GRAFF: GRaph-Augmented Fine-grained Fusion for Large Language Models

Himanshu Chaudhary[1] **Ruida Wang**[2] **Gowtham Ramesh**[3] **Junjie Hu**[1]
[1]Department of Computer Sciences, University of Wisconsin–Madison,
[2]University of Illinois Urbana-Champaign, [3]AMD
`{hchaudhary3, junjie.hu}@wisc.edu`,
`ruidaw@illinois.edu`, `gowtham.ramesh@amd.com`

Abstract

Recent advancements in large language models (LLMs) have showcased remarkable text generation capabilities. However, due to the inherent ambiguity of natural language and the unstructured nature of text modality, LLMs still struggle to integrate structured information (e.g., graphs) effectively. This hinders their ability to leverage high-quality structured data in specialized domains. Thus, recent research has explored various methods to integrate graph structures into LLMs to improve generation. However, existing methods typically compress the graph's structural information into only a single token, which is concatenated with detailed text tokens for LLMs, restricting their ability to capture deep semantic and structural information. To overcome these limitations, we propose **GR**aph-**A**ugmented **F**ine-grained **F**usion (GRAFF), a novel method that integrates fine-grained node-level structural information with corresponding text entities to LLMs via a lightweight, structure adapter module. Specifically, we introduce a dual-channel graph input mechanism to separate structural and semantic components for graph encoding, producing more expressive graph representations. We then incorporate a graph attention (GAT) module into LLMs' intermediate decoder layers to process structural information, enhancing the model's capability in graph-based question answering. Extensive experiments show that GRAFF significantly improves LLMs' graph-understanding ability in question answering, outperforming baselines by an average of 10.14% across four datasets. The official code for this work is available at `https://github.com/hcpv/GRAFF`.

1 Introduction

Recently, large language models (LLMs) have revolutionized numerous areas with their strong text-generation capabilities (Devlin et al., 2019; Brown et al., 2020). Leveraging the expressive power of language, the versatility of next-token prediction training (Radford, 2018), and autoregressive generation, LLMs achieve state-of-the-art performance on text tasks. However, due to the inherent ambiguity of natural language and the unstructured nature of massive noisy text data used in training, LLMs are prone to generate hallucinated contents, particularly in highly specialized areas where the domain data is limited (Wang et al., 2024b). This drawback not only hinders the reliability of LLMs but also prevents them from leveraging high-quality structured data annotated in downstream applications (Kasai et al., 2024; Dhingra et al., 2022).

To mitigate hallucinations and improve the performance of LLMs on unfamiliar tasks and knowledge, various methods have been proposed (Wei et al., 2021; Houlsby et al., 2019; Hu et al., 2021a; Wang et al., 2023b; Lewis et al., 2020). An overview of previous methods can be found in

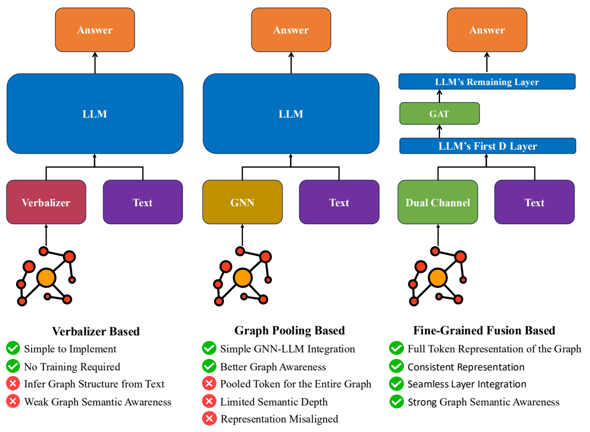
Answer
LLM
Verbalizer
Text
Answer
LLM
GNN
Text
Answer
LLM's Remaining Layer
GAT
LLM's First D Layer
Dual Channel
Text
Verbalizer Based
✓ Simple to Implement
✓ No Training Required
✗ Infer Graph Structure from Text
✗ Weak Graph Semantic Awareness
Graph Pooling Based
✓ Simple GNN-LLM Integration
✓ Better Graph Awareness
✗ Pooled Token for the Entire Graph
✗ Limited Semantic Depth
✗ Representation Misaligned
Fine-Grained Fusion Based
✓ Full Token Representation of the Graph
✓ Consistent Representation
✓ Seamless Layer Integration
✓ Strong Graph Semantic Awareness

Figure 1: Traditional methods for integrating graphs into LLMs rely on verbalizers to flatten the graph into text. While this approach is simple, it fails to capture the complex structural relationships within the graph. Graph Pooling based methods compress the graph into a single token before being input into the LLM, limiting the GNN's ability to capture the complex semantics of individual nodes. GRAFF addresses the limitations of previous methods while preserving their advantages.

Figure 1. Among them, retrieval-augmented generation (RAG) (Guu et al., 2020; Lewis et al., 2020; Shuster et al., 2021) has gained significant attention for its ability to improve LLM performance by retrieving information from dynamic data sources. Building on RAG, a growing line of work has proposed to retrieve structured information from graph-based data sources, and directly verbalize the retrieved graph as flattened text inputs to LLMs (He et al., 2023). However, this *verbalization-based* paradigm makes it hard for LLMs to capture fine-grained, structured relationships among entity nodes in the graph, limiting their effectiveness in complex question-answering tasks (Dziri et al., 2023; Wang et al., 2024a). This challenge has motivated research into more effective fusion methods for integrating graph structure directly into modern LLMs.

An alternative line of work addresses this by learning neural components that encode graphs into embeddings for augmentation (Guo et al., 2023; Stechly et al., 2023; Fatemi et al., 2023). Recent approaches such as GraphToken (Perozzi et al., 2024), GRAG (Hu et al., 2024), and G-Retriever (He et al., 2024) leverage graph attention networks (GATs) to encode graphs as soft prompts or prefixes to LLMs, outperforming *verbalization-based* methods on graph QA tasks. However, these methods face key limitations. They typically encode graphs only into shallow layers of LLMs, limiting access to deeper semantic information that emerges in later layers (Jin and Rinard). Furthermore, most approaches adopt a *graph-pooling* paradigm that compresses the graph into a single embedding token prepended to text inputs, which, while efficient, often loses the fine-grained graph signal. Moreover, models like G-Retriever use a separate encoder (e.g., BERT) for graph nodes, leading to misalignment between graph and language representations.

To address these challenges, we propose GRAFF, a lightweight, structure-aware module for integrating *fine-grained node-level* information from retrieved graphs into LLMs. GRAFF follows parameter-efficient fine-tuning principles by introducing a plug-in adapter at intermediate LLM layers alongside a modified input schema. It employs a *dual-channel graph input* strategy that separates a text-attributed graph input into structural and textual components. A lightweight GAT module encodes the structural channel and injects the resulting embeddings into the LLM at an intermediate decoder layer. This design offers two key advantages: (1) it eliminates embedding space misalignment by reusing LLM's own embeddings for graph encoding, avoiding reliance on external encoders; and (2) it enables seamless integration at deeper layers to fuse structural and semantic information for improved graph-based QA performance.

We conduct extensive experiments on three publicly available real-world datasets and one synthetic dataset for graph-based question answering. The real-world datasets include WebQSP (tau Yih et al., 2016), ExplaGraph (Saha et al., 2021), and SceneGraph (Hudson and Manning, 2019). The synthetic dataset contains counterfactual examples created by perturbing graph nodes to produce altered answers. Experimental results show that incorporating GRAFF improves LLMs' QA performance over fine-tuning methods such as LoRA or G-Retriever by at least **3.77%** on the real-world datasets. More notably, GRAFF outperforms other fine-tuning methods by at least a **36.19%** on the counterfactual synthetic dataset. This substantial gain suggests that the adapter effectively guides the LLM to rely on graph-structured information rather than its memorized pretraining knowledge on graph-related questions. Furthermore, we conduct comprehensive ablation and hyperparameter studies to better understand the sensitivity of GRAFF.

In summary, our contributions are threefold:

- We propose **GRAFF**, a lightweight adapter that fuses text-attributed graphs into the intermediate decoder layers of LLMs, enhancing their graph-understanding ability.
- **GRAFF** introduces a novel *dual-channel graph input* approach, which separates the structural and textual components of graph nodes within the LLM's input.
- Extensive experiments show that GRAFF significantly improves the graph-understanding ability of LLMs in graph-based question answering, outperforming the strongest baseline by an average of **10.14%** across four datasets.

2 Method

This section details the proposed GRAFF. Figure 2 shows an overview of the architecture. The central idea is to incorporate text-attributed graphs directly into the LLM's generation process. We begin by outlining the preliminaries of graph attention networks (GAT; §2.1) and the preprocessing steps used to encode graph structure (§2.2). We then describe the design of our GRAFF module (§2.3) and the training procedure for GRAFF (§2.4).

2.1 Preliminary: Graph Attention Network

The graph attention network (GAT) (Veličković et al., 2017) is widely used for encoding graph-structured information. In this work, we leverage the GAT architecture to design the key encoding module for encoding text-attributed graphs in our GRAFF. We define an input graph as $G = (V, E)$, where $V = \{v_i\}_{i=1}^{n_v}$ is the set of nodes and $E = \{e_{i,j} \mid v_i, v_j \in V\}$ is the set of edges. Here, v_i denotes the i-th node, n_v is the total number of nodes in the graph, and $e_{i,j}$ denotes a relational edge between v_i and v_j. Notably, edges can be heterogeneous, encoding different types of relations between nodes. Additionally, let $\mathcal{N}(i)$ denote a set of neighbors of node v_i, excluding self-connections.

The main function of GAT is to aggregate information from adjacent nodes and encode the structured information into the node representation. To do so, we present the layer-wise computation of a GAT block as follows:

$$\begin{aligned}\mathbf{h}_i^{(l+1)} &= \mathrm{GAT}^{(l)}\left(\mathbf{h}_i^{(l)}, \mathcal{N}(i)\right) \\ &= \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}^{(l)} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)} \in \mathbb{R}^d \end{aligned} \tag{1}$$

$$\alpha_{ij}^{(l)} = \frac{\exp\left(f_{ij}^{(l)}\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(f_{ik}^{(l)}\right)} \in \mathbb{R} \tag{2}$$

$$f_{ij}^{(l)} = \sigma\left(\mathbf{a}^\top [\mathbf{W}_h^{(l)} \mathbf{h}_i^{(l)}; \mathbf{W}_h^{(l)} \mathbf{h}_j^{(l)}; \mathbf{W}_e^{(l)} \mathbf{e}_{ij}^{(l)}]\right) \tag{3}$$

where σ is a non-linear activation function such as the LeakyReLU function, $\mathbf{W}^{(l)}, \mathbf{W}_h^{(l)}, \mathbf{W}_e^{(l)} \in \mathbb{R}^{d \times d}$, $\mathbf{a} \in \mathbb{R}^{3d}$ are all learnable parameters at the l-th layer of the GAT, and $l \in [0, L]$.

2.2 Graph Preprocessing

Following G-Retriever (He et al., 2024), our model considers the input of the text-attributed graph in which each node or edge is associated with a text label. However, due to the LLM's tokenization, the text label of a node may be tokenized into a span of sub-word tokens. We denote the text spans of a node and an edge as $\text{text}(v_i) = \left[x_1^{(i)}, \cdots, x_{t_v}^{(i)}\right]$, $\text{text}(e_{i,j}) = \left[y_1^{(i,j)}, \cdots, y_{t_e}^{(i,j)}\right]$ respectively, where $x_k^{(i)}, y_k^{(i,j)}$ are text tokens, and t_v and t_e are the lengths of the text spans. Our method starts by preprocessing the graph nodes and edges to prepare the input embeddings for the GAT module.

Firstly, we use average pooling to aggregate text embeddings of different tokens into a single vector to obtain the input for GRAFF. We reuse the LLM itself to encode the text spans of a node and an edge, i.e., $\text{embed}(v_i) = \left[\boldsymbol{x}_1^{(i)}, \cdots, \boldsymbol{x}_{t_v}^{(i)}\right]$, $\text{embed}(e_{i,j}) = \left[\boldsymbol{y}_1^{(i,j)}, \cdots, \boldsymbol{y}_{t_e}^{(i,j)}\right]$. Next, we formulate the input embeddings of a node v_i and an edge $e_{i,j}$ to the GAT module as:

$$\mathbf{h}_i' = \frac{1}{t_v} \sum_{k=1}^{t_v} \boldsymbol{x}_k^{(i)} \in \mathbb{R}^d \tag{4}$$

$$\mathbf{e}_{i,j}' = \frac{1}{t_e} \sum_{k=1}^{t_e} \boldsymbol{y}_k^{(i,j)} \in \mathbb{R}^d \tag{5}$$

where d is the LLM's hidden dimension. We apply this process at the initial input stage no matter which layer we add our adapter to. This reduces the number of input tokens for each node and edge and simplifies the internal structure of the model.

2.3 GRAFF

We present our GRAFF, beginning with the *dual-channel graph input* method (§2.3.1), which enhances the representation of graph signals to LLM. We then present the GAT module (§2.3.2), which enables the LLM to capture multi-hop structural information. Finally, we describe how this information is re-integrated into the LLM (§2.3.3).

2.3.1 Dual-Channel Graph Input

To disentangle the graph-structured information from the semantic information of nodes' and edges' text spans, we propose the *Dual-Channel Graph Input* method to encode these two types of information separately. Specifically, we consider a text-associated graph $G = (V, E)$ where an edge span can be represented as a text span consisting of a head node, a relation, and a tail node. The conventional way to consider relational edges as input to LLMs is to directly concatenate the text embeddings of all edge spans in the graph.

$$\begin{aligned}\boldsymbol{G}_{\text{text}} = [&\text{embed}(v_i), \text{embed}(e_{i,j}), \text{embed}(v_j), \\ &\cdots], \ \forall e_{i,j} \in E\end{aligned} \tag{6}$$

where $\boldsymbol{G}_{\text{text}} \in \mathbb{R}^{L_g \times d}$. L_g is the total number of text tokens for the flattened graph. The encoded flattened text-graph, together with the question, is passed to the LLM for generating the answer. However, this single-channel input method has no

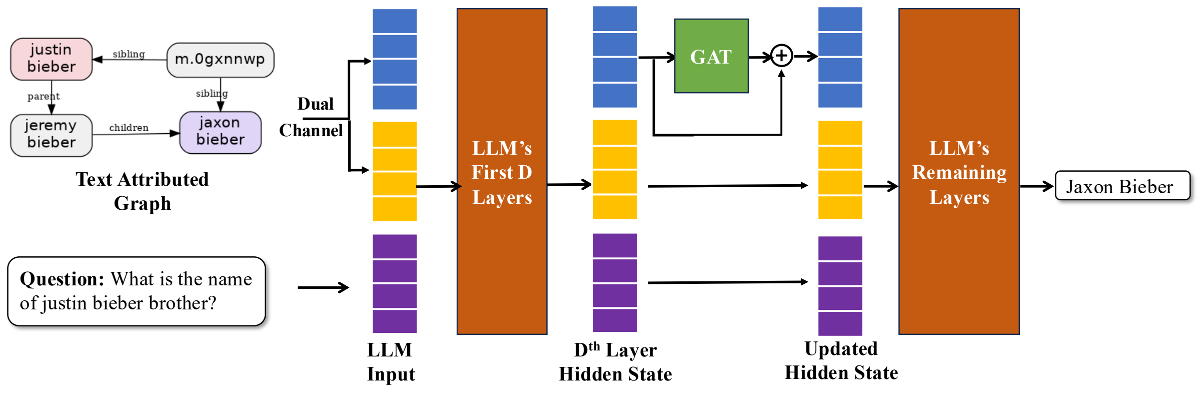

Figure 2: Overview of GRAFF. Given an input text-attributed graph and a question, we first construct a Dual Channel Graph to separately encode structural and textual information. Structural inputs (blue tokens), textual inputs (yellow tokens), and the question (violet tokens) are fed into the model. These are processed through the first D decoder layers of the LLM. At the D^{th} layer, the GAT module operates over the structural tokens to inject multi-hop graph information into their hidden representations. The updated states are then passed through the remaining LLM layers. This allows GRAFF to enhance reasoning by combining graph structure with language semantics.

explicit way to consider multi-hop structured information in the graph and does not effectively separate language semantic information and graph-structured information.

To overcome this, we propose to add an additional channel of input for the graph structures, which is the preprocessed single-token representation of the graph, denoted as:

$$\boldsymbol{G}_{\text{agg}} = [\mathbf{h}_i', \mathbf{e}_{i,j}', \mathbf{h}_j', \cdots],\ \forall e_{i,j} \in E \tag{7}$$

where $\boldsymbol{G}_{\text{agg}} \in \mathbb{R}^{3|E|\times d}$ denotes the representations of the aggregated graph. This aggregated graph channel of input will be used by the GAT module to further encode graph-structured information. Finally, we concatenate both channel inputs and pass them to the LLM decoder.

$$\boldsymbol{G}_{\text{input}} = [\boldsymbol{G}_{\text{agg}}, \boldsymbol{G}_{\text{text}}] \in \mathbb{R}^{(L_g+3|E|)\times d} \tag{8}$$

Notably, we place the $\boldsymbol{G}_{\text{agg}}$ at the beginning of $\boldsymbol{G}_{\text{input}}$, which allows us to easily extract the corresponding node embeddings based on their indices in the combined sequence. This also simplifies our processing to extract the node embeddings for the GAT adapter in Eq. (10) in the following subsection.

2.3.2 GAT Process

We use LLMs to encode the dual channel inputs obtained from Eq. (8), and add the GAT module at the D-th decoder layer, where D is a user-tunable hyperparameter. Formally, we denote the processed graph input channel as:

$$\boldsymbol{G}_{\text{agg}}^{(D)} = \left[\boldsymbol{h}_i^{(D)}, \boldsymbol{e}_{i,j}^{(D)}, \boldsymbol{h}_j^{(D)}, \cdots\right],\ \forall e_{i,j} \in E \tag{9}$$

where $\mathbf{h}_i^{(D)}$ is the single-token representation for node v_i at the D-th LLM decoder layer. We then pass the embeddings of each node $\boldsymbol{h}_i^{(D)}$ to the GAT module to process structural information. We also perform a downsampling to ensure that our lightweight GAT module uses a smaller hidden dimension than the LLMs' hidden dimension.

$$\boldsymbol{h}_{i,\text{down}}^{(D)} = \boldsymbol{W}_{\text{down}}\boldsymbol{h}_i^{(D)} + \boldsymbol{b} \tag{10}$$

$$\boldsymbol{h}_{i,\text{GAT}}^{(D)} = \text{GAT}\left(\boldsymbol{h}_{i,\text{down}}^{(D)}, \mathcal{N}(i)\right) \tag{11}$$

2.3.3 Graph-LLM Integration

After we obtain the GAT processed structural embeddings for all nodes $\{\mathbf{h}_{i,\text{GAT}}^{(D)}\}_{v_i\in V}$, we integrate them back into the subsequent LLM layers. Inspired by Wang et al. (2023a), we propose the following integration method. First, we project the GAT output embeddings back to the LLM embedding space by a linear upsample layer as follows:

$$\mathbf{h}_{i,\text{up}}^{(D)} = \mathbf{W}_{\text{up}}\mathbf{h}_{i,\text{GAT}}^{(D)} + \boldsymbol{b} \in \mathbb{R}^d \tag{12}$$

where $\mathbf{W}_{up} \in \mathbb{R}^{d\times n}$ is a learnable up projection matrix and $\boldsymbol{b} \in \mathbb{R}^d$ is learnable bias term.

Subsequently, we add the GAT-processed node embeddings back to the LLM embedding through a residual connection to maintain the staleness of the network as follows:

$$\mathbf{h}_{\mathbf{i}}^{(\mathbf{D})} \leftarrow \mathbf{h}_i^{(D)} + \mathbf{h}_{i,\text{up}}^{(D)} \in \mathbb{R}^d. \tag{13}$$

After this residual connection, the updated node embedding $\mathbf{h}_{\mathbf{i}}^{(\mathbf{D})}$ has fused the graph-structured information and is passed through the remaining LLM layers higher than the D-th layer.

2.4 Training

We use supervised fine-tuning to train our GRAFF with a base LLM on a graph-annotated dataset, consisting of (G, Q, A) tuples. Specifically, given a question input Q and a structured graph input G, the model generates an answer sequence $A = (a_1, a_2, ..., a_r)$ autoregressively. The input to the LLM consists of both the textual and structured graph representations as processed by Eq. (8). The generation process is defined as:

$$p_{\theta,\phi}(A|Q,G) = \prod_{i=1}^{r} p_{\theta,\phi}(a_i|a_{<i}, Q, \boldsymbol{G}_{\text{input}}),$$

where ϕ is the trainable GRAFF, and θ is the frozen LLM parameters. The training only updates ϕ to minimize the negative log-likelihood loss:

$$\mathcal{L}(\phi) = -\sum_{i=1}^{r} \log p_{\theta,\phi}(a_i|a_{<i}, Q, \boldsymbol{G}_{\text{input}}). \quad (14)$$

3 Experiment

We conduct extensive experiments to compare the performance of different models in this section. Additionally, we conduct hyperparameter studies (§3.3) and ablation studies (§3.4) to evaluate the effectiveness of GRAFF.

3.1 Experiment Setup

3.1.1 Dataset and Task

We evaluate our method on the GraphQA benchmark (He et al., 2024), which consists of three text-attributed graph datasets: ExplaGraphs (Saha et al., 2021), SceneGraphs (Hudson and Manning, 2019), and WebQSP (tau Yih et al., 2016). ExplaGraphs includes commonsense-augmented graphs designed for commonsense reasoning tasks, while SceneGraphs contains visual scenes using textual object-attribute relationships. WebQSP contains multi-hop questions over structured knowledge graphs. Strictly following the retrieval setup of G-Retriever (He et al., 2024), we use the same retrieved graphs as G-Retriever for SceneGraphs and WebQSP. We report QA accuracy for ExplaGraphs and SceneGraphs, and Hit@1 for WebQSP. The dataset statistics are summarized in Table 2.

To further evaluate the effectiveness of GRAFF, we introduce *SyntheticGraph*, a synthetic graph dataset derived from ExplaGraphs. This dataset introduces counterfactual information in both the graph and the questions, serving as a challenging dataset for assessing the LLM's ability to rely solely on the provided graph rather than its pre-trained knowledge. We construct *SyntheticGraph* by shuffling the nodes and corresponding arguments in the original questions from ExplaGraphs while preserving the original edge connections. This transformation ensures that the reasoning path remains valid and the correct answer is unchanged, while simultaneously disrupting shallow linguistic cues in the context. As a result, the model cannot rely on memorized textual patterns from pre-training but must effectively utilize the structured information in the graph.

3.1.2 Implementation Details & Baseline

All experiments are conducted on a single NVIDIA A40 GPU with 48 GB of memory. We use LLaMA-3.2-3B (LLaMATeam, 2024) as the base language model, and employ a 2-layer Graph Attention Network (GAT) with 4 attention heads per layer and a hidden dimension size of 3,072 to encode the input graph. The entire model, including the adapter module, is fine-tuned end-to-end using the training split provided in the GraphQA benchmark introduced by G-Retriever (He et al., 2024). We optimize the model using the AdamW optimizer (Loshchilov and Hutter, 2019) with an initial learning rate of 1e-5 and a weight decay of 0.05. Each experiment is run for up to 10 epochs with a batch size of 2 and is repeated across 4 random seeds to ensure robustness.

We compare a comprehensive set of strong baselines grouped into three categories:

- **Inference Only:** Methods that operate LLMs without any task-specific fine-tuning, including Zero-Shot (Base), Zero-Shot (Chat) using the base and chat versions of LLaMa-3.2 (3B), and KAPING (Baek et al., 2023).
- **Tuning LLMs without GNN:** Methods that adapt the LLM using parameter-efficient tuning strategies without leveraging graph structure; this includes Prompt Tuning (Li and Liang, 2021) and LoRA (Hu et al., 2021b).
- **Tuning LLMs with GNN:** Methods that integrate graph structure explicitly through GNN modules into LLMs for fine-tuning, such as GraphToken (Perozzi et al., 2024), KG-Adapter (Tian et al., 2024), GRAG (Hu et al., 2024), G-Retriever (He et al., 2024), and our proposed method GRAFF.

Category	Method	WebQSP	ExplaGraph	SceneGraph	Synthetic Graph
Inference Only	Zero Shot (Base)	32.7	13.5	33.1	11.1
	Zero Shot (Chat)	53.4	52.6	50.7	44.7
	KAPING*	52.6	62.2	43.7	–
Tuning LLMs w/o GNN	Prompt Tuning	57.9	60.2	58.3	50.3
	LoRA	<u>71.1</u>	88.9	<u>85.3</u>	<u>58.3</u>
Tuning LLMs w/ GNN	GraphToken*	57.1	85.1	49.1	–
	KG-Adapter*	68.7	–	–	–
	GRAG	68.9	<u>88.9</u>	–	–
	G-Retriever	67.4	83.7	82.3	56.4
	GRAFF	**72.2**	**92.5**	**90.2**	**79.4**
	Relative % Gain (Best vs 2nd)	+1.5	+4.1	+5.7	+36.2

Table 1: Performance comparison with different baselines. Bold values indicate the best performance, while underlined values denote the second-best. Results for methods marked with * are taken from their original publications; all other results are based on our implementations using the LLaMA-3.2-3B model.

Dataset	WebQSP	ExplaGraph	SceneGraph
#Graphs	4,737	2,766	100,000
Avg. #Nodes	8.39	5.17	8.21
Avg. #Edges	8.16	4.25	12.00

Table 2: Statistics of the experiment datasets.

3.2 Results and Analysis

The main results are summarized in Table 1, showing that LLMs enhanced with GRAFF consistently achieve the best performance across all datasets in the GraphQA benchmark. Among methods in the *Tuning LLMs with GNN* category, GRAFF outperforms all competing approaches, including GRAG and G-Retriever, by an average margin of 6.3%. When compared across all categories, including *Inference Only* and *Tuning LLMs without GNN*, GRAFF achieves an average improvement of 3.77% over the best-performing baseline, LoRA.

The most substantial gains are observed on the Synthetic Graph dataset, where GRAFF surpasses the strongest baseline by 36.2%. This highlights the model's superior capacity for reasoning under counterfactual perturbations and its ability to effectively leverage fine-grained structural signals at the node level. Overall, the results demonstrate that integrating node-level graph information at intermediate layers of the LLM enables more consistent and accurate multi-hop reasoning across both real-world and synthetic graph-based QA tasks. We further report performance variability and statistical significance tests for all datasets in Appendix A. To assess whether these improvements are specific to the LLaMA-3.2-3B backbone, we additionally report results using an alternative LLM backbone in Appendix B.

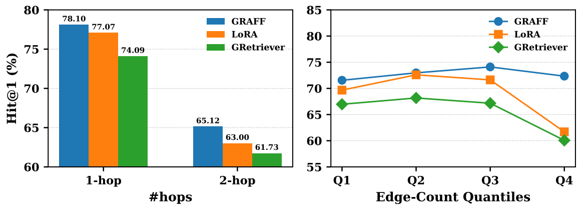

Figure 3: Comparison of Hit@1 results among GRAFF, LoRA and G-Retriever across two settings: (**left**) query complexity measured by hop count; and (**right**) graph size measured by four edge-count quantiles.

3.2.1 Multi-Hop Question Analysis

We analyze the multi-hop reasoning capability of GRAFF across datasets with varying graph complexity. WebQSP provides explicit hop annotations, consisting of approximately 65% one-hop and 35% two-hop questions, making it well-suited for controlled hop-based analysis. In contrast, ExplaGraphs (Saha et al., 2021) exhibits substantially more complex reasoning structures, with 58% of questions involving non-linear explanation graphs and an average graph depth of approximately four. Similarly, while explicit hop statistics are not reported for the Scene Graph dataset, prior work (Khan et al., 2025) shows that stepwise reasoning with up to $T = 3$ iterations is required to achieve optimal performance, indicating reasoning beyond simple two-hop paths.

Given the clarity of hop supervision in WebQSP, we focus our quantitative hop-wise evaluation on this dataset for interpretability. Figure 3 (left) presents the performance breakdown based on hop count on WebQSP.

For one-hop questions, GRAFF achieves a 1.3% improvement over the strongest baseline (LoRA), demonstrating its effectiveness on straightforward queries. The performance gap widens significantly for two-hop questions, where GRAFF outperforms the best baseline by 3.4%. This result is particularly notable, as multi-hop reasoning requires integrating fine-grained information across multiple nodes and relations, posing challenges for graph-pooling-based approaches such as G-Retriever. The strong gains on WebQSP, together with competitive performance on ExplaGraphs and Scene Graph datasets, indicate that GRAFF effectively generalizes beyond two-hop and linear reasoning structures, even though we restrict explicit hop-wise analysis to WebQSP for simplicity.

3.2.2 Effect of Graph Size on Performance

We further analyze how model performance varies with different levels of edge density in the retrieved graphs. To this end, we partition the WebQSP dataset into four quantile-based buckets (Q1–Q4) based on the number of edges in each graph, where Q1 contains graphs with the fewest edges (2–6) and Q4 contains the most densely connected graphs (18–25 edges). Figure 3 (right) reports the Hit@1 performance for GRAFF, LoRA, and G-Retriever across these buckets. GRAFF performs consistently well across all quantiles and particularly achieves the largest gains in Q3 and Q4, which correspond to structurally more complex graphs. These results highlight GRAFF 's robustness and ability to scale effectively with increasing graph complexity.

3.3 Hyper-parameter Study

Next, we examine how the placement of GRAFF at a specific decoder layer D and the depth of the GAT module impact the model's performance.

3.3.1 Placement of GRAFF

To analyze the effect of layer placement, we experiment with inserting GRAFF at different decoder layers. The results, shown in Figure 4 (left), indicate that placing GRAFF in the 6th-8th decoder layers yields the best performance, with accuracy peaking in this range. This suggests that mid-to-deep layers capture richer semantic and structural information, which is crucial for effective graph-based reasoning. However, inserting GRAFF closer to the output prediction head leads to a consistent decline in accuracy. We hypothesize that, at such top-layer placements, the fusion of graph-structured information into the LLM is insufficient.

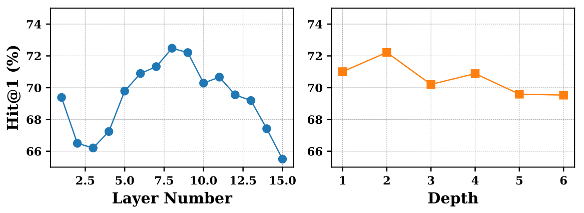

Figure 4: Impact of GRAFF placement and GAT module depth on model accuracy on WebQSP.

3.3.2 GAT Depth

We also analyze the impact of GAT module depth by varying the number of layers from 1 to 6, with results shown in Figure 4 (right). Our observations indicate that a lightweight GAT module (1-3 layers) achieves the best performance, with accuracy peaking at 2 layers. However, increasing depth beyond this results in a decline, likely due to over-smoothing of GNN (Rusch et al., 2023), where node representations become too homogeneous.

3.4 Ablation Study

In this section, we present the results of our ablation studies. We train two variants of GRAFF: the first, where GRAFF is placed between all decoder layers, and the second, where only a single graph input, processed by GRAFF, is passed through the model. We also train variants of GRAFF by removing either structural or textual channel inputs.

3.4.1 Fusion in One vs. All Layers

In this study, we evaluate the performance of adding the GAT component to all layers compared to adding it to a single layer. The results are presented in Table 3. The results indicate that applying GRAFF to all layers leads to a significant performance drop. This decline occurs because inserting GAT modules between all decoder layers substantially increases the number of trainable parameters, even when efforts are made to keep the parameter count within a comparable range. The resulting model becomes harder to train effectively given the limited amount of training data, leading to instability. Additionally, the all-layers approach may cause the adapter to capture unnecessary information from both shallow and deep layers, potentially confusing the model and impairing learning.

Model	ExplaGraph	SceneGraph
LoRA	88.98	85.27
GRAFF- All Layer	89.96	85.66
GRAFF- Specific Layer	92.50	90.29

Table 3: Ablation Results of Hit@1 for fusion of GRAFF in one vs. all LLM decoder layers on two datasets: ExplaGraph and SceneGraph.

3.4.2 Dual vs. Single-Channel Graph Input

To evaluate the effectiveness of our GRAFF, we compare it against two single-channel variants of our method: (1) a model that uses only structural information of the retrieved graph, encoded by a tunable GAT; and (2) a model that uses only textual information of the retrieved graph alongside with a list of tunable prompt tokens. As shown in Table 4, the dual-channel model outperforms the best-performing single-channel variant by a substantial margin of **24.82%**. The results demonstrate that the dual-channel graph input enables more effective integration of both structural and textual information from a text-attributed graph, leading to significantly improved reasoning capabilities.

Model	ExplaGraph	SceneGraph
Dual Channel Graph Input	92.5	90.2
Structure-Only Input	72.9	64.6
Text-Only Input	60.2	58.3

Table 4: Ablation Results of Hit@1 for single- vs. double-channel inputs on ExplaGraph and SceneGraph.

4 Related Work

4.1 Knowledge Graph Question Answering

Knowledge Graph Question Answering (KGQA) aims to answer natural language questions by leveraging structured information from knowledge graphs (KGs), often by translating questions into executable queries to retrieve relevant information from KGs (Lan et al., 2021). Recent methods based on pre-trained language models (PLMs) such as BERT have improved multi-hop KGQA by jointly modeling subgraph retrieval and reasoning. For instance, UniKGQA (Jiang et al., 2023b) integrates these two stages into a single unified framework, while ReasoningLM (Jiang et al., 2023a) integrates KG structure into self-attention, eliminating the need for learning external GNNs. In contrast, GreaseLM (Zhang et al., 2022) fuses GNNs with encoder-only language models such as RoBERTa via modality-interaction layers, enabling fine-grained cross-modal reasoning for classification-style question answering.

LLM-based approaches further advance KGQA by enabling more faithful and interpretable reasoning. ChatKBQA (Luo et al., 2024a) employs a generate-then-retrieve strategy to better align question intent with KG grounding. Reasoning-on-Graphs (Luo et al., 2024b) improves transparency through explicit relation path planning, while Think-on-Graph (Sun et al., 2024) conceptualizes the LLM as an agent that iteratively explores the KG via dynamic path expansion.

Despite their effectiveness, these methods often rely on structured query generation or symbolic reasoning components, which can be brittle and complex. In contrast, GRAFF introduces a lightweight, decoder-only architecture that integrates latent graph structure directly into the LLM. This enables robust, end-to-end reasoning over text-attributed graph inputs and generalizes beyond traditional KGQA settings.

4.2 Graph Enhanced LLMs

To improve factual consistency and mitigate hallucinations in large language models, recent research has explored integrating structured graph knowledge directly into LLMs. Figure 1 illustrates the three main categories of architectural strategies developed for this purpose.

The first category, *verbalizer-based methods*, transforms graph structure into natural language prompts, enabling off-the-shelf LLMs to operate over textualized representations of the graph (Mavromatis and Karypis, 2024; Fatemi et al., 2023) without further fine-tuning. While simple and training-free, the verbalization process often loses complex, structured information such as node density degrees, and often struggles to capture fine-grained relational semantics among nodes.

The second category, *graph pooling based methods*, encodes the entire graph into a single latent embedding using a graph neural network, which is then prepended as input to the LLM (Perozzi et al., 2024; He et al., 2024; Tian et al., 2023; Hu et al., 2024). Although more structure-aware than verbalization, this approach compresses the rich relational information of variable-sized graphs into a single token, often leading to significant information loss. Furthermore, the combination of this coarse graph-pooled token with fine-grained textual tokens within the LLM often results in representation misalignment.

In contrast, GRAFF adopts a fine-grained fusion strategy with a modular architecture. It disentangles structural and textual inputs via a novel dual-channel mechanism and integrates a lightweight graph attention network at an intermediate decoder layer. This design enables node-level graph representation, promotes semantic alignment with detailed text tokens within the LLM, and supports more robust and generalizable reasoning across diverse graph-centric tasks.

5 Conclusion

In this work, we present GRAFF, a novel graph fusion approach that enhances the LLMs' graph reasoning capabilities by integrating structured knowledge through a lightweight, multi-hop structure-aware module. Our method introduces a dual-channel graph input strategy to separate structural and textual components of text-attributed graphs, enabling more effective graph encoding, and incorporates a graph attention network module into intermediate decoder layers of LLMs for deep fusion. Extensive experiments demonstrate that GRAFF significantly improves LLMs' graph understanding, outperforming the strongest baseline by 10.14% across four datasets.

6 Limitation

While GRAFF achieves promising results, it also presents several limitations that warrant further investigation. A primary constraint stems from the Dual-Channel Graph Input mechanism, which preprocesses the input graph into two separate channels—one capturing structural information and the other encoding semantic content. Although this design enhances the integration of structured knowledge, it substantially increases the number of input tokens. As the graph size grows, the token count scales accordingly, resulting in higher computational costs and memory consumption. This issue becomes especially pronounced when handling large or complex graphs, potentially hindering the model's efficiency in long-context scenarios. In this paper, we demonstrate the effectiveness of GRAFF on small-sized sub-graphs retrieved from a much larger graph. Future work should explore strategies for improving the token efficiency of graph representations to address these scalability challenges.

Acknowledgment

Research reported in this publication was partially supported by the National Science Foundation under Award Number IIS-2449768. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Science Foundation.

References

Jinheon Baek, Alham Fikri Aji, and Amir Saffari. 2023. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. *Preprint*, arXiv:2306.04136.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Bhuwan Dhingra, Jeremy R. Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W. Cohen. 2022. Time-aware language models as temporal knowledge bases. *Transactions of the Association for Computational Linguistics*, 10:257–273.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and fate: Limits of transformers on compositionality. *Preprint*, arXiv:2305.18654.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. Talk like a graph: Encoding graphs for large language models. *Preprint*, arXiv:2310.04560.

Jiayan Guo, Lun Du, and Hengyu Liu. 2023. Gpt4graph: Can large language models understand graph structured data ? an empirical evaluation and benchmarking. *ArXiv*, abs/2305.15066.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. 2023. Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning. *arXiv preprint arXiv:2305.19523*.

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

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021a. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021b. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. 2024. Grag: Graph retrieval-augmented generation. *Preprint*, arXiv:2405.16506.

Drew A. Hudson and Christopher D. Manning. 2019. Gqa: A new dataset for real-world visual reasoning and compositional question answering. *Preprint*, arXiv:1902.09506.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yaliang Li, and Ji-Rong Wen. 2023a. Reasoninglm: Enabling structural subgraph reasoning in pre-trained language models for question answering over knowledge graph. *Preprint*, arXiv:2401.00158.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. 2023b. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. *Preprint*, arXiv:2212.00959.

Charles Jin and Martin Rinard. Emergent representations of program semantics in language models trained on programs. In *Forty-first International Conference on Machine Learning*.

Jungo Kasai, Keisuke Sakaguchi, Yoichi Takahashi, Ronan Le Bras, Akari Asai, Xinyan Yu, Dragomir Radev, Noah A. Smith, Yejin Choi, and Kentaro Inui. 2024. Realtime qa: What's the answer right now? *Preprint*, arXiv:2207.13332.

Muhammad Junaid Khan, Adil Masood Siddiqui, Hamid Saeed Khan, and Jaleed Khan. 2025. Enhancing visual question answering with common sense knowledge: a data-driven neurosymbolic graph routing approach. *International Journal of Data Science and Analytics*, 20(7):6391–6406.

Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. A survey on complex knowledge base question answering: Methods, challenges and solutions. *Preprint*, arXiv:2105.11644.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *Preprint*, arXiv:2101.00190.

LLaMATeam. 2024. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. *Preprint*, arXiv:1711.05101.

Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, Yifan Zhu, and Anh Tuan Luu. 2024a. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, page 2039–2056. Association for Computational Linguistics.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024b. Reasoning on graphs: Faithful and interpretable large language model reasoning. *Preprint*, arXiv:2310.01061.

Costas Mavromatis and George Karypis. 2024. Gnn-rag: Graph neural retrieval for large language model reasoning. *Preprint*, arXiv:2405.20139.

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. *arXiv preprint arXiv:2402.05862*.

Alec Radford. 2018. Improving language understanding by generative pre-training.

T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. 2023. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*.

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

Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval augmentation reduces hallucination in conversation. *arXiv preprint arXiv:2104.07567*.

Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. 2023. Gpt-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems. *Preprint*, arXiv:2310.12397.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *Preprint*, arXiv:2307.07697.

Wen tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Annual Meeting of the Association for Computational Linguistics*.

Shiyu Tian, Yangyang Luo, Tianze Xu, Caixia Yuan, Huixing Jiang, Chen Wei, and Xiaojie Wang. 2024. KG-adapter: Enabling knowledge graph integration in large language models through parameter-efficient fine-tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3813–3828, Bangkok, Thailand. Association for Computational Linguistics.

Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V. Chawla, and Panpan Xu. 2023. Graph neural prompting with large language models. *Preprint*, arXiv:2309.15427.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2024a. Can language models solve graph problems in natural language? *Preprint*, arXiv:2305.10037.

Ruida Wang, Raymond Chi-Wing Wong, and Weile Tan. 2023a. Sr-predictao: Session-based recommendation with high-capability predictor add-on. *arXiv preprint arXiv:2309.12218*.

Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. 2024b. Theoremllama: Transforming general-purpose llms into lean4 experts. *arXiv preprint arXiv:2407.03203*.

Ruida Wang, Wangchunshu Zhou, and Mrinmaya Sachan. 2023b. Let's synthesize step by step: Iterative dataset synthesis with large language models by extrapolating errors from small models. *arXiv preprint arXiv:2310.13671*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Xikun Zhang, Antoine Bosselut, Michihiro Yasunaga, Hongyu Ren, Percy Liang, Christopher D. Manning, and Jure Leskovec. 2022. Greaselm: Graph reasoning enhanced language models for question answering. *Preprint*, arXiv:2201.08860.

A Statistical Significance Analysis

To assess the robustness and statistical significance of our results, we report performance variability and conduct hypothesis testing across all datasets. Each experiment is repeated over four random seeds, and results are reported as mean performance along with standard deviation. We further perform two-sided Welch's t-tests comparing GRAFF against the strongest competing baseline on each dataset.

Table 5 summarizes the results. Across all benchmarks, GRAFF consistently outperforms the best baseline with statistically significant improvements ($p < 0.05$), indicating that the observed gains are unlikely to arise from random variation.

Dataset	**Ours**	**Baseline**	**p-value**	**Sig.**
WebQuestions	72.2 ± 0.55	71.1 ± 0.52	0.0444	Yes
ExplaGraphs	92.5 ± 0.70	88.9 ± 1.54	0.00002	Yes
Scene Graph	90.2 ± 0.22	85.3 ± 2.78	0.0181	Yes
Synthetic	79.4 ± 1.73	58.3 ± 1.17	0.0016	Yes

Table 5: Statistical significance analysis comparing GRAFF with the strongest competing baseline on each dataset. Results are reported as mean ± standard deviation over four random seeds. All improvements are statistically significant under Welch's t-test ($p < 0.05$).

B Results on a Different Backbone

In addition to our primary experiments using LLaMA-3.2-3B, we report results on an alternative LLM backbone to assess whether the effectiveness of GRAFF is specific to a particular architecture. Specifically, we evaluate GRAFF using Qwen-2.5-3B, which has a comparable parameter scale but differs in architecture and pretraining.

We conduct experiments on WebQSP and ExplaGraphs and compare GRAFF against LoRA and G-Retriever under the same experimental setup. Table 6 presents the results. On WebQSP, GRAFF outperforms the strongest baseline by 3.3 points (5.3% relative improvement), while on ExplaGraphs it achieves an 8.0-point gain (10.1% relative improvement).

Method	**WebQSP**	**ExplaGraphs**
LoRA	60.5	79.7
G-Retriever	61.9	79.0
GRAFF	**65.2**	**87.0**

Table 6: Performance comparison on Qwen-2.5-3B across different methods.