GRIP: In-Parameter Graph Reasoning through Fine-Tuning Large Language Models

Anonymous Authors[1]

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities in modeling sequential textual data and generalizing across diverse tasks. However, adapting LLMs to effectively handle structural data, such as graphs, remains a challenging problem. Some approaches adopt complex strategies to convert graphs into text sequences, resulting in significant token overhead and rendering them impractical for large-scale graphs. Others introduce additional modules to encode graphs into fixed-size token representations for LLMs. However, these methods typically require large-scale fine-tuning and complex alignment procedures, yet often yield sub-optimal results due to poor modality alignment. Inspired by in-parameter knowledge injection for test-time adaptation of LLMs, we propose GRIP, a novel framework that equips LLMs with the ability to internalize complex relational information from graphs through carefully designed fine-tuning tasks. This knowledge is efficiently stored within lightweight LoRA parameters, enabling the fine-tuned LLM to perform a wide range of graph-related tasks **without requiring access to the original graph** at inference time. Extensive experiments across multiple benchmarks validate the effectiveness and efficiency of our approach.

1. Introduction

In recent years, Large Language Models (LLMs) such as ChatGPT (OpenAI & et al., 2024) and DeepSeek (Liu et al., 2024) have revolutionized the field of artificial intelligence. Pre-trained on large-scale corpora of human knowledge using next-token prediction, these models have demonstrated remarkable generalization capabilities across a variety of downstream tasks, including math problem solving (Cobbe et al., 2021), coding (Guo et al., 2024), tool use (Shen et al., 2023), and knowledge-intensive applications (Lewis et al., 2020; Lála et al., 2023). However, to fully leverage the strong reasoning abilities of LLMs, tasks must first be formulated and expressed in human-readable textual formats. This conversion process is often complex and, in some cases, infeasible—particularly for tasks involving intricate data structures such as graphs.

There have been extensive efforts to adapt LLMs for graph-related tasks. Existing methods can be broadly classified into two categories. The first class of approaches applies LLMs to graph data by converting graphs into text sequences (Wang et al., 2023; Ye et al., 2024; Lin et al., 2024). However, representing graphs as sequences is itself nontrivial and often results in **excessive token overhead or degraded inference performance** (Ye et al., 2024), rendering this approach impractical for large-scale graphs. Moreover, a recent study has demonstrated the inherent limitation of sequential LLMs in solving graph tasks under node permutation (Wu et al., 2024). We provide a more detailed discussion in Appendix A.2. The second class of approaches integrates specialized modules, such as Graph Neural Networks (GNNs), to process graph data and aligns the resulting graph embeddings with LLMs through fine-tuning (Kong et al., 2025; Tang et al., 2024; Chen et al., 2024). However, these methods typically demand **carefully crafted model design and fine-tuning tasks to achieve effective alignment between language and graph modalities**, introducing substantial complexity into the adaptation process. Additional discussion on related works can be found in Appendix A.1.

Inspired by recent advances in parameterized knowledge injection for LLM test-time adaptation (Wang et al., 2024; Mao et al., 2024; Su et al., 2025), we investigate the potential of directly embedding graph information into LLM parameters using Parameter-Efficient Fine-Tuning (PEFT) techniques, such as LoRA (Hu et al., 2021). Specifically, we proposed a method called Graph Reasoning In-Parameterization (GRIP), as shown in Figure 1. In GRIP, we design specialized fine-tuning tasks that guide LLMs to memorize graph contexts and effectively utilize this memorized knowledge for solving downstream tasks. Through the

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

fine-tuning process, the LLM internalizes the structural information of the given graph and encodes it directly into the LoRA parameters. At inference time, the model can perform various tasks related to the injected graph **without requiring access to the original data**. This approach removes the need for specialized graph modules or graph-to-sequence conversions during inference. Extensive experiments validate the effectiveness of our method and demonstrate its strong potential to enhance test-time adaptation of LLMs for structured graph data.

2. Methods

2.1. Preliminaries

Denote a text input by $\mathbf{x} = (x_1, x_2, \ldots, x_L)$ with length L. The language modeling task can be formalized as $\mathcal{L}_{LM}(\mathbf{x};\theta) = -\sum_{i=1}^{L} \log \mathbb{P}(x_i \mid \mathbf{x}_{1:i-1};\theta)$. Low-Rank Adaptation (LoRA) is a widely used technique for efficiently fine-tuning LLMs. Instead of directly fine-tuning the original LLM parameter $W \in \mathbb{R}^{d\times d}$, where d is the hidden dimension of an LLM layer, LoRA learns two matrices $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times d}$, where $r \ll d$. The update to the original weight is given by $\Delta W = A \cdot B$, and the final fine-tuned parameter becomes $\hat{W} = W + \Delta W$.

In this work, we primarily focus on Text-Attributed Graphs (TAGs). A TAG can be represented as $G = (V, E, X_V, X_E)$, where V and E are the sets of nodes and edges, respectively. Each edge $e \in E$ is represented by a triplet $e = (s, r, t)$, where s is the source node, r is the relation, and t is the target node. Each node $v \in V$ and each edge $e \in E$ is associated with a textual description $\mathbf{x}_v \in X_V$ or $\mathbf{x}_e \in X_E$, respectively.

2.2. Graph context memorization

Internalizing graph context into the parameters of a LoRA adapter requires the adapter to first encode (memorize) the relevant graph information. A straightforward approach is to apply existing graph-to-sequence methods to convert the entire graph into a long sequence and fine-tune the LoRA adapter on it. However, this approach inherits the limitations discussed earlier, including substantial token overhead and sensitivity to node ordering permutations. We note that, while converting the entire graph into a sequence aims to provide full graph information for accurate in-context inference, this is unnecessary for fine-tuning. Instead, in GRIP, we decompose the graph into *smaller, independent components* and *encode them separately* by fine-tuning the model to memorize individual nodes' and edges' texts directly. Formally, the task can be defined as:

$$\mathcal{L}_{context} = \sum_{v\in V} \mathcal{L}_{LM}(\mathbf{x}_v;\theta) + \sum_{e\in E} \mathcal{L}_{LM}(\mathbf{x}_e;\theta). \quad (1)$$

This approach also avoids the node permutation issue. However, independently memorizing nodes and edges is insufficient for LLMs to capture the high-order structural information of a graph. To address this limitation, GRIP introduces the summarization task. Specifically, given a graph input, we randomly sample N_s subgraphs, each rooted at a randomly selected node. The size of each subgraph can be controlled by the sampling strategy to avoid excessive token costs. We then prompt the LLM to generate a summary that captures the key information within each subgraph (the detailed prompt design is provided in Appendix B.1). These summaries contain rich local structural information, and their union captures the entire graph information. Denote each generated summary as $\mathbf{s}_i$. After obtaining N_s summaries, we fine-tune the LoRA adapter to memorize them by:

$$\mathcal{L}_{summary} = \sum_{i=1}^{N_s} \mathcal{L}_{LM}(\mathbf{s}_i;\theta). \quad (2)$$

By fine-tuning the LoRA to memorize these summaries, the model directly captures high-order structural and relational information from the graph data. It is also worth noting that this task functions as an input augmentation strategy, a technique shown to be highly effective for knowledge storage and retrieval in LLMs (Allen-Zhu & Li, 2023).

2.3. Enabling in-parameter reasoning through question answering

So far, we have focused on enabling the model to memorize graph context. However, to effectively solve various downstream tasks using the memorized graph knowledge, the model must also learn how to retrieve and apply this information. To this end, we design two types of question answering (QA) tasks: context QA and reasoning QA. Next, we describe each QA task in detail.

Context QA task. The context QA task is designed to instruct the model how to retrieve memorized knowledge. To achieve this, we first randomly sample N_c edges from the graph. For each edge $e = (s, r, t)$, we randomly mask one of s, r, or t and ask the model to predict the missing component given the remaining two. For example, "What is the relationship between s and t?" or "Which node has the relationship of r to node t?" The detailed question templates are shown in Appendix B.2. By training the model to answer such questions, we encourage the model to strengthen its memorization and retrieval of the encoded graph.

Reasoning QA task. The reasoning QA task focuses on enhancing the LLM's ability to perform reasoning over the entire graph. To achieve this, we first sample N_r subgraphs, each rooted at a randomly selected node. We then prompt the LLM to generate reasoning questions and corresponding answers that can be answered by leveraging the provided subgraph. To diversify the types of questions, we design four distinct prompts, encouraging the LLM to generate questions from different reasoning categories. Specifically, questions are divided into four types: local, multi-hop, global, and binary. The detailed prompt for each type can be found in Appendix B.1. For each subgraph, we randomly select one question type for QA task generation.

Let the i-th question and answer be denoted by $\mathbf{q}_i$ and $\mathbf{a}_i$, respectively. The QA objective is formalized as:

$$\mathcal{L}_{QA} = -\sum_{i=1}^{N_c} log\mathbb{P}(\mathbf{a}_i|\mathbf{q}_i;\theta) - \sum_{j=1}^{N_r} log\mathbb{P}(\mathbf{a}_j|\mathbf{q}_j;\theta). \quad (3)$$

2.4. Fine-tuning and inference of GRIP

To facilitate effective fine-tuning, GRIP divide the fine-tuning into two stages. In the first stage, we only fine-tune the LoRA on graph context memorization. The overall loss for the first stage is:

$$\mathcal{L}_{stage1} = \mathcal{L}_{context} + \mathcal{L}_{summary}. \quad (4)$$

At the second stage, we jointly fine-tune the LoRA on both context memorization and reasoning QA:

$$\mathcal{L}_{stage2} = \mathcal{L}_{context} + \mathcal{L}_{summary} + \mathcal{L}_{QA} \quad (5)$$

Other training details can be found in Appendix B.3.

During inference, the model already encodes the graph context within the LoRA parameters, allowing it to directly solve downstream tasks without requiring the graph as explicit input. Let the downstream task be denoted by $\mathbf{t}$ and the graph context by $\mathbf{g}$. The inference process of GRIP is simply $\mathbf{a} = \text{GRIP}(\mathbf{t})$, whereas standard LLM inference requires $\mathbf{a} = \text{LLM}(\text{concat}(\mathbf{g}, \mathbf{t}))$. Since $\mathbf{g}$ typically forms a long sequence—especially for large-scale graphs—standard LLM inference demands a significantly larger context window and incurs higher computational cost compared to GRIP.

3. Experiments

In this section, we conduct various experiments to validate the effectiveness of GRIP. Specifically, we would like to answer the following questions: **Q1**: Can GRIP match or even surpass the performance of standard LLM and existing graph-based models with and without graph context? **Q2**: How effective are the designed fine-tuning tasks in GRIP? More details about the implementation and experimental setting can be found in Appendix B and Appendix C, respectively.

3.1. Setup

Dataset. We evaluate GRIP on three different graph tasks, including scene graph (He et al., 2024b), FB15K237 (Bordes et al., 2013), and WN18RR (Dettmers et al., 2018). The scene graph is a dataset where each graph corresponds to an image. The task is to answer various questions related to the image. FB15K237 and WN18RR are two knowledge graphs, and the task is to complete the missing relations between entities.

Table 1. Performance on the knowledge graphs (IT: input token; OOC: out of context window); .

Model	FB15K237 ↑	Avg #IT ↓	WN18RR ↑	Avg #IT ↓
OFA	70.84	1313.79	30.96	1342.34
GOFA	80.69	1313.79	32.89	1342.34
Qwen2.5-7b	62.60	210.83	33.25	115.01
Qwen2.5-7b$_{\text{context}}$	76.10	1340.59	41.83	1353.46
GRIP-qwen2.5-7b	<u>83.87</u>	210.83	<u>48.50</u>	115.01
Llama3.1-8b	60.90	210.65	24.12	115.69
Llama3.1-8b$_{\text{context}}$	43.13	1313.79	51.24	1342.34
GRIP-llama3.1-8b	<u>63.07</u>	210.65	50.54	115.69
Full-context	OOC	>10M	OOC	>4M

Model and baselines. For GRIP, we use Qwen2.5-7b (Yang et al., 2024) and Llama3.1-8b (Grattafiori et al., 2024) as the base LLM model. For fine-tuning task generation, we use Qwen2.5-7b for all experiments. To ensure a fair comparison, we include baselines from two different types. For LLM-based methods, we choose Qwen2.5-7b and Llama3.1-8b and run the experiments ourselves. For the graph-based method, we include two graph foundational models that can perform zero-shot inference, including OFA (Liu et al., 2023) and GOFA (Kong et al., 2025). The results are directly obtained from the original paper.

Metrics. For all three datasets, we use accuracy as the primary evaluation metric. However, for LLM-based methods, including GRIP, the generated answers may not exactly match the ground-truth labels. To ensure a fair evaluation, we adopt the LLM-as-Judge approach (Li et al., 2024). Specifically, we employ the Qwen2.5-32B model as the judge to assess whether the answers produced by the LLMs are semantically equivalent to the ground-truth labels.

3.2. Main Results

In this section, we present the main experimental results. Performances on the FB15K237 and WN18RR datasets are reported in Table 1, while the results on the scene graph dataset are shown in Table 2. For LLM-based methods, we evaluate two variants. In models labeled with the subscript context, the LLM is provided with both the question and the relevant graph context. For the knowledge graph datasets, where the entire graph is too large to fit within the LLM's context window, we sample a subgraph centered around the two target entities for relation prediction. Additionally, we include a row labeled full-context to indicate the total number of tokens required to represent the complete knowledge graphs when converted into sequential text. For models without the subscript, only the question is provided

Table 2. Performance on the scene graph dataset (IT: input token).

Model	Accuracy ↑	Avg #IT ↓
GOFA	34.06	3354.55
Qwen2.5-7b	4.82	22.06
Qwen2.5-7b$_{\text{context}}$	61.12	4268.84
GRIP-qwen2.5-7b	45.55	22.06
Llama3.1-8b	8.25	23.06
Llama3.1-8b$_{\text{context}}$	53.60	3354.55
GRIP-llama3.1-8b	46.80	23.06

as input—identical to the input used for GRIP.

To answer **Q1**, we first observe that GRIP consistently outperforms baseline LLMs without context across all three datasets. This result demonstrates the effectiveness of GRIP in injecting graph knowledge directly into the LoRA parameters, enabling effective task solving without the need for explicit graph context. When comparing GRIP to corresponding models that utilize explicit graph context, we find that it achieves comparable or even superior performance, particularly on the knowledge graph datasets (indicated in underlined scores). This is because GRIP can effectively embed knowledge from the entire graph into the model parameters—a capability that baseline methods lack, as they rely on subgraph sampling to fit within the model's context window. Consequently, GRIP **can leverage information from the entire graph** to better solve downstream tasks. Furthermore, baseline models require substantially more input tokens to encode graph context, while GRIP achieves comparable or better performance with significantly fewer tokens, highlighting its strong potential, especially for reasoning over large-scale graphs.

However, GRIP's performance on the scene graph dataset is still suboptimal compared to LLMs with explicit context. This may be due to the fact that many questions involve reasoning over numerical object coordinates, which GRIP does not explicitly handle in its current implementation, limiting its ability to memorize and reason over such data.

To answer **Q2**, we conduct an ablation study on the WN18RR dataset to evaluate the impact of the designed summarization and QA tasks. Specifically, we keep all experimental settings fixed while varying the number of summarization tasks N_s and QA tasks N_r, and N_c. The result is reported in Table 3. From the results, we observe that both the summarization and QA tasks play critical roles in enhancing performance. Without both tasks, the model is only capable of memorizing graph context but fails to effectively utilize the stored knowledge for downstream reasoning, consistent with prior observations in (Allen-Zhu & Li, 2023). Introducing either the summarization task or the QA tasks enables the model to begin extracting and applying the memorized knowledge to solve tasks. When both tasks are combined, their synergistic effect significantly improves the model's ability to comprehend and reason over the graph information embedded within the LoRA parameters. We also provide additional ablation studies in Appendix D.

Table 3. Effects of summarization and QA tasks on WN18RR.

N_s	N_c	N_r	Accuracy ↑
0	0	0	2.23
6000	0	0	21.95
6000	2000	2000	32.77
0	6000	6000	31.59
2000	6000	6000	48.12
6000	6000	6000	45.69
10000	10000	10000	48.50

Table 4. Inference time (seconds) on test dataset with Qwen2.5-7b.

dataset	Standard (with context)	GRIP
FB15K237	1366.04	889.76
WN18RR	818.19	469.88
Scene Graph	655.54	302.99

Finally, we report the inference time on all test datasets using both standard LLMs and GRIP in Table 4. As GRIP does not require explicit access to the graph context during inference, it achieves significantly lower inference time compared to standard LLMs, further demonstrating its efficiency. Note that GRIP does involve a fine-tuning phase. However, once fine-tuned, the resulting LoRA parameters can be saved and reused for any future inference tasks on the same graph. As a result, the one-time fine-tuning cost becomes negligible compared to the inference cost, particularly in scenarios such as graphRAG or large-scale knowledge graph applications, where inference is performed repeatedly.

4. Conclusion and Limitation

In this work, we proposed GRIP, a novel approach that enables pre-trained LLMs to adapt to graph-structured data through PEFT and in-parameter knowledge injection. The fine-tuned model can perform various downstream graph tasks without requiring explicit graph context or specialized graph processing modules. We evaluated the effectiveness of GRIP through various experiments. Notably, GRIP achieves comparable or even better performance to baseline methods that rely on explicit graph context, all while operating without such context during inference. However, GRIP does require fine-tuning of the LLM on carefully designed tasks, which may introduce additional computational overhead compared to direct inference approaches. Particularly, for small-scale graphs, the fine-tuning cost can easily surpass the inference cost. Additionally, the current design of the QA task generation mechanism requires further refinement to better support diverse types of graph data and tasks, such as numerical information. We plan to further explore and enhance the capabilities of GRIP in the future.

References

Allen-Zhu, Z. and Li, Y. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316*, 2023.

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

Chen, R., Zhao, T., Jaiswal, A. K., Shah, N., and Wang, Z. Llaga: Large language and graph assistant. In *International Conference on Machine Learning*, pp. 7809–7823. PMLR, 2024.

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

Dettmers, T., Pasquale, M., Pontus, S., and Riedel, S. Convolutional 2d knowledge graph embeddings. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence*, pp. 1811–1818, February 2018. URL https://arxiv.org/abs/1707.01476.

Feng, J., Liu, H., Kong, L., Zhu, M., Chen, Y., and Zhang, M. Taglas: An atlas of text-attributed graph datasets in the era of large graph and language models. *arXiv preprint arXiv:2406.14683*, 2024.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Guo, D., Zhu, Q., Yang, D., Xie, Z., Dong, K., Zhang, W., Chen, G., Bi, X., Wu, Y., Li, Y., et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

He, X., Bresson, X., Laurent, T., Perold, A., LeCun, Y., and Hooi, B. Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning. In *The Twelfth International Conference on Learning Representations*, 2024a.

He, X., Tian, Y., Sun, Y., Chawla, N., Laurent, T., LeCun, Y., Bresson, X., and Hooi, B. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems*, 37:132876–132907, 2024b.

He, Y., Sui, Y., He, X., and Hooi, B. Unigraph: Learning a unified cross-domain foundation model for text-attributed graphs. *arXiv preprint arXiv:2402.13630*, 2024c.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

Kong, L., Feng, J., Liu, H., Huang, C., Huang, J., Chen, Y., and Zhang, M. GOFA: A generative one-for-all model for joint graph language modeling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=mIjblC9hfm.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Lála, J., O'Donoghue, O., Shtedritski, A., Cox, S., Rodriques, S. G., and White, A. D. Paperqa: Retrieval-augmented generative agent for scientific research. *arXiv preprint arXiv:2312.07559*, 2023.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

Li, H., Dong, Q., Chen, J., Su, H., Zhou, Y., Ai, Q., Ye, Z., and Liu, Y. Llms-as-judges: A comprehensive survey on llm-based evaluation methods, 2024. URL https://arxiv.org/abs/2412.05579.

Li, J., Li, D., Savarese, S., and Hoi, S. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pp. 19730–19742. PMLR, 2023.

Li, J., Wu, R., Zhu, Y., Zhang, H., Chen, L., and Zheng, Z. Are large language models in-context graph learners?, 2025. URL https://arxiv.org/abs/2502.13562.

Lin, T., Yan, P., Song, K., Jiang, Z., Kang, Y., Lin, J., Yuan, W., Cao, J., Sun, C., and Liu, X. Langgfm: A large language model alone can be a powerful graph foundation model. *arXiv preprint arXiv:2410.14961*, 2024.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Liu, H., Feng, J., Kong, L., Liang, N., Tao, D., Chen, Y., and Zhang, M. One for all: Towards training one graph model for all classification tasks. In *The Twelfth International Conference on Learning Representations*, 2023.

Mao, Y., Li, J., Meng, F., Xiong, J., Zheng, Z., and Zhang, M. Lift: Improving long context understanding through long input fine-tuning. *arXiv preprint arXiv:2412.13626*, 2024.

OpenAI and et al. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

Shen, Y., Song, K., Tan, X., Li, D., Lu, W., and Zhuang, Y. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

Su, W., Tang, Y., Ai, Q., Yan, J., Wang, C., Wang, H., Ye, Z., Zhou, Y., and Liu, Y. Parametric retrieval augmented generation. *arXiv preprint arXiv:2501.15915*, 2025.

Tan, Y., He, S., Liao, H., Zhao, J., and Liu, K. Dynamic parametric retrieval augmented generation for test-time knowledge enhancement, 2025. URL https://arxiv.org/abs/2503.23895.

Tang, J., Yang, Y., Wei, W., Shi, L., Su, L., Cheng, S., Yin, D., and Huang, C. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 491–500, 2024.

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

Wang, Y., Ma, D., and Cai, D. With greater text comes greater necessity: Inference-time training helps long text generation. *arXiv preprint arXiv:2401.11504*, 2024.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Huggingface's transformers: State-of-the-art natural language processing, 2020. URL https://arxiv.org/abs/1910.03771.

Wu, X., Shen, Y., Shan, C., Song, K., Wang, S., Zhang, B., Feng, J., Cheng, H., Chen, W., Xiong, Y., et al. Can graph learning improve planning in llm-based agents? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., Dong, G., Wei, H., Lin, H., Tang, J., Wang, J., Yang, J., Tu, J., Zhang, J., Ma, J., Xu, J., Zhou, J., Bai, J., He, J., Lin, J., Dang, K., Lu, K., Chen, K., Yang, K., Li, M., Xue, M., Ni, N., Zhang, P., Wang, P., Peng, R., Men, R., Gao, R., Lin, R., Wang, S., Bai, S., Tan, S., Zhu, T., Li, T., Liu, T., Ge, W., Deng, X., Zhou, X., Ren, X., Zhang, X., Wei, X., Ren, X., Fan, Y., Yao, Y., Zhang, Y., Wan, Y., Chu, Y., Liu, Y., Cui, Z., Zhang, Z., and Fan, Z. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Ye, R., Zhang, C., Wang, R., Xu, S., and Zhang, Y. Language is all a graph needs. In *EACL (Findings)*, 2024.

Zhang, M., Sun, M., Wang, P., Fan, S., Mo, Y., Xu, X., Liu, H., Yang, C., and Shi, C. Graphtranslator: Aligning graph model to large language model for open-ended tasks. *arXiv preprint arXiv:2402.07197*, 2024.

Zhu, X., Xue, H., Zhao, Z., Xu, W., Huang, J., Guo, M., Wang, Q., Zhou, K., and Zhang, Y. Llm as gnn: Graph vocabulary learning for text-attributed graph foundation models, 2025. URL https://arxiv.org/abs/2503.03313.

A. Additional Discussions

A.1. Related Works

The success of the foundational language models inspired many works to adapt them to the graph domains and design a foundational model on the graph domain. Typically, existing methods can be divided into two categories: GNN-LLM-based and pure LLM-based.

GNN-LLM based methods. This line of work typically focuses on designing specialized modules for graph data processing, leveraging the capabilities of LLMs to enhance the model's generalization ability. These methods can be broadly categorized into two groups: LLM-as-Enhancer and LLM-as-Predictor. LLM-as-Enhancer approaches utilize LLMs to unify the input space, enabling cross-domain inference across various types of graph data. For example, OFA (Liu et al., 2023) employs LLMs to standardize input features from different datasets, transforming multiple graph classification tasks into a unified binary classification format. TAPE (He et al., 2024a) uses LLMs to generate question-answer pairs and explanations as enriched node features to improve learning. LLM-as-Predictor methods, on the other hand, aim to align embeddings learned from graph models with the representation space of LLMs. For instance, GraphGPT (Tang et al., 2024) fine-tunes a projection module to align embeddings between a pretrained GNN and an LLM. LLaGA (Chen et al., 2024) introduces a creative template-based approach that represents subgraphs using pooled node embeddings for LLM input. Inspired by Q-former (Li et al., 2023), GraphTranslator (Zhang et al., 2024) aligns node and text tokens by connecting pretrained GNN and LLM representations. UniGraph (He et al., 2024c) pretrains a GNN using masked word prediction and then learns a projection function to map graph embeddings into the language space, enabling zero-shot inference. However, these methods often require a carefully designed alignment module and task formulations to achieve successful alignment and struggle to generalize to domains that differ significantly from the training data.

LLM-based methods. Many researchers have also explored the potential of directly using LLMs for graph reasoning. For example, NLGraph (Wang et al., 2023) represents graphs as sequences and evaluates this approach on various structural tasks. InstructGLM (Ye et al., 2024) further investigates alternative strategies for describing graph data in textual form. LangGFM (Lin et al., 2024) directly fine-tunes LLMs on graph sequences and achieves impressive results across several graph benchmarks. Similarly, (Li et al., 2025) interpret the GNN process as a form of Retrieval-Augmented Generation (RAG) and designs specific patterns to represent input graphs as text sequences that simulate GNN computations. PromptGFM (Zhu et al., 2025) also follows this idea by directly simulating the message-passing process of GNNs using LLMs. However, these methods inherently rely on converting graphs into sequences—a nontrivial and often challenging task. A common strategy is to represent the graph using its edge list as input to the LLM. Yet, for graphs with high node degrees or long node feature representations, this approach leads to substantial token overhead, since node information must be redundantly repeated for each connected edge. Moreover, recent work (Wu et al., 2024) has highlighted that LLMs are fundamentally limited in performing graph reasoning tasks due to their sensitivity to the ordering of nodes and edges; such permutations can significantly impact downstream performance.

Parameterized continual learning for LLMs. Recently, the concept of continual learning for LLMs has gained significant attention due to the growing need to adapt LLMs to the latest knowledge and knowledge-intensive tasks during test time. Among various approaches, parameterized continual learning has emerged as a promising direction, offering efficient inference by directly injecting new knowledge into model parameters or PEFT adapters. For example, Parametric RAG (Su et al., 2025) fine-tunes a unique LoRA adapter for each document, allowing the system to retrieve and apply the corresponding adapter at inference time based on the user query, rather than retrieving raw text chunks. DyPRAG (Tan et al., 2025) further advances this idea by training a meta-network to dynamically predict LoRA weights from input documents. LIFT (Mao et al., 2024) and Temp-Lora (Wang et al., 2024) explore the potential of fine-tuning long-context inputs into PEFT parameters, achieving remarkable results. However, to the best of our knowledge, no existing work has extended these ideas to the graph domain.

A.2. The limitation of graph-to-sequence methods

Let the average number of tokens for node and edge features be denoted by t_n and t_e, respectively, and let the average node degree of the graph be d. The minimum total number of tokens required to represent a graph with n nodes is then $nt_n + ndt_e$. To enable an LLM to directly solve graph tasks, a crucial step is to represent the graph data as a text sequence. Typically, there are two standard methods for achieving this: (1) edge with index and (2) edge list. In the following, we describe these two methods in detail and analyze both their token costs and their impact on downstream performance.

Table 5. Ablation study on graph-to-sequence methods with scene graph.

Method	Theoretical avg. token cost	Practical avg. token cost	Accuracy
edge with index	$nt_n + ndt_e$	1726.85	53.86
edge list	$2ndt_n + ndt_e$	4268.84	61.12

Edge with index. For the edge with index method, the graph representation starts by describing all nodes as a sequence, assigning each node a unique ID. For example, each node is represented as: ”NODE_ID: node feature.” Next, each edge is converted into a text sequence of the form ”SRC_NODE_ID, edge feature, TGT_NODE_ID”, where SRC_NODE_ID and TGT_NODE_ID are the IDs of the source and target nodes, respectively. The final sequence representation of the graph is formed by concatenating all node and edge descriptions. Since node IDs are typically simple tokens and their cost can be ignored, the total number of tokens required using the edge with index method is $nt_n + ndt_e$, which matches the minimum token count. However, as each edge now only contains node IDs rather than full node features, the LLM must retrieve the corresponding node features by resolving the IDs within the sequence, introducing additional complexity to the reasoning process, especially for large graphs.

Edge list. Another approach is to directly provide the LLM with the full edge list. In this case, each edge is represented as ”source node feature, edge feature, target node feature”, and there is no need to separately describe all nodes. This method appears more advantageous than the previous one, as the LLM can now directly access the contextual information of each edge without resolving node IDs. However, since the feature representation of each node is repeated for every connected edge, the node information is redundantly included as many times as the node's degree. Consequently, the total number of tokens required to represent the graph becomes $2ndt_n + ndt_e$, which is significantly larger than $nt_n + ndt_e$ when the average degree d is large (as is common in protein-protein interaction or social networks) or when the node feature size t_n is large (as often seen in citation networks).

In Table 5, we compare the performance of the LLM using two different graph representation methods on the scene graph dataset with Qwen2.5-7b (Yang et al., 2024). Although the edge with index method results in a significantly smaller average number of tokens compared to the edge list method, it also degrades downstream performance. For a fair comparison, all LLM baseline results reported in this paper are based on the edge list method. Notes that there are also other advanced methods for converting a graph to a sequence, like the one introduced in PromptGFM (Zhu et al., 2025) or Graph as RAG (Li et al., 2025). However, they usually introduce even much larger token costs than the edge list, and we will not compare them in this work. Meanwhile, there is room for different datasets to further optimize the sequential graph representation. For example, in the node list, we can describe the full feature of each node. In the edge list, we obtain another abstract or keyword for each node. However, this strategy only works for some specific datasets and may not be able to generalize.

B. Implementation Details

In this section, we provide our implementation details of GRIP. The overview of the GRIP is shown in Figure 1. The source code is provided in `https://anonymous.4open.science/r/graph_lora-EBF0/`.

B.1. Prompts Design

In this section, we describe various prompts we used for task generation in GRIP.

B.1.1. Summarization task

As described in the main paper, the goal of the summarization task is to summarize the information in a given sampled subgraph. Further, to facilitate input augmentation, we want the generated summarizations to have different formats and styles. Therefore, given one sampled subgraph, the prompt will ask the LLM to generate two summaries, and the two versions should be as different as possible. Specifically, the prompt we used for the summarization task generation is as follows:

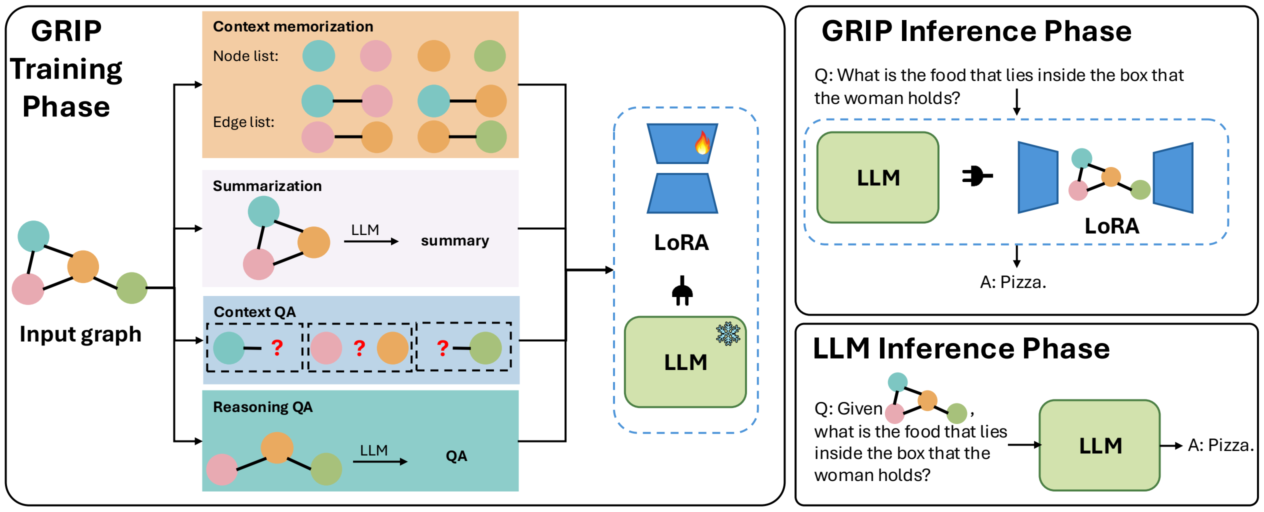

Figure 1. Overview of GRIP. During the fine-tuning phase, we design a variety of tasks to inject graph context into the LoRA parameters and explicitly instruct the model to utilize this context for solving downstream tasks. In the inference phase, GRIP can directly answer user queries without requiring explicit graph context. In contrast, standard LLM-based inference over graphs relies on providing explicit graph context, which introduce significant token overhead, especially for large-scale graphs.

You are given several text snippets as context. Generate two distinct summaries that rephrase all information while exploring potential relationships across the snippets. Avoid directly copying the provided context and try to use a completely different way, tone, writing style, and logic to summarize the original context. Some techniques can be used, including reordering, exchanging active and passive sentences, or synonym replacement. Ensure summaries are concise, accurate, fluent, and complete, without missing any factual or numeric details. Be creative and make sure the two summaries are as DIFFERENT as possible from all aspects described above.
Please answer in the following format:
Summary: [summary] {tuple_delimiter} Summary: [summary].
Please DON'T output quotes when outputting evidence, and separate two summaries by tuple_delimiter. The following are the pieces of context separated by;: {context}

B.1.2. Reasoning QA task

The task generation of the reasoning QA task is similar to summarization. For each sampled subgraph, we will generate two questions. As discussed in the main paper, we divided the reasoning QA into four different types: local, global, multi-hop, and binary. Here we show the prompt we used for each type of QA task generation.

Local QA: You are given several text snippets from a graph as context. Generate two diverse questions and answers focusing on retrieving a single fact using partial information (e.g., infer the entity from an attribute like color, appearance, or infer an attribute from the entity). The answer should be concise and brief, like phrase, words. Keep answers concise (single words or short phrases). Meanwhile, provide one brief evidence sentence per question. Be creative and avoid repetitive patterns or ask relevant information in two questions. Please answer in the following format:
Question: [question] Answer: [answer] Evidence: [evidence] {tuple_delimiter} Question: [question] Answer: [answer] evidence: [evidence]
Please DON'T output quotes when outputting evidence and separate two questions by {tuple_delimiter}. The following are the pieces of context separated by ;: {context}

Global QA: You are given several text snippets as context. Generate two diverse questions and answers that require reasoning over the full context. Keep answers concise (single words or short phrases). Meanwhile, provide one brief evidence sentence per question. Be creative and avoid repetitive patterns or ask relevant information in two questions. Please answer in the following format:
Question: [question] Answer: [answer] Evidence: [evidence] {tuple_delimiter} Question: [question] Answer: [answer] Evidence: [evidence]
Please DON'T output quotes when outputting evidence and separate two questions by {tuple_delimiter}. The following are the pieces of context separated by ;: {context}

Multi-hop QA: You are given several text snippets from a graph as context. Generate two diverse, reasoning-focused questions and answers based on the context. Questions should involve indirect or multi-hop relationships between entities. Each question should involve at least two pieces of context. Keep answers concise (single words or short phrases). Meanwhile, provide one brief evidence sentence per question. Be creative and avoid repetitive patterns or ask for relevant information in two questions. Please answer in the following format:
Question: [question] Answer: [answer] Evidence: [evidence] {tuple_delimiter} Question: [question] Answer: [answer] Evidence: [evidence]
Please DON'T output quotes when outputting evidence and separate two questions by {tuple_delimiter}. The following are the pieces of context separated by ;: {context}

Binary QA: You are given several text snippets as context. Generate two diverse questions and answers: one with the answer "yes" and the other with "no". The question can be asked in the following manner: is there, are there, does, can, has, is it, et al. Meanwhile, provide one brief evidence sentence per question. Be creative and avoid repetitive patterns or ask relevant information in two questions. Please answer in the following format:
Question: [question] Answer: [answer] Evidence: [evidence] {tuple_delimiter} Question: [question] Answer: [answer] Evidence: [evidence]
Please DON'T output quotes when outputting evidence, and separate two questions by {tuple_delimiter}. The following are the pieces of context separated by ;: {context}

B.2. Implementation setting

We implement GRIP based on PyTorch (Paszke et al., 2019), the Hugging Face Transformers library (Wolf et al., 2020), and LIFT (Mao et al., 2024).

For the LoRA adapter, we apply LoRA by default to the MLP modules of each LLM layer. The impact of applying LoRA to different model components is further analyzed in Appendix D.

For subgraph sampling in both the summarization and reasoning QA task generation, we first randomly select a root node from the entire graph. We then iteratively expand the subgraph by sampling neighbors at each hop. Specifically, at each hop, we sample up to three neighbors per node and perform this process for three hops. This procedure results in a subgraph containing at most 10 nodes.

For efficient task generation, we leverage the vLLM inference framework (Kwon et al., 2023). Specifically, we use the Qwen2.5-7B model (Yang et al., 2024) without quantization as the task generator.

To standardize the various tasks in GRIP, we adopt an instruction-tuning template for both context memorization and QA tasks. Each fine-tuning sample is formatted using the `apply_chat_template` function from the Hugging Face Transformers library.

In the following, we describe the detailed construction of samples for each task.

Context memorization. For context memorization tasks, the user prompt asks the model to recite the information in the graph. Specifically, the user prompt and corresponding answer template for node and edge is as follows:

Node Prompt: Given the context graph, recite the information of the node in the graph accurately.
Node Answer: There is a node {node}.

Edge Prompt: Given the context graph, recite the information of the edge in the graph accurately.
Edge Answer: the node {src} is {rel} the node {tgt}.

Summarization task. The user prompt for summarization is similar to the context memorization, where we ask the model to summarize the information in the graph. The detailed user prompt and corresponding answer template is as follows:

Prompt: Based on the context graph, summarize the information in the graph accurately.
Answer: {summary}

Context QA task. For context QA task, given a sampled edge triplet (s, r, t), we will generate a prompt to ask the model to predict one element based on the rest of two. Namely, let Src, Rel, Tgt to represent s, r, and t, we have:

Src Prompt: In this context graph, which node has the relation {rel} to node {tgt}?
Src Answer: {src}

Rel Prompt: Based on the context graph, what is the relation between the node {src} and the node {tgt}?
Rel Answer: {rel}

Tgt Prompt: Given this context graph, which node has the relation {rel} from the node {src}?
Tgt Answer: {tgt}

Reasoning QA tasks. For the reasoning QA task, the user prompt and answer is just the generated question and the answer.

B.3. Fine-tuning setting

The fine-tuning is divided into two stages. For both stages, we will set a maximum number of training epochs and also monitor the training loss as the threshold for early stopping to determine the end of the stage. The optimizer is AdamW for all datasets. Other hyperparameter settings that are different for different datasets are described in Appendix C

C. Experiment details

C.1. Dataset details

Table 6. Dataset statistics. (W. represent word.)

Dataset	Avg. #N	Avg. #E	Avg. #N. W.	Avg. #E. W.	# G
FB15K237	14,541	310,116	20.1	8.4	1
WN18RR	40,943	93,003	23.3	11.0	1
Scene Graph	19.13	68.44	20.1	9.8	100000

In this section, we describe the details of all datasets we used for evaluation. The statistic of all datasets can be found in Table 6. The raw data of all datasets are obtained from TAGLAS (Feng et al., 2024).

Scene Graph. Scene Graphs is a graph question-answering dataset on scene graphs. Each graph in SceneGraphs contains objects connected by the relationship between two objects. It contains 59,978/19,997/20,025 graph samples for the train/val/test sets. Due to the time limitation, we only evaluate all methods on the first 500 test samples.

FB15K237 FB15K237 is a knowledge graph. The dataset contains 14,541 nodes and 310,116 relations. Nodes in the dataset are entities in the knowledge graph and edges represent the relation between two entities. It contains 237 different relation types. There are a total of 272,115/17,535/20,466 samples for train/val/test sets, respectively. For simplicity, in evaluation, we only evaluate the first 3000 samples from the test set, and each sample is formalized as a 10-way classification task. That is, the question will ask the model to select the correct one relation from 10 candidate relations, instead of all 237 relations.

WN18RR WN18RR is another knowledge graph extracted from WordNet. It contains 40,943 nodes and 93,003 relations, where each node is an English word and each edge represents the relation between two words. It contains 11 different relation types and 86,835/3,034/3,134 samples for train/val/test sets, respectively. For WN18RR, we evaluate on the whole test set.

C.2. Training process and hyperparameters

Table 7. The hyperparameter settings for all experiments

hyperparameters	Scene Graph	FB15K237	WN18RR
lora_r	8	24	16
lora components	MLP	MLP	MLP
N_s	100	10000	10000
N_r	100	10000	10000
N_c	100	10000	10000
stage1 maximum epoch	100	1	1
stage2 maximum epoch	100	2	2
early stop loss threshold	0.5	0.15	0.15
gradient accumulation steps	full	512	512
learning rate	1e-3	1e-3	1e-3
scheduler	linear	linear	linear
β_1	0.9	0.9	0.9
β_2	0.98	0.98	0.98
ϵ	1e-4	1e-4	1e-4
maximum gradient norm	1.0	1.0	1.0

All experiments can be conducted on a single NVIDIAA100_SXM4_80GB GPU. To accelerate the training and inference process, we use 4 GPUs in total and split all datasets into 4 subsets and process them independently on each GPU. In Table 7, we provide hyperparameter settings for all three datasets.

D. Addtitional ablation studies

D.1. The effects of LoRA

Table 8. Ablation study on LoRA adapter position.

Accuracy	ATT	MLP	ALL
Scene Graph	37.62	45.55	44.61

Table 9. Ablation study on LoRA adapter rank.

Accuracy	r=4	r=8	r=16	r=24
FB15K237	61.67	75.63	83.87	83.93

In this section, we present an additional ablation study to investigate the impact of applying the LoRA adapter to different components of the transformer architecture. Specifically, the LoRA adapter can be applied to both the attention module and the feed-forward MLP module. To isolate the effect of each component, we fix all other settings and vary only the placement of the LoRA adapter. We evaluate three configurations: (1) applying LoRA to the attention module (including the Key, Value, and Query weight matrices), denoted as **ATT**; (2) applying LoRA to the feed-forward MLP module, denoted as **MLP**; and (3) applying LoRA to both modules, denoted as **ALL**.

Table 10. Effects of summarization and QA tasks on FB15K237.

N_s	N_c	N_r	Accuracy ↑
0	0	0	7.53
10000	0	0	26.90
10000	4000	4000	80.97
0	10000	10000	71.70
4000	10000	10000	74.73
6000	6000	6000	70.47
10000	10000	10000	83.93

We train GRIP on the scene graph dataset, and the results are summarized in Table 8. We can see that applying LoRA only on attention modules achieves much worse performance compared to the other two settings. The findings indicate that applying LoRA to the MLP modules is most critical for improving downstream performance. This result is intuitive, as the MLP modules are typically responsible for knowledge storage, while the attention modules primarily handle information retrieval. Therefore, apply LoRA on the MLP module can be more effective for graph knowledge injection.

Next, we investigate the effect of the rank of LoRA using the FB15K237 dataset. To conduct the ablation study, we fix all other hyperparameters and only vary the rank r. Notes that for this ablation study, we apply LoRA only on MLP modules. The result is shown in Table 9. We can see that as the rank become larger, and the performance also increases, and eventually becomes saturated. At this time, the bottomneck becomes the task design instead of the LoRA parameters.

D.2. Additional ablation results on the task design

To further verify the effectiveness of our proposed fine-tuning tasks in GRIP, we further conduct the ablation study with FB15K237. The experimental setting is similar to the ablation study using the WN18RR dataset, and the result is shown in Table 10. We can see the results are consistent with Table 3 and both the summarization tasks and QA tasks are crucial for the downstream performance.