimodal benchmark centered on graph-level question answering, requiring compositional reasoning that intertwines structural and semantic cues. Our comprehensive evaluations show that, even when multimodal architectures have explicit access to both modalities, GLMs do not outperform unimodal, language-only baselines on CLEGR even when the demand for integrated reasoning increases through larger graphs and domain transfer scenarios. This persistent parity indicates that current GLM architectures and integration strategies are not yet effective at leveraging their multimodal capabilities.

CLEGR thus serves as a focused probe of multimodal reasoning, illuminating the challenges that remain and providing the community with a testbed to drive advances in truly multimodal graph-language models. As GLMs evolve, we plan to continually refine CLEGR to track, challenge, and benchmark progress in this area.

7 Limitations

In this paper, we propose CLEGR, a multimodal question-answering benchmark designed to assess genuine graph-language integration. To facilitate controlled evaluation and eliminate pretraining contamination, we utilize synthetic subway networks as the primary source of graph-structured information. We believe that extending this approach to more real-world applicability from diverse domains represents a promising direction for future improvements. Besides, while CLEGR demonstrates zero-shot transfer to computer networks, evaluating generalization across a broader range of domains with different structural and semantic properties would further validate our findings.

Additionally, CLEGR tries to integrate four core reasoning types (filtering, aggregation, path reasoning, and topology analysis), which represent fundamental graph reasoning operations. Expanding the benchmark to include additional reasoning capabilities such as graph generation, planning under uncertainty, or temporal reasoning over dynamic graphs is a direction worth exploring. Our evaluation primarily employs exact match accuracy as the metric, which provides objective assessment but may not capture semantically equivalent answers phrased differently. Incorporating semantic similarity measures (*e.g.,* BLEU, BERTScore) could provide complementary insights, particularly for list-based and string-based responses.

Due to computational constraints, we evaluate models up to the 14B parameter scale. While our consistent findings across multiple model sizes (Llama3-8B, Phi3-3.5B, Phi4-14B) and architectures suggest our conclusions are robust, evaluating larger models (*e.g.,* 70B+ parameters) would provide additional validation. Furthermore, we focus on prominent GLM architectures that represent recent state-of-the-art approaches. As the field rapidly evolves, evaluating emerging architectures with CLEGR will help track progress in multimodal graph-language reasoning.

References

Palaash Agrawal, Shavak Vasania, and Cheston Tan. 2024. Can llms perform structured graph reasoning? *Preprint*, arXiv:2402.01805.

Arvindh Arun, Sumit Kumar, Mojtaba Nayyeri, Bo Xiong, Ponnurangam Kumaraguru, Antonio Vergari, and Steffen Staab. 2025. Semma: A semantic aware knowledge graph foundation model. *Preprint*, arXiv:2505.20422.

Maya Bechler-Speicher, Ben Finkelshtein, Fabrizio Frasca, Luis Müller, Jan Tönshoff, Antoine Siraudin, Viktor Zaverkin, Michael M. Bronstein, Mathias Niepert, Bryan Perozzi, Mikhail Galkin, and Christopher Morris. 2025. Position: Graph learning will lose relevance due to poor benchmarks. *Preprint*, arXiv:2502.14546.

Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, and Jiliang Tang. 2024. Exploring the potential of large language models (llms) in learning on graphs. *Preprint*, arXiv:2307.03393.

Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S Dhillon. 2022. Node feature extraction by self-supervised multi-scale neighborhood prediction. *Preprint*, arXiv:2111.00064.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. 2023. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Preprint*, arXiv:2305.06500.

William L. Hamilton, Rex Ying, and Jure Leskovec. 2018. Inductive representation learning on large graphs. *Preprint*, arXiv:1706.02216.

Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. 2024a. Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning. *Preprint*, arXiv:2305.19523.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024b. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Preprint*, arXiv:2402.07630.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2021. Open graph benchmark: Datasets for machine learning on graphs. *Preprint*, arXiv:2005.00687.

Bowen Jin, Wentao Zhang, Yu Zhang, Yu Meng, Xinyang Zhang, Qi Zhu, and Jiawei Han. 2023. Patton: Language model pretraining on text-rich networks. *Preprint*, arXiv:2305.12268.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *Preprint*, arXiv:1609.02907.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *Preprint*, arXiv:2104.08691.

Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. 2024a. A survey of graph meets large language model: Progress and future directions. *Preprint*, arXiv:2311.12399.

Yuhan Li, Peisong Wang, Xiao Zhu, Aochuan Chen, Haiyun Jiang, Deng Cai, Victor Wai Kin Chan, and Jia Li. 2024b. Glbench: A comprehensive benchmark for graph with large language models. *Preprint*, arXiv:2407.07457.

Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2024a. One for all: Towards training one graph model for all classification tasks. *Preprint*, arXiv:2310.00149.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *Preprint*, arXiv:2304.08485.

Zheyuan Liu, Xiaoxin He, Yijun Tian, and Nitesh V. Chawla. 2024b. Can we soft prompt llms for graph learning tasks? In *Companion Proceedings of the ACM Web Conference 2024*, WWW '24, page 481–484. ACM.

Yuankai Luo, Lei Shi, and Xiao-Ming Wu. 2024. Classic gnns are strong baselines: Reassessing gnns for node classification. *Preprint*, arXiv:2406.08993.

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

Andrew Kachites McCallum and 1 others. 2000. Automating the construction of internet portals with machine learning. In *AAAI Spring Symposium on Mining Answers from Textual Data*.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let your graph do the talking: Encoding structured data for llms. *Preprint*, arXiv:2402.05862.

Swarnadeep Saha, Prateek Yadav, Lisa Bauer, and Mohit Bansal. 2021. Explagraphs: An explanation graph generation task for structured commonsense reasoning. *Preprint*, arXiv:2104.07644.

Abulhair Saparov, Srushti Pawar, Shreyas Pimpalgaonkar, Nitish Joshi, Richard Yuanzhe Pang, Vishakh Padmakumar, Seyed Mehran Kazemi, Najoung Kim, and He He. 2025. Transformers struggle to learn to search. *Preprint*, arXiv:2412.04703.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine*, 29(3):93–106.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Pitfalls of graph neural network evaluation. *Preprint*, arXiv:1811.05868.

Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. 2025. Grapharena: Evaluating and exploring large language models on graph computation. *Preprint*, arXiv:2407.00379.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. *Preprint*, arXiv:1710.10903.

Duo Wang, Yuan Zuo, Fengzhi Li, and Junjie Wu. 2024. Llms as zero-shot graph learners: Alignment of gnn representations with llm token embeddings. *Preprint*, arXiv:2408.14512.

Qiming Wu, Zichen Chen, Will Corcoran, Misha Sra, and Ambuj K. Singh. 2025. Grapheval36k: Benchmarking coding and reasoning capabilities of large language models on graph datasets. *Preprint*, arXiv:2406.16176.

Cheng Xu, Shuhao Guan, Derek Greene, and M-Tahar Kechadi. 2024a. Benchmark data contamination of large language models: A survey. *Preprint*, arXiv:2406.04244.

Yuhao Xu, Xinqi Liu, Keyu Duan, Yi Fang, Yu-Neng Chuang, Daochen Zha, and Qiaoyu Tan. 2024b. Graphfm: A comprehensive benchmark for graph foundation model. *Preprint*, arXiv:2406.08310.

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. 2023. A comprehensive study on text-attributed graphs: Benchmarking and rethinking. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhuo Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. 2023a. Graphformers: Gnn-nested transformers for representation learning on textual graph. *Preprint*, arXiv:2105.02605.

Zhengyuan Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Chung-Ching Lin, Zicheng Liu, and Lijuan Wang. 2023b. The dawn of lmms: Preliminary explorations with gpt-4v(ision). *Preprint*, arXiv:2309.17421.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *Preprint*, arXiv:1603.08861.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *Preprint*, arXiv:1809.09600.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. QA-GNN: Reasoning with language models and knowledge graphs for question answering. In *Proceedings of the 2021 Conference of the North American Chapter*

of the Association for Computational Linguistics: Human Language Technologies, pages 535–546, Online. Association for Computational Linguistics.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. 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)*, pages 201–206, Berlin, Germany. Association for Computational Linguistics.

Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. 2025. Gracore: Benchmarking graph comprehension and complex reasoning in large language models. *Preprint*, arXiv:2407.02936.

Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. 2023. Learning on large-scale text-attributed graphs via variational inference. *Preprint*, arXiv:2210.14709.

A Appendix

A.1 TEA-GLM

The TEA-GLM (Token Embedding-Aligned Graph Language Model) (Wang et al., 2024) methodology is a novel framework designed to enhance zero-shot graph machine learning by integrating GNNs with instruction-fine-tuned 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.

A.2 G-Retriever

G-Retriever (He et al., 2024b) is a framework for question answering on textual graphs, integrating Retrieval-Augmented Generation (RAG) with GNNs and LLMs to enable users to "chat with their graph." It addresses complex queries on real-world textual graphs by first developing a 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.

A.3 Graph-Token (G-Token)

In the Graph-Token methodology (Perozzi et al., 2024) 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 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.

A.4 GraphPrompter

GraphPrompter (Liu et al., 2024b) is designed for node classification tasks by integrating structural and textual information through graph token prompting. For each target node, a k-hop subgraph is extracted and encoded using a GNN to capture local structural context, followed by a projection layer that maps the graph representation into the LLM embedding space. In parallel, the textual attributes associated with each node are processed by a text embedder to obtain semantic representations. The resulting graph and text embeddings are concatenated to form a graph token soft prompt, which is prepended to the input of a frozen LLM to guide downstream graph learning tasks.

A.5 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.

A.5.1 Graph Convolutional Networks (GCN)

GCNs (Kipf and Welling, 2017) update node embeddings using a normalized sum over neighboring features:

$$h_v^{(l)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\hat{d}_v \hat{d}_u}} h_u^{(l-1)} W^{(l)}\right) \quad (5)$$

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.

A.5.2 GraphSAGE

GraphSAGE (Hamilton et al., 2018) aggregates neighborhood information using a fixed aggregation function (e.g., mean):

$$h_v^{(l)} = \sigma\left(h_v^{(l-1)} W_1^{(l)} + \left(\text{mean}_{u \in \mathcal{N}(v)} h_u^{(l-1)}\right) W_2^{(l)}\right) \tag{6}$$

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

A.5.3 Graph Attention Networks (GAT)

GATs (Veličković et al., 2018) apply masked self-attention over neighbors. The attention coefficient between nodes v and u is:

$$e_{vu}^{(l)} = \text{LeakyReLU}\left(a^\top \left[W h_v^{(l-1)} \parallel W h_u^{(l-1)}\right]\right) \tag{7}$$

$$\alpha_{vu}^{(l)} = \frac{\exp(e_{vu}^{(l)})}{\sum_{r \in \mathcal{N}(v)} \exp(e_{vr}^{(l)})} \tag{8}$$

and the node update is:

$$h_v^{(l)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(l)} W h_u^{(l-1)}\right) \tag{9}$$

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.

A.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 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 (Hamilton et al., 2018) 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 Wang et al. (2024) (Table 5-6). For CiteSeer, we use 500 PCA-projected features due to its higher original feature dimensionality, which may contribute to the lower performance observed on this dataset.

A.7 GNN Training

We follow the works of (Luo et al., 2024) for GNN baselines; we follow the same training setup, adopting their recommended hyperparameters and optimization configurations for each dataset. (Table 4)

A.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 (Luo et al., 2024), 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 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.

A.9 Computing Infrastructure

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

Table 4: 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 5: 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	GraphSAGE	Graph	768	1024	1024	1024	3	0.5

B CLEGR Benchmark Construction

B.1 CLEGR Subway Networks

B.1.1 Overview

The CLEGR dataset is a graph-based question answering benchmark built on synthetic subway networks extending on the work by (Mack and Jefferson, 2018). Each graph represents a fictional subway system with randomly generated nodes and attributes.

B.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`
- `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.

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	GAT	Graph	768	1024	1024	1024	3	0.5

Table 6: GLM hyperparameter settings for using the GAT models. All configurations use consistent hidden/project dimensions and dropout.

B.2 Graph Statistics

Table 7 presents the graph statistics for CLEGR Subway Networks.

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

Table 7: Graph statistics for CLEGR Subway Networks with different graph sizes.

B.3 Dataset Generation Pipeline

B.3.1 Questions

The dataset contains 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.

B.3.2 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 Tables 8, 9,10).

C 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.

C.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.

C.2 Feature Schema

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

C.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'`).

C.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').

C.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; and **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?").

C.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'`.

C.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.

C.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}.

C.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.

C.4.4 Data Tuples

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

C.4.5 Dataset Statistics

The total default number of graphs generated is 500. The CLEGR dataset contains 32,248 questions (after natural filtration of invalid questions from a total of 34,000 questions), with 68 reasoning-based questions per graph. The Train, Validation and Test sets contain 300, 100 and 100 graphs (and their corresponding questions) respectively.

C.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.

C.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 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.

C.6 Feature Schema

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

C.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.

C.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.

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

Table 8: CLEGR Question Templates for the Subway Networks (Scope: Node)

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 Question Templates for the Subway Networks (Scope: Edge)

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

Table 10: CLEGR Question Templates for the Subway Networks (Scope: Sub-graph)

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

C.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.

C.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}.

C.7 Graph Statistics

Table 11 shows the graph statistics for CLEGR Computer Networks, which is also similar to that of CLEGR Subway Networks.

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

Table 11: Graph statistics for CLEGR Computer Networks with different graph sizes.

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

D CLEGR Evaluation

D.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.

D.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.

E Prompt Templates

E.1 Example Prompts for Node Classification

E.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 separated Category List}. \nAnswer: [/INST] {Label} [/s]

E.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]

E.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]

E.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]

E.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]

E.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]

E.1.7 CLEGR

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 (He et al., 2024b) 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.

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:

F CLEGR (Zero Shot Transfer)

This section presents the accuracy results of GLMs on different transfer learning scenarios across domains. Table 13 shows zero-shot domain transfer from CLEGR Subway Networks to CLEGR Computer Networks, while Table 14 presents in-domain performance on CLEGR Computer Networks.

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

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

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 13: Zero-shot Transfer Accuracy: Trained on CLEGR Subway Networks, Tested on CLEGR Computer Networks

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 14: In-Domain Accuracy: Trained and Tested on CLEGR Computer Networks