Graph Representation Augmented Inference for Language Models

Ethan Reinhart

June 13, 2025

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities in natural language understanding and generation. However, their performance on structured problems involving graphs remains limited unless specialized conditioning techniques are applied. We propose an expansion of previous frameworks to handle challenging graph problems in GRAIL (Graph Representation Augmented Inference for Language Models). GRAIL highlights the efficiency of prefix injection in conditioning frozen LLMs for better structured reasoning on graph problems. We experiment with projecting learned latent vectors to the embedding input space of an LLM. For the scope of this project, we evaluate performance on synthetic instances of TSP. We observe notable accuracy improvements on small graphs and significant reductions in hallucination frequency.

1 Introduction

Large-scale usage of LLMs (Large Language Models) has highlighted their key abilities process and analyze textual data. While broadly used, a notable concern of these models is their tendency to "hallucinate" by providing results that are untrue or uncorrellated with their prompt. This problem is innate to the architecture of LLMs and has been mitigated to an extent by reasoning models through the use of CoT (Chain of Thought), sampling different answer trajectories and picking the most likely, and MoE (Mixture of Experts), where sub-networks are trained on specific task and the inference pipeline traverses only through relevant sub-networks.

Another concern is their reliance on observed data, leading to the issue of "freshness", where new information that the model hasn't observed cannot be processed accurately. Techniques to remedy this have included providing additional recent contextual information to enrich the prompts and enhance model reasoning. Their success over textual representations has driven recent consideration into the reasoning ability over structured data.

The most commonly used method of overcoming these concerns is by implementing a RAG (Retrieval Augmented Generation) pre-processing framework to dynamically query and append relevant information to the prompt prior to model inference. RAG models will often retrieve structured information whose spatial connotations prove difficult in LLM reasoning.

Structured data is integral in real-world applications. Social networks, e-commerce interactions, and spatial relativity are all important observations that come innately to human perceptions, yet LLMs struggle severely. This lack of "visualization" in 2D and 3D spaces stunts LLMs from being able to properly reason distances, connectivity, and scale of graph-based data. The majority of past efforts to handle these problems is through textual enrichment of prompts to provide supplementary information about graph architecture and connectivity. Some, such as GraphArena [12], append information regarding node connectivity, while others, such as GraphWiz [4] seek to improve comprehensibility of reasoning over these tasks. While these approaches have shown improvements, their input scales exponentially with graph size.

While past work deals predominantly with textual enrichment for encoding graph structure, Let Your Graph Do the Talking: Encoding Structured Data for LLMs (GraphToken) [11] and Talk Like a Graph [5] handle structurally encoded tokenization of graphs to handle simple graph-related questions. GraphToken analyzes the efficiency of various GNN (Graph Neural Network) architectures to encode graphs into vectors that are projected onto the LLM input space to provide "black-box" enrichment of prompts. GraphToken shows promising results on LLM question and answering on simple existence, connectivity, and structural tasks such as edge and node existence, degree calculations, cycle checking, triangle counting, shortest path, among others, this fails to highlight how these methods can generalize to problems of non-linear order. This is essential in practical use as advanced reasoning over structured data is necessarily non-polynomial.

In this paper, we propose GRAIL, an expansion on GraphToken that analyzes the efficiency of GNN-encoded prefix injection on advanced graph problems. Due to computational and time constraints, we limit the observations of this paper to the Traveling Salesman problem. This problem is known to be NP-Complete, which we argue is a good candidate as proper reasoning over this problem would imply the capability of proper reasoning over all NP problems due to the nature of NP-completeness.

Our Contributions

1. GRAIL, an extension of GraphToken using different encoding methods while maintaining parameter efficiency
2. Analysis of efficiency over NP-complete problems, highlighting potential improvement on difficult structured problems
3. A proposition for future work

2 Background and Related Work

We introduce the related work in LLMs, prompting methods, GNNs, graph encoders, and graph models combined with LLMs.

2.1 The Traveling Salesman Problem (TSP)

TSP is a classical combinatorial optimization problem where a salesman must visit n cities exactly once and return to the origin while minimizing travel cost. The problem is NP-hard, and solving it via neural methods typically requires careful conditioning to provide structural context. In this paper, we use the general Euclidean TSP problem, which consists of a complete. Therefore there are $\frac{(n-1)!}{2}$ Hamiltonian cycles to observe to solve the TSP. We argue that this is a good candidate for evaluating the performance of our outlined techniques on difficult structured reasoning problems for LLMs. This is due to the nature of NP-completeness. Euclidean TSP is known to be NP-Hard therefore it is at least as hard as the most challenging problem in NP. The runtime efficiency of a solution algorithm for an NP-hard problem can be translated to solve any problem in NP with the same run-time efficiency, up to a polynomial time computation. Further, performance on other NP-hard problems should be relatively similar due to their similar computational complexity and their ability to generalize to NP. With this, we determine TSP as a good candidate to consider generalization to less complex, structured problems.

2.2 LLMs

Language models (LMs) are probabilistic models that assign probabilities to sequences of words by breaking the probability of a sequence into the product of the probabilities of the next tokens given the previous ones. LLMs extend this through unsupervised training on massive amounts of textual data. Generalization of an LLM is essential for wide-scale consumer use, yet increased generalization comes at the expense of a lack of specification. A recent area of interest for adapting pre-trained LLMs to perform better on specific tasks is Parameter Efficient Fine-Tuning (PEFT). PEFT allows for freezing of the majority, or all, of the LLM parameters and only updating gradients on encoded input or a small subset of the model's parameters.

PEFT Methods

1. Adapter-Based
   This freezes the entire LLM and adds new trainable parameters to various parts of the model. Current work analyzes locations for adding parameters to improve reasoning [7].
2. LoRa
   This freezes the majority of the LLM but keeps gradient updates on for a subset of model weights. The resulting model is identical in architecture to the original, but the subset of updated weights is distinct from the original [8].
3. Prefix-Tuning
   This freezes the entire LLM and trains a separate encoder to append projected prefixes to the prompt. Most notably, this maintains the original LLMs accuracy on all benchmarks but has been shown to improve accuracy over the encoder's domain [10]. Other architectures also exist in this category as shown in Standing on the Shoulders of Frozen LLMs [9].

Parameter efficiency stems from the ability to only compute gradients and update weights on only a minute subset of total model weights, in Adapter-Based and LoRa methods, or on a completely separate encoder, in Prefix-Tuning methods.
GRAIL falls under that category of Prefix-Tuning methods, however, we argue that this could easily be applied to other fine-tuning methods. Still, we argue that this method is optimal for solving structural tasks in inference as it improves accuracy on this subdomain while maintaining accuracy on all other benchmarks.

2.3 Graph-Structured Conditioning of LLMs

Standard LLMs are not inherently designed to process structured data such as graphs. Recent efforts like **GraphArena** [**grapharena2023**] and **GraphWiz** [**graphwiz2023**] propose training protocols and benchmarks to equip LLMs with the ability to handle structured domains by leveraging graph encoders. These efforts highlight the insufficiency of raw text for encoding graph structure. *Let Your Graph Do the Talking* [**letgraph2024**] introduces a modality-bridging approach to encode graphs as embedded prefixes that allows LLMs to reason about graph structure and connectivity. While these results are promising, the scope of problems addressed is limited to granular cycle calculations, node and edge existence, and shortest path calculations. These results show promise for encoding graph structure and connectivity into embedded vectors but fail to highlight how these could be applied to structured, real-world tasks. In this paper, we seek to bri

3 Data

As mentioned, methods for encoding graph structure are predominantly based on textual encoding of graph attributes or GNN encoding and projection to LLM input. Analysis of our methods are predominantly focused on the latter, yet we integrate both to allow more advanced reasoning.

3.1 Graph Generation

Euclidean TSPs are complete graphs that can be projected to Euclidean Space to better represent distances between nodes. With this, our graph generation method reverses this process by randomly generating points within a 100x100 unit square and fully connecting these points in the graph to generate a complete Euclidean graph. For this report, we segment graphs into 'easy' and 'hard', as in GraphArena, where easy and hard graphs contain node counts from 4 to 9 and 10 to 15, respectively. We recognize these node counts are relatively small in scope, as we were limited by computational ability and memory. Exact TSPs were calculated using an exact solver, namely the $fast - tsp$ Python package, which remains feasible for such node counts.

3.2 Textual Input

We utilize the textual input framework provided in GraphArena as our prompt. This generates text prompt inputs in the following format:

Model Prompt Input

You are required to solve the Traveling Salesman Problem for an undirected flight route network. Your objective is to determine the shortest possible route that visits each of the listed airports exactly once and returns to the starting point.
Problem to Solve
- Airports to visit: A0, A1, A2, A3. ...
- Travel distances (in kilometers) between each pair of airports: A0 to A1: 40 A0 to A2: 57 A0 to A3: 91 ...
Please calculate the shortest tour and format your answer as follows: [Airport A, Airport B, ..., Airport A]

While this textual representation doesn't yield significant further reasoning ability, we argue that this is essential in providing the LLM with the data required to be able to solve the problem, as shown in GraphArena.

4 Methods

We use a selection of three encoders that are capable of different encoding methods. These were selected as they have architecturally different methods of conserving graph structures, node features, and connectivity. We use node degree as the only node feature in our model input. Further, we use a Top-K pooling variant that incorporates attention to produce K learned vectors representing the original graph. As priorly mentioned, during training, only the encoders weights are updated and the LLM remains frozen.

4.1 Pooling

Pooling distills a variable-size graph into a compact latent representation. Its *expressiveness*, the degree to which it preserves the distinguishing power of the upstream message–passing (MP) layers, is critical whenever all structural and feature information must survive downstream reasoning. *The expressive power of pooling in Graph Neural Networks* proposes that DiffPool pooling is proficient at preserving expressiveness [2].

DiffPool [15] learns a soft assignment matrix $S \in \mathbb{R}^{N \times K}$ by row–softmaxing GNN-generated logits, so each node chooses a probability distribution over K clusters. The pooled feature matrix is $Z^\top S$ while the adjacency is coarsened to $S^\top A S$, enabling deep hierarchical encoders. DiffPool is fully differentiable and powerful but carries a quadratic memory overhead in K and mandates graph rewiring at every layer.

Graph Multiset Transformer (GMT) [1] dispenses with graph coarsening and instead introduces K learnable *query* tokens that attend to all node embeddings via multi-head dot-product attention. Each query produces one pooled vector, yielding a fixed-length set $\{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$ regardless of graph size. GMT is highly expressive but incurs the $\mathcal{O}(NK)$ cost of full attention and adds several projection matrices per head.

Soft Top-K Attention Pooling (ours). We take the benefits of both methods while remaining lightweight: (i) a single linear layer produces a K-dimensional logit vector $\mathbf{s}_i$ for every node; (ii) a graph-local *column-wise* softmax converts logits to attention weights w_{ik}, normalising across nodes for each cluster k; (iii) cluster embeddings are $\mathbf{c}_k = \sum_i w_{ik} \mathbf{h}_i$, giving exactly K vectors with no graph rewiring. All nodes participate—gradients stay dense—yet high-score nodes dominate each cluster, mimicking a soft topk without explicit pruning. Compared with DiffPool we avoid the quadratic assignment matrix and pooled adjacency; compared with GMT we eliminate heavy attention projections and reduce complexity to $\mathcal{O}(NK)$ matrix–vector operations. The result is a parameter-efficient, fully differentiable pooling operator tailored to scenarios (e.g., LLM prompting) that require a strict token budget and fast inference.

4.2 Encoders

We wrap three widely–adopted GNN backbones in the common degree–embedding, batch-norm, and soft Top-K pooling framework described above. Their diversity lets us probe the trade-off between *inductive generalisation* (handling unseen nodes), *expressive power* (distinguishing graph structures), and *computational cost.*

1. **GraphSAGE–Mean** [6].
   Each layer computes

$$h_i^{(l+1)} \;=\; \sigma\Big(W^{(l)} \,[\, h_i^{(l)} \,\|\, \operatorname*{mean}_{j \in \mathcal{N}(i)} h_j^{(l)}]\Big),$$

   where "$\|$" denotes concatenation. The *mean* aggregator is permutation-invariant, needs no attention coefficients, and runs in time $\mathcal{O}(d\,|E|)$, making it ideal for million-node graphs such as Reddit (232 K nodes) and the PPI benchmark. Crucially, parameters are tied across nodes, so the model *generalises inductively* to nodes—and even entire subgraphs—never seen during training.

2. **GAT (Graph Attention Network)** [13].
   GAT augments the message-passing kernel with multi-head additive self-attention:

$$\alpha_{ij}^{(l,h)} \;=\; \mathrm{softmax}_{j \in \mathcal{N}(i)}\big(a^\top \big[W_h^{(l)} h_i^{(l)} \,\|\, W_h^{(l)} h_j^{(l)}\big]\big),$$

$$h_i^{(l+1)} = \Vert_{h=1}^{H} \sigma\Big(\sum_j \alpha_{ij}^{(l,h)} W_h^{(l)} h_j^{(l)}\Big) \Vert,$$

where H is the number of heads. Attention allows the model to weigh neighbours differently, capturing heterophily or hub dominance without expensive eigendecompositions. GAT set a new state-of-the-art on citation networks (Cora, CiteSeer, PubMed) with *few* parameters, yet remains linear in the number of edges.

3. **GIN (Graph Isomorphism Network)** [14].
GIN replaces mean/attention with a learnable *sum* aggregator proven to match the 1-Weisfeiler–Lehman (1-WL) test's discriminative power:

$$h_i^{(l+1)} = \mathrm{MLP}^{(l)}\Big(\big(1+\epsilon^{(l)}\big)\, h_i^{(l)} + \sum_{j\in\mathcal{N}(i)} h_j^{(l)}\Big),$$

where $\epsilon^{(l)}$ is either fixed or learned. By sandwiching the sum inside a two-layer MLP, GIN acts as a universal approximator over multisets, achieving top performance on graph-classification benchmarks that hinge on subtle structural differences (e.g. MUTAG, NCI1). The trade-off is a heavier parameter count and no built-in attention sparsity.

Why three encoders? GraphSAGE offers the lightest inductive alternative, GAT injects neighbour-wise importance, and GIN maximises structural expressiveness. Coupling each with our *soft Top-K attention pooling* provides a controlled test-bed: the downstream LLM sees a *fixed* token budget while the upstream GNN varies in capacity, letting us isolate how backbone expressiveness translates to performance in parameter-efficient graph-to-text pipelines.

4.3 Features

The features used for all encoders were kept constant. Each node vector was represented by a 32-dimensional vector where the node was indexed by its degree. The hidden dimension at each layer was set to 128 dimensions, 3 layers were used, and 8 latent vectors were projected into the LLM input space. A dropout of 0.2 was used across all encoders to reduce overfitting, and only one attention head was used in each.

4.4 Base Language Models

To demonstrate that our graph–prompt adapter is *model-agnostic*, we plug it into two open-source causal language models at the 100–200M parameter scale. Both are decoder-only Transformers but differ in positional encoding, activation functions, training corpus, and tokenizer—letting us gauge how such architectural choices interact with our graph encoders.

- **facebook/opt-125m** [16][1]
Architecture. 12 Transformer blocks, hidden size $d_{\text{model}} = 768$, 12 attention heads, feed-forward dimension 3072, learned absolute positional embeddings, GELU nonlinearities. *Training.* 180 B English tokens drawn from CommonCrawl, Books, Wikipedia, and Reddit; context window 2048; Adam with linear decay; standard causal-LM objective. *Tokenizer.* 50 480-slot GPT-2 BPE. *Why OPT?* It is Meta's reproduction of GPT-2/3 with careful alignment to the original hyperparameters, providing a strong *baseline decoder* that many PEFT papers adopt.
- **EleutherAI/pythia-160m**[3][2]

[1]Released under a non-commercial CC-BY-NC licence.

[2]Apache-2.0 licence with full training logs and checkpoints.

Architecture. 12 layers, $d_{\text{model}} = 768$, 12 heads, **Ro**tary **P**ositional **E**mbeddings (RoPE), SwiGLU feed-forward blocks, the GPT-NeoX recipe scaled to 160 M parameters. *Training.* 300 B tokens of The Pile v1 (22 heterogeneous sub-corpora including academic PDFs and code); cosine LR schedule; context 2048. *Tokenizer.* NeoX GPT-Neo-compatible BPE with 50 432 merges. *Why Pythia?* It serves as a *transparency-first* baseline: every checkpoint is accompanied by full training-curve artifacts, enabling controlled ablations and reproducibility checks.

Rationale for scale. Models in the 100–200 M range (i) fit on a single consumer GPU for both PEFT and inference, (ii) expose meaningful architectural differences without overwhelming compute cost, and (iii) act as a lower bound for extrapolating to billion-parameter backbones. Because our pooling layer produces a *fixed K-token prompt*, the method transfers unchanged to any larger decoder-only or encoder–decoder LLM; we simply replace the base model's token embedding table with the graph-specific soft tokens learned by our adapter.

5 Results

We evaluate our defined metrics on the two LLM backbones and three encoders previously described, as well as with no encoder as a baseline.[3]

5.1 Metrics

We define the following three metrics:

- **Accuracy**
  This is defined as standard raw accuracy, when the predicted token sequence is equal to exactly that of the solution.
- **Feasibility**
  This is defined as the model that produces a valid Hamiltonian path. Considering all paths without duplicates are Hamiltonian paths in a complete graph, the model needs only to produce a sequence whose first token is the same as its last and does not contain any duplicates other than the start and end node.
- **Hallucination**
  By definition, this is the complement of feasibility. We enumerate different observed methods of hallucination and can infer learning based on the type of hallucination.

  – *Wrong Format*
    This is defined to be the event that the model did not properly format its solution sequence. This stems from the model not wrapping its solution in brackets, outputting text within its sequence or any other reason not encompassed by the following three events.

  – *Wrong Length*
    This is defined to be the event that the model produces a sequence that is properly formatted but is longer or shorter than the actual solution. This solution was properly wrapped in brackets, yet did not match the solution size.

  – *Duplicate Nodes*
    This is defined to be the event that the model produces a sequence that is the same length as the solution and is properly formatted, but contains more than one duplicate (the source and terminal node are duplicates by default).

  – *Made-Up Node*
    This is defined to be the event that the model produces a sequence that is the

[3]Full per-instance metrics and code are provided in the supplementary repository. GitHub

same length as the solution and is properly formatted, but contains a node that does not appear in the problem.

5.2 Accuracy and Feasibility

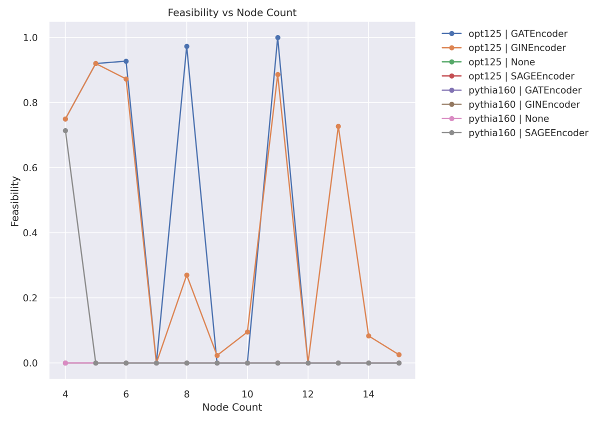

Figure 1: Feasibility vs. Node Count

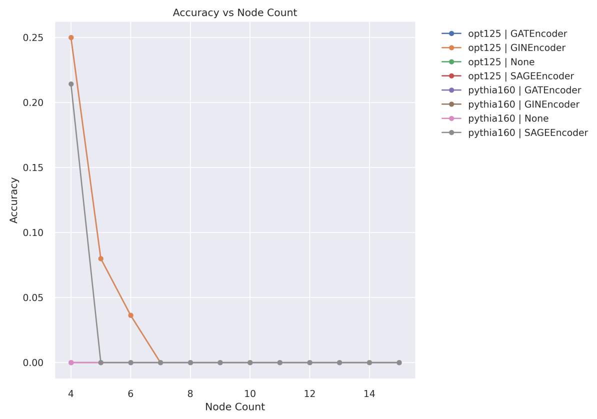

Figure 2: Accuracy vs. Node Count

5.2.1 Node-count sensitivity

Figure 2 plots accuracy as a function of $|V|$. We observe that inference on $|V| \geq 7$ produces no accurate solutions. This cliff likely coincides with the prompt length crossing the LLM's attention span after replacement by our K=8 prefix tokens.
Figure 1 plots feasibility as a function of $|V|$. More interestingly, we see varying output of feasible solutions concerning node count, with the highest feasibility occurring at $|V| = 11$.

5.2.2 Overall trends

- **Small graphs are solvable.**
  All non-zero accuracies occur at $4 \leq |V| \leq 5$. Performance collapses once the prompt must encode > 6 nodes, implying that the fixed-token budget is saturated.
- **Encoders beat the "raw prompt" baseline.**
  Both LLMs never produce a single feasible solution when not coupled with an encoder.
- **OPT-125M > Pythia-160M**
  With identical prompts, OPT attains up to **6.1%** exact accuracy on small graphs and **65.6%** feasibility (GAT/GIN), while Pythia's best exact match is 3.6% (SAGE) on small graphs and its feasibility peaks at 53.4%.

5.2.3 Take-Aways

GAT and GIN give similar averages on OPT, but GAT achieves a 100 % feasibility at $|V|$=11 (albeit with zero exact matches), whereas GIN reaches a higher exact accuracy at $|V|$=4. GraphSAGE shines on Pythia for the simplest graphs but fails to scale.

5.3 Hallucination

We label four error classes: *Wrong Format*, *Wrong Length*, *Duplicate Nodes*, and *Made-Up Node*.

5.3.1 Hallucination Severity

While all hallucination types are a result of the LLM failing to accurately respond to the

Table 1: Zero-shot accuracy on GraphQA (higher is better). Best result within each column is **bold**.

	Easy			**Hard**		
Encoder + Model	Acc	Feas	Hall	Acc	Feas	Hall
OPT	0.0	0.0	1.0	0.0	0.0	1.0
Pythonia	0.0	0.0	1.0	0.0	0.0	1.0
GraphSAGE + OPT	0.0	0.0	1.0	0.0	0.0	1.0
GraphSAGE + Pythonia	0.036	0.155	0.845	0.0	0.0	1.0
GIN + OPT	**0.061**	0.534	0.466	0.0	**0.303**	**0.697**
GIN + Pythonia	0.0	0.0	1.0	0.0	0.0	1.0
GAT + OPT	**0.061**	**0.656**	**0.344**	0.0	0.167	0.833
GAT + Pythonia	0.0	0.0	1.0	0.0	0.0	1.0

task, we define a severity ranking of these metrics to analyze encoder benefit. In order of accuracy detriment, we define these in order of decreasing severity as such

- *Wrong Format* : **Most Severe**
  This is a result of the absolute failure of the LLM to provide the response in the requested format. Empirically, this was shown to be generating problem text again, providing no new meaningful information. Figure 5 shows this frequency with regard to $|V|$.
- *Wrong Length*: **Moderately Severe**
  The LLM correctly brackets the result yet fails to generate a sequence of proper length. Empirically, this is shown to be similar to the first, where the LLM will generate problem text again, but *will* bracket the result, therefore understanding the prompt slightly better than the prior. Figure 6 shows this frequency with regard to $|V|$.
- *Duplicate Nodes*: **Less Severe**
  The result is correctly bracketed and has proper length, but it contains more than one duplicate node. This shows clearer prompt understanding, yet breaks the section of the prompt mandating the generation of only one duplicate node per sequence. Figure 3 shows this frequency with regard to $|V|$.
- *Made-Up Node*: **Less Severe**
  The result is correctly bracketed and has proper length, but it contains a false node. This shows clearer prompt understanding, yet breaks the section of the prompt mandating the generation of only nodes described in the prompt. Figure 4 shows this frequency with regard to $|V|$.

5.3.2 Hallucination Analysis

- **Wrong Length** We observe this accounts for 83 % of failures on Pythia and 66 % on OPT. Further, we observe that this accounts for all hallucinations (all inferences as none were feasible) by both models without utilizing the encoders.
- **Wrong Format** is the second most common, predominantly on the GAT and GraphSage encoders.
- **Duplicate_nodes** is rare, other than OPT + GAT at $|V| = 12$, indicating the model remembers to avoid revisiting airports.
- **Made-Up Node** is rarest, occurring only at Pythia + GIN at $|V| = 7$, indicating that the model understands its context.

5.3.3 Take-Aways

While base models tend to hallucinate in the same way, we see variability with hallucinations with regard to encoder types. We observe that base LLMs always produce moderately severe hallucinations, however, they fail to ever produce feasible or accurate output. We observe that encoders attached to the LLM input consistently format incorrectly, outputting the most severe hallucination type. However, more importantly, we note that these models have the capability to properly answer such tasks and are capable of consistently outputting feasible output, especially on small graphs.

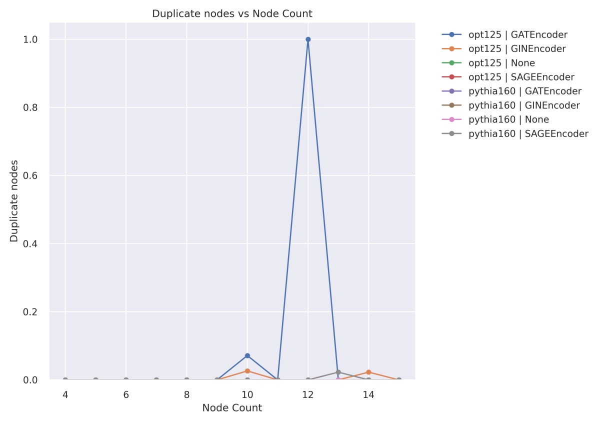

Figure 3: Hallucination: Duplicates vs. Node Count

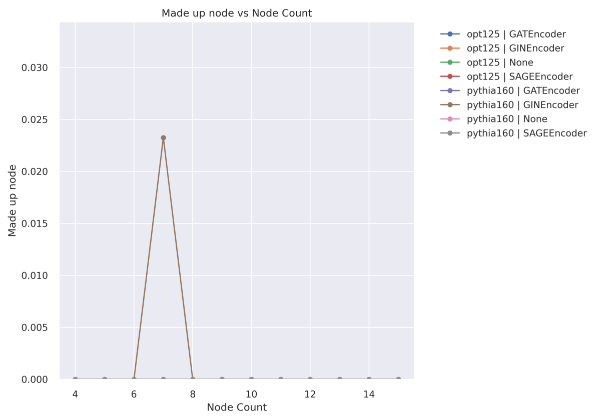

Figure 4: Hallucination: Made-up Nodes vs. Node Count

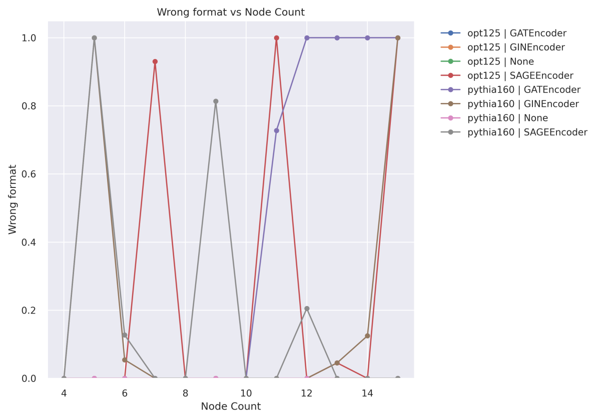

Figure 5: Hallucination: Wrong Format vs. Node Count

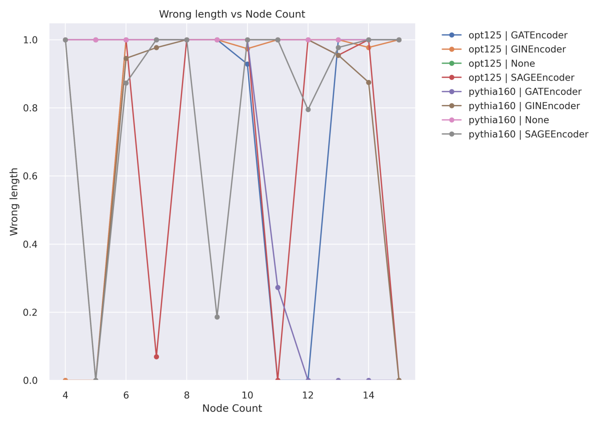

Figure 6: Hallucination: Wrong Length vs. Node Count

6 Summary

Our adapter elevates small-scale TSP accuracy from 0% (raw prompt) to **6.1%** exact match and **65.6%** feasible tours on OPT-125 M. Further our adapter, allows for a reduction in hallucination frequency for all encoder-model combinations, providing strong evidence that our adapter provides the capability of the model to reason regarding prompt and graph structure.

Scaling to larger graphs remains an open challenge: once the number of nodes exceeds the soft-token budget, accuracy collapses and *wrong length* hallucinations dominate. Still, we believe this adapter can be expanded in the future to work with more powerful models and append longer prefix lengths to increase model accuracy even further. Further, we believe this adapter is essential for integrating reasoning capabilities on structured data, as a small accuracy increase is observed while the model has no reduction across any benchmarks as seen in other PEFT variants (ie LoRa or adapter-based).

7 Future Work

As we were heavily constrained by computational capability for this project, keeping graph and model sizes small was essential. In the future, we plan to

- **Use Large Models**
  We intend to implement this with large, open-source models such as LLAMA and DeepSeekR1. As these models possess stronger reasoning abilities and have shown to perform well across a variety of benchmarks, we feel these are excellent candidates to enhance their performance on structured tasks.
- **Increase Prefix Size**
  In this project, we fixed the number of prefix tokens, K, to be 8. In future work, we would like to see how increasing the number of prefix tokens appended to a prompt can increase accuracy. We speculate that the number of prefix tokens should be proportional to the number of datapoints (nodes, in our case) in the structure task, as we observed accuracy decrease once $|V| \geq K$.
- **Comparison with Other PEFT Methods**
  While we only address prefix injection, we hypothesize that implementation of other PEFT methods will show similar increases in accuracy. Further, we believe that a model-level fine-tuning, such as LoRa or adapter-based variant, integrated with prefix injection would yield the highest level of accuracy.

References

[1] Jinheon Baek, Minki Kang, and Sung Ju Hwang. *Accurate Learning of Graph Representations with Graph Multiset Pooling*. 2021. arXiv: 2102.11533 [cs.LG]. URL: https://arxiv.org/abs/2102.11533.

[2] Filippo Maria Bianchi and Veronica Lachi. *The expressive power of pooling in Graph Neural Networks*. 2023. arXiv: 2304.01575 [cs.LG]. URL: https://arxiv.org/abs/2304.01575.

[3] Stella Biderman et al. *Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling*. 2023. arXiv: 2304.01373 [cs.CL]. URL: https://arxiv.org/abs/2304.01373.

[4] Nuo Chen et al. *GraphWiz: An Instruction-Following Language Model for Graph Problems*. 2024. arXiv: 2402.16029 [cs.CL]. URL: https://arxiv.org/abs/2402.16029.

[5] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. *Talk like a Graph: Encoding Graphs for Large Language Models*. 2023. arXiv: 2310.04560 [cs.LG]. URL: https://arxiv.org/abs/2310.04560.

[6] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. 2017. arXiv: 1706.02216 [cs.LG]. URL: https://arxiv.org/abs/1706.02216.

[7] Neil Houlsby et al. *Parameter-Efficient Transfer Learning for NLP*. 2019. arXiv: 1902.00751 [cs.LG]. URL: https://arxiv.org/abs/1902.00751.

[8] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685.

[9] Yoav Levine et al. *Standing on the Shoulders of Giant Frozen Language Models*. 2022. arXiv: 2204.10019 [cs.CL]. URL: https://arxiv.org/abs/2204.10019.

[10] Xiang Lisa Li and Percy Liang. *Prefix-Tuning: Optimizing Continuous Prompts for Generation*. 2021. arXiv: 2101.00190 [cs.CL]. URL: https://arxiv.org/abs/2101.00190.

[11] Bryan Perozzi et al. *Let Your Graph Do the Talking: Encoding Structured Data for LLMs*. 2024. arXiv: 2402.05862 [cs.LG]. URL: https://arxiv.org/abs/2402.05862.

[12] Jianheng Tang et al. *GraphArena: Evaluating and Exploring Large Language Models on Graph Computation*. 2025. arXiv: 2407.00379 [cs.AI]. URL: https://arxiv.org/abs/2407.00379.

[13] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML]. URL: https://arxiv.org/abs/1710.10903.

[14] Keyulu Xu et al. *How Powerful are Graph Neural Networks?* 2019. arXiv: 1810.00826 [cs.LG]. URL: https://arxiv.org/abs/1810.00826.

[15] Rex Ying et al. *Hierarchical Graph Representation Learning with Differentiable Pooling*. 2019. arXiv: 1806.08804 [cs.LG]. URL: https://arxiv.org/abs/1806.08804.

[16] Susan Zhang et al. *OPT: Open Pre-trained Transformer Language Models*. 2022. arXiv: `2205.01068 [cs.CL]`. URL: `https://arxiv.org/abs/2205.01068`.