GraphInsight: Unlocking Insights in Large Language Models for Graph Structure Understanding

Yukun Cao[1,2*]   **Shuo Han**[1,2*]   **Zengyi Gao**[1,2]
Zezhong Ding[1,2]   **Xike Xie**[1,2†]   **S. Kevin Zhou**[1,3]
[1] University of Science and Technology of China, China
[2] Data Darkness Lab, MIRACLE Center, USTC, China
[3] MIRACLE Center, Suzhou Institute for Advance Research, USTC, China
{ykcho, shuo.han, gzy02, zezhongding}@mail.ustc.edu.cn
{xkxie, skevinzhou}@ustc.edu.cn

Abstract

Although Large Language Models (LLMs) have demonstrated potential in processing graphs, they struggle with comprehending graphical structure information through prompts of graph description sequences, especially as the graph size increases. We attribute this challenge to the uneven memory performance of LLMs across different positions in graph description sequences, known as "Positional bias". To address this, we propose **GraphInsight**, a novel framework aimed at improving LLMs' comprehension of both macro- and micro-level graphical information. GraphInsight is grounded in two key strategies: 1) placing critical graphical information in positions where LLMs exhibit stronger memory performance, and 2) investigating a lightweight external knowledge base for regions with weaker memory performance, inspired by retrieval-augmented generation (RAG). Moreover, GraphInsight explores integrating these two strategies into LLM agent processes for composite graph tasks that require multi-step reasoning. Extensive empirical studies on benchmarks with a wide range of evaluation tasks show that GraphInsight significantly outperforms all other graph description methods (e.g., prompting techniques and reordering strategies) in understanding graph structures of varying sizes.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in natural language processing (NLP) (Shen et al., 2024; Naveed et al., 2023; Ge et al., 2023), enabling their initial applications across various data domains, such as graphs (Chen et al., 2024b; Wang et al., 2024c; Besta et al., 2024), time-series data (Jin et al., 2023; Yu et al., 2023), tabular data (Sui et al., 2024;
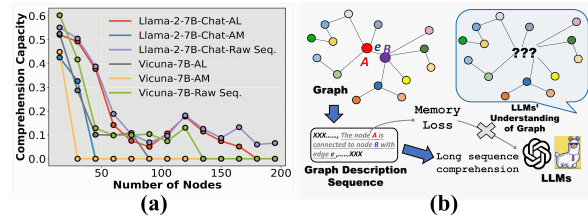

Figure 1: Capabilities of LLMs on Graph Structure Understanding

Hegselmann et al., 2023), and other structured or semi-structured data types (Ko et al., 2024; Perozzi et al., 2024; Deng et al., 2024). Among these domains, leveraging LLMs to tackle applications involving graphs has emerged as a burgeoning field of research, as graphs represent fundamental structures that capture intricate relationships and interactions in the real world (Wang et al., 2021; Xu, 2021). For example, Fatemi et al. have explored the potential of LLMs by converting various types of graphs, such as knowledge graphs (Baek et al., 2023; Pan et al., 2024) and social network graphs (Santra, 2024; Babic, 2023), into natural language descriptions, thereby enabling LLMs to perform question-answering tasks related to these graphs.

A **key observation** is that enhancing LLM performance in graph-related applications depends critically on LLMs' ability to comprehend graph structures through natural language descriptions. Existing studies (Shang and Huang, 2024; Li et al., 2023) primarily utilize two direct methods to transform graphs into text inputs for LLMs: the *structural format transforming*, such as adjacency matrices (termed as *AM*) or lists (termed as *AL*), and the *sequential format transforming*, such as edge-by-edge descriptions (termed as *Raw Seq*). However, extensive empirical studies (Yuan et al., 2024) have shown that LLMs face significant challenges in understanding and reasoning about graph structures using current graph transformation methods, espe-

12096

cially as graph size increases, leading to a "comprehension collapse". As shown in Figure 1(a), several common LLMs perform poorly on graph structure understanding tasks (see benchmarks in Section 5.1), and their comprehension declines sharply as the graph size increases, ultimately leading to complete failure.

In this paper, we focus on enhancing the ability of LLMs to understand graph structures by developing new graph-transforming methods and other novel techniques. To achieve this, we first analyze the inherent nature of graph understanding tasks in LLMs and the reasons behind the poor performance of existing methods. At its core, the challenge of LLM understanding graph structures can be viewed as a *stringent long sequence comprehension problem* (Liu et al., 2024; Wang et al., 2024b). This challenge arises primarily due to two factors, as illustrated in Figure 1(b). First, graph structural information is conveyed to LLMs solely through language descriptions. As the graph size increases, their descriptions become longer, challenging LLMs' ability to comprehend long-sequence inputs. Second, to accurately understand the graph's structure, LLMs must retain every detail in the natural language description. Any memory lapse, especially regarding critical nodes or edges (e.g., central nodes, bridging edges), notably impairs the LLMs' ability to infer the correct structure from the description.

However, the challenge is further exacerbated by the uneven memory performance of LLMs across different positions within long sequences—a phenomenon known as *"positional bias"* (Tang et al., 2023; Hsieh et al., 2024; Zhang et al., 2024a), which mainly stems from limitations in the attention mechanism (Xiao et al., 2024) and the internal memory capacity (An et al., 2024) of LLMs. For example, as shown in Figure 2(a), many studies indicate that LLMs generally perform better at understanding the head and tail of sequences (i.e., *strong memory regions*), with a noticeable dip in performance in middle parts (i.e., *weak memory regions*), termed as *"lost-in-the-middle"* (Liu et al., 2024). Consequently, LLMs struggle to meet the stringent requirements for understanding graph structures from descriptive sequences, due to they always lose some positions in the sequence. This issue becomes particularly pronounced as the graph size increases (i.e., sequence lengthens), leading to "comprehension collapse", as mentioned in Figure 2(a).
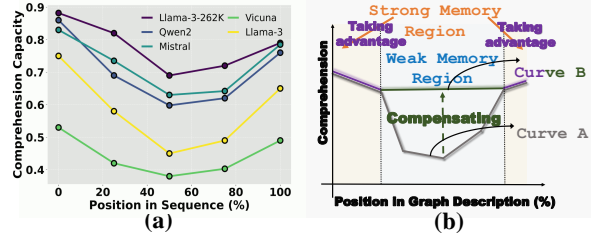

Figure 2: Analysis on Positional bias of LLMs

Thus, improving LLMs' ability to comprehend graph structures relies on improving their ability to retain information across different sequence positions in graph descriptions, thereby reducing the impact of positional bias. To this end, our work is grounded in two high-level insights: **1) Taking advantage of the Strong Memory Regions:** For sequence positions where LLMs demonstrate strong memory capabilities, we strategically place descriptions of critical nodes or edges in these locations. **2) Compensating for the Weak Memory Regions:** We construct a lightweight external knowledge base for positions where LLMs show weak memory performance, inspired by the concept of retrieval-augmented generation (RAG). As illustrated in Figure 2(b), following the above "*different horses for different courses*" principle, the LLMs' comprehension of graph description sequences improves from Curve A to Curve B.

Starting from the premises, we propose *GraphInsight*, a framework aimed at enhancing the ability of LLMs in graph comprehension tasks through a series of innovative optimization techniques applied to graph description sequences. Our framework incorporates two key techniques tailored for macro-level and micro-level graph understanding tasks, respectively. Specifically, For Macro-level tasks (i.e., issues related to global graph structures), we reconstruct the original graph description sequence according to the relative importance of local graph structures, ensuring that critical graph components are aligned with the LLMs' strong memory regions, thereby improving the overall memory retention and understanding of the graph. For Micro-level tasks (i.e., issues related to local detailed graph structures), we build a lightweight knowledge base for the graph description sequences corresponding to the LLMs' weak memory regions, enhancing the LLMs' comprehension in fine-grained graph tasks by enabling efficient retrieval of relevant information. Moreover, we explore integrating two techniques into LLMs agent processes to tackle

composite graph understanding tasks that require multi-step reasoning, which involves multiple interconnected micro-level tasks.

The main contributions of this paper are listed as follows:

- We conduct a pioneering analysis of the fundamental issues, challenges, and high-level solutions for LLMs in understanding graph structures based on natural language graph descriptions.

- To the best of our knowledge, from the perspective of the "positional bias" inherent in LLMs, we propose the first framework, **GraphInsight**, designed to enhance LLMs' ability to understand graph structures, integrating a series of innovative techniques.

- We introduce **GraphSQA**, a benchmark designed to evaluate LLMs' ability to understand and reason about graph structures across two levels of tasks, from macro-level to micro-level comprehension.

- Extensive empirical studies across various evaluation tasks and LLMs demonstrate the effectiveness and superiority of our framework.

2 Related Work

2.1 LLMs' Understanding of Graph Structures Through Description Sequences

Research on the ability of LLMs to understand graph structures by inputting natural language descriptions of graphs into these models has become an emerging area of study. Existing research has focused on developing evaluation benchmarks for graph structure understanding tasks and analyzing their results. For instance, Wang et al. and Guo et al. conducted preliminary empirical assessments of LLMs on coarse-grained graph structure tasks, indicating the nascent stage of this field. Subsequent benchmarks, such as GraphEval2000 (Wu et al., 2024b), GraphArena (Tang et al., 2024), GraphInstruct (Luo et al., 2024), and GraCoRe (Yuan et al., 2024), have shifted focus to fine-grained tasks, including graph theory problems. Ge et al. also explored the impact of the sequence of graph descriptions on LLMs comprehension, though their study was limited to a few tasks and lacked deep analysis. Unlike previous work, GraphInsight aims to enhance LLMs' capabilities in graph structure understanding tasks.

2.2 Positional Bias in Long Sequences for LLMs

Extensive research (Tang et al., 2023; Hsieh et al., 2024) has shown that the position of inputs and the order of answer choices can significantly impact LLMs' performance and output generation (i.e., "positional bias"). Liu et al. analyzed how LLMs utilize information in long input sequences and identified the "lost-in-the-middle" problem. Additionally, Xiao et al. discovered that attention scores tend to be biased towards initial tokens of input sequences due to the Softmax operation.

Some recent works have attempted to mitigate the positional bias issue in LLMs. For instance, Zhang et al. and Hsieh et al. introduced attention instructions to guide the LLMs to focus on specific sequence segments. An et al. proposed training with a synthesized long-sequence QA dataset to mitigate the "lost-in-the-middle" problem. Wu et al. proposed interpolating positional encodings via index adjustment, requiring fine-tuning within the pre-trained context window. Note that these broad strategies are not specifically designed for graphs and do not directly apply to graph description sequences; instead, they are generally implemented during the training/fine-tuning phases of LLMs. Therefore, these methods are orthogonal to GraphInsight and could further enhance its performance if computational resources permit.

3 Preliminaries

In this section, we define the basic format of graph description sequences, outline our assessment of LLM capabilities on these sequences, and introduce two levels of graph understanding tasks as the foundation of our framework.

There are two primary methods for converting graphs into description sequences for LLMs: **1)** structural format transforming (e.g., adjacency matrices/lists) and **2)** sequential format transforming (e.g., edge-by-edge descriptions). Since empirical studies (see Section 5.2) show that sequential format is more conducive to LLMs' understanding and can seamlessly integrate additional semantic information (e.g., node and edge attributes, labels, etc.), our framework focuses on optimizing graph description sequences under this format. Therefore, for a given graph G, we define the standard graph description sequences $\mathcal{T}$ as follows:

Definition 1 (Sequential Format Graph Description). *For a graph G, consisting of $V = \{v_1, v_2, ...\}$ and*

$E = \{e_{ij}\}$. *The sequential format transforming is to transform G to $\mathcal{T}$:*

> This graph is described as follows:
>   Node v_i is connected to node v_j by edge e_{ij} with weight w_{ij}; . . .

The description $\mathcal{T}$ consists of the descriptions of all edges $e_{ij} \in E$. Based on Definition 1, the LLMs' ability to understand $\mathcal{T}$ can be quantified as shown in Definition 2.

Definition 2 (LLMs' Capacity for Graph Understanding). *The capacity of LLM for graph understanding can be quantified by $C_{LLM}(\mathcal{T}) = \int C(\mathcal{T}, p) \, dp$, where $C(\mathcal{T}, p)$ is the comprehension ability at a specific position p within $\mathcal{T}$.*

Thus, the $C(\mathcal{T}, p)$ at position p, can be modeled as following a specific distribution $\Psi(p)$:

$$C(\mathcal{T}, p) \sim \Psi(p) \tag{1}$$

Here, $\Psi(p)$ denotes the positional bias curve, which represents the distribution of an LLM's inherent comprehension at position p. As mentioned in Figure 2, $\Psi(p)$ typically displays stronger comprehension at the head and tail of a sequence, with weaker comprehension in the middle, forming a U-shape curve. In this paper, we follow the above assumption and define the positions corresponding to the head $\alpha\%$ and tail $\beta\%$ of the sequence as strong memory regions, and the rest as weak memory regions.

Next, we introduce two levels of graph understanding tasks, as shown in Definition 3.

Definition 3 (Graph Understanding Tasks). *Integrating on existing benchmarks (Wang et al., 2024a; Yuan et al., 2024), we categorize LLM graph structure understanding into two levels: 1) **Macro-level**, involving coarse-grained reasoning related to the overall graph structure (e.g., node counting, connectivity detection, cycle detection); 2) **Micro-level**, focusing on fine-grained reasoning related to local structures (e.g., direct connection detection, node degree calculation, leaf node identification, neighbor recognition). Moreover, some composite graph understanding tasks may require multiple micro-level reasoning steps (e.g., complete subgraph verification, and third-order neighbor identification).*

Following Definition 3, we build the GraphSQA benchmark (see Section 5.1) to comprehensively and fairly evaluate LLMs' performance in graph structure understanding.
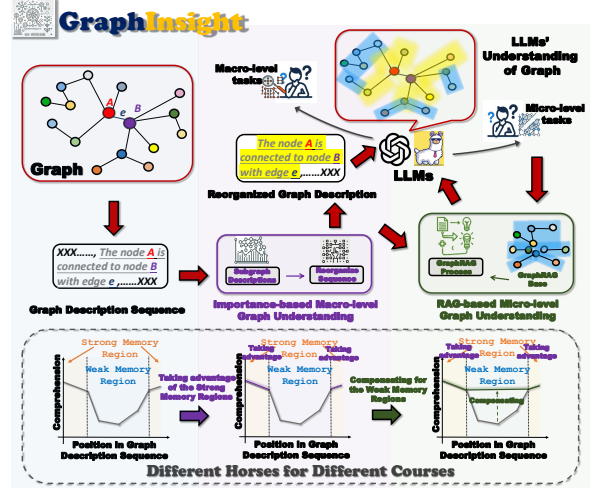

Figure 3: Framework of GraphInsight

4 Methodology

4.1 Overview

In this section, we present the *GraphInsight* framework, as shown in Figure 3, which consists of two key techniques that enhance LLMs' graph comprehension.

4.2 Importance-based Macro-level Graph Understanding

For macro-level graph understanding tasks, guiding LLMs to focus on the memory and comprehension of key elements within the graph structure can potentially enhance their reasoning performance. Thus, the intuitive idea is to align the LLMs' comprehension distribution across different sequence positions (i.e., $\Psi(p)$) with the importance distribution of these positions in the graph description sequence. Let $\Phi(p)$ represent the importance of each corresponding position p in the sequence $\mathcal{T}$ relative to the graph structure. In this context, our objective is:

$$\operatorname{argmin}_{\Psi} D_{\text{KL}} \left(\Phi(p) \,\|\, \Psi(p) \right), \quad \text{for all } p \in \mathcal{T} \tag{2}$$

, where D_{KL} represents the Kullback-Leibler divergence, quantifying the alignment between $\Phi(p)$ and $\Psi(p)$.

From theory to practice, we need to address two key challenges: 1) defining and quantifying $\Phi(p)$; and 2) aligning $\Phi(p)$ with $\Psi(p)$. Given that in this study we assume the comprehension distribution of LLMs follows a U-shaped curve[1], we adopt a

---

[1]Advancements in pre-training techniques and corpus optimization may alter the exact shape of this curve. Nevertheless, the variation in LLM comprehension across different sequence positions is likely to persist. Our framework is not restricted to

Algorithm 1 Importance-based Description Reorganization

Require: Graph $G = (V, E)$ and corresponding description $\mathcal{T}$, $\lambda = 0.85$, Memory regions $\alpha\%$, $\beta\%$
Ensure: Reorganized sequence $\hat{\mathcal{T}}$
1: **1. Compute PageRank:**
2: **for** each $v \in V$ **do**
3:    Calculate PR(v) based on Equation 3
4: **end for**
5: Sort nodes by PR(v) in descending order, and get sort list V_s
6: **2. Get Subgraph Descriptions :**
7: Initialize $E_{\text{used}} = \emptyset$, $\mathcal{T}_s = \emptyset$
8: **for** each v_{c_i} in sorted list V_s **do**
9:    Initial subgraph $G_i : \{V_i, E_i\}$
10:    **for** each edge (v_{c_i}, u) where $u \in \text{Neighbors}(v_{c_i})$ **do**
11:       **if** $(v_{c_i}, u) \notin E_{\text{used}}$ and Degree(v_{c_i}) > Degree(u) **then**
12:          $E_i \leftarrow E_i \cup (v_{c_i}, u)$, $V_i \leftarrow V_i \cup u$
13:          $E_{used} \leftarrow E_{used} \cup (v_{c_i}, u)$
14:       **end if**
15:       $t_i = $ Description of G_i according Definition 1.
16:       **Add** t_i in $\mathcal{T}_s$ with importance $\mathcal{I}(t_i) = \text{PR}(v_{c_i})$
17:    **end for**
18: **end for**
19: **3. Reorganize Graph Description Sequence:**
20: Sort $\mathcal{T}_s$ by $\mathcal{I}(t_i)$, reorganize sequence according Equation 4
21: **return** $\hat{\mathcal{T}}$

straightforward approach to achieve this alignment. Overall, we decompose the graph description $\mathcal{T}$ into a series of mutually exclusive subgraph descriptions $\mathcal{T}_s : \{t_1, t_2, ...\}$, with the importance of each subgraph $\mathcal{I}(t_i)$ measured by the PageRank score (calculated over the entire graph) of its highest-degree node. Finally, we reorder these subgraph descriptions based on their importance and reorganize them within the strong memory regions of the LLMs (i.e., the head and tail of the graph description), resulting in a new graph description sequence $\hat{\mathcal{T}}$. The detailed procedure for this Importance-based Description Reorganization is outlined in Algorithm 1.

Specifically, for the graph description decomposition, given a graph G and its corresponding graph description sequence $\mathcal{T}$, we first iteratively calculate the PageRank score $\text{PR}(v)$[2] for each node $v \in V$ in G as follows:

$$\text{PR}(v) = \lambda \sum_{u \in \text{InNb}(v)} \frac{\text{PR}(u)}{\text{OutDeg}(u)} + (1 - \lambda)\frac{1}{|V|} \tag{3}$$

, where λ is the damping factor, typically set to 0.85, $\text{InNb}(v)$ represents the set of nodes with

a U-shaped curve; once the precise curve shape of a particular LLM is empirically estimated, our framework can be easily adjusted and adapted.

[2]PR can be replaced with others describe the importance of graph structures. Designing graph structure importance remains an open problem (Besta et al., 2023; Liu and Gao, 2023; Geng et al., 2022), and is orthogonal to our work.

edges directed towards v, and $\text{OutDeg}(u)$ is the number of edges leaving node u.

After calculating the PageRank scores, we sort the nodes in descending order. Starting from the node with the highest score, we iteratively construct subgraphs centered on each node, including directly connected neighbors with lower degrees, provided the connecting edges haven't been used in other subgraphs. This ensures that each edge is only included in one subgraph, guaranteeing that the total length of the merged subgraph descriptions equals the original graph description sequence. This is because, as per Definition 1, we describe the graph edge by edge. The descriptions for these subgraphs $G_1, G_2, \ldots, G_k$ are denoted as $t_1, t_2, \ldots, t_k$. Then, each subgraph description t_i is formed by combining the descriptions of all edges within the subgraph G_i, according to the graph description defined previously in Definition 1. The importance $\mathcal{I}(t_i)$ of each subgraph description t_i is defined as the PageRank score of its central node v_{c_i}: $\mathcal{I}(t_i) = \text{PR}(v_{c_i})$.

Finally, the subgraph descriptions are reordered based on their importance and organized within the strong memory regions of the LLM, as defined in Definition 2. Specifically, the head $\alpha\%$ and the tail $\beta\%$ of the graph description sequence are designated as the strong memory regions. Consequently, the most important subgraph descriptions are prioritized and placed in these regions, while the remaining descriptions, sorted by importance, occupy the middle and are considered the weak memory region. The final graph description sequence $\hat{\mathcal{T}}$ is:

$$\hat{\mathcal{T}} = \mathcal{T}[: \alpha\%] \cup \mathcal{T}[\alpha\% : (100-\beta)\%] \cup \mathcal{T}[(100-\beta)\% :] \tag{4}$$

, where $\mathcal{T}[: \alpha\%]$ and $\mathcal{T}[(100 - \beta)\% :]$ contain the most important subgraph descriptions.

4.3 RAG-based Micro-level Graph Understanding

For micro-level graph understanding tasks, when these tasks involve information about nodes or edges within graph description sequences that correspond to the weak memory regions of the LLMs, the LLMs are inevitably prone to forgetting this information, thereby failing to generate accurate responses. To address this, inspired by the commonly used Retrieval-Augmented Generation (RAG) idea for enhancing the LLM reasoning capabilities, we propose establishing a lightweight, optional-scale RAG knowledge base for the nodes and edges in

weak memory regions. This knowledge base, powered by RAG algorithms, retrieves relevant node and edge information for specific graph understanding tasks, thereby enhancing the comprehension of the LLMs.

Next, we introduce the construction of our framework's RAG knowledge base, termed "*GraphRAG base*", and the corresponding RAG process, termed the "*GraphRAG process*". Note that existing RAG techniques (Ghosh et al., 2024; Rorseth et al., 2024; Sojitra et al., 2024; Deng et al., 2023; Das et al., 2024; Cao et al., 2024; Gao et al., 2025) and optimizations are orthogonal to our framework and can further enhance RAG quality, but here we focus only on the most basic RAG methods.

GraphRAG Base. Conventional RAG base typically require substantial storage and rely on extensive structured or unstructured data (e.g., documents, knowledge graphs). In contrast, our GraphRAG base, denoted as $\mathcal{K}$, demands minimal storage overhead. Specifically, for the graph description sequence $\hat{\mathcal{T}}$ generated by the importance-based description reorganization, the nodes and edges of the subgraph structures corresponding to the weak memory regions within $\hat{\mathcal{T}}$ will be stored as the GraphRAG base. The proportion of stored subgraph structures denoted as $\gamma\%$, is adjustable. Since the subgraph structures corresponding to the weak memory regions have already been ranked by importance, we can conveniently select and store only the top $\gamma\%$ of these structures. Moreover, to facilitate efficient RAG retrieval, we store the node and edge information of the subgraph structures that need to be included in the GraphRAG base separately: $\mathcal{K} = \{\mathcal{K}_{node}, \mathcal{K}_{edge}\}$. Among them,

$$\mathcal{K}_{node} = \{(v, \deg(v)) \mid v \in V'\} \quad (5)$$

, where $V' \subseteq V$ represents the set of nodes in the GraphRAG base, and $\deg(v)$ is the degree of node v.

$$\mathcal{K}_{edge} = \{(u, v, w(u, v)) \mid (u, v) \in E'\} \quad (6)$$

, where $E' \subseteq E$ are the edges in the GraphRAG base, and $w(u, v)$ is the edge weight between nodes u and v.

GraphRAG Process. Based on the GraphRAG base $\mathcal{K}$, when augmented graph information is needed for a micro-level graph understanding task q, we first extract/identify all nodes from q by their names or identifiers. This can be accomplished using various entity recognition (Li et al., 2022) or semantic parsing models (Lewis et al., 2020; Zhu et al., 2024): $V_q = \text{EntityRecognition}(q)$, where V_q represents the set of node entities extracted from the task q. Next, based on the V_q, we perform retrieval operations on $\mathcal{K}_{edge}$. Specifically, for $\mathcal{K}_{node}$, we directly retrieve each node and its degree information based on the nodes in V_q, resulting in the augmented information set $\mathcal{K}_{\text{node}}^q$:

$$\mathcal{K}_{\text{node}}^q = \{(v, \deg(v)) \mid v \in V_q \cap V'\} \quad (7)$$

For $\mathcal{K}_{\text{edge}}$, we retrieve all edges information associated with the nodes in V_q, resulting in the augmented information set:

$$\mathcal{K}_{\text{edge}}^q = \{(u, v, w(u, v)) \mid u \in V_q \text{ or } v \in V_q, (u, v) \in E'\} \quad (8)$$

Finally, the two parts of the augmented information are organized into a prompt, which is then input into LLMs to assist in the reasoning process for the task q:

$$\text{Response}_q = \text{LLMs}\left(\text{Prompt}(\mathcal{K}_{\text{node}}^q, \mathcal{K}_{\text{edge}}^q, q)\right) \quad (9)$$

Composite Tasks. For composite tasks requiring multi-step reasoning, the common approach in LLMs is to design agents (Zhao et al., 2024; Kirk et al., 2024) that use multi-step prompts to aid reasoning. For example, identifying a node's third-order neighbors involves first finding its direct (first-order) neighbors, then their neighbors (second-order), and finally the neighbors of those (third-order). This task essentially consists of three micro-level graph understanding tasks.

GraphInsight framework incorporates two key techniques that can be seamlessly integrated into the agent-based processes of LLMs to enhance the performance of such multi-step reasoning tasks:

- Initially, during the LLMs agent process's inception phase, our framework's importance-based description reorganization method can be applied to the sequence of graph descriptions input into the LLMs. This enhances the LLMs' overall comprehension of the graph structure.

- Subsequently, in the multi-step reasoning phase of the LLMs agent process, our framework's GraphRAG method can provide LLMs with enriched information relevant to each step of the reasoning process, thereby improving the quality of the reasoning.

Table 1: Analysis on Macro- and Micro-level Tasks with Different Baseline Methods and Models.

Models	Tasks	Raw Seq.	GraphToken	Prompting Methods			Reordering Methods			Structural Methods		GraphInsight
				COT	FS	BAG	BFS	DFS	SP	AL	AM	
Mistral-7B	Overall	0.5476	0.2644	0.5315	0.4671	0.5145	0.5481	0.5482	0.5486	0.5320	0.1922	**0.6811**
	Macro	0.5222	0.3342	0.4323	0.5439	0.4249	0.5212	0.5205	0.5222	0.5001	0.1808	**0.5846**
	Micro	0.5635	0.2208	0.5935	0.4192	0.5705	0.5650	0.5655	0.5651	0.5519	0.1994	**0.7425**
Llama-3-8B	Overall	0.4513	0.3480	0.4504	0.4173	0.4358	0.4551	0.4591	0.4559	0.4048	0.1434	**0.6765**
	Macro	0.4379	0.3989	0.4187	0.3702	0.4060	0.4381	0.4376	0.4404	0.4336	0.1198	**0.5422**
	Micro	0.4597	0.3161	0.4702	0.4468	0.4545	0.4658	0.4726	0.4656	0.3869	0.1581	**0.7605**
Qwen2-7B	Overall	0.5640	0.3550	0.5663	0.5400	0.5677	0.5381	0.5290	0.5292	0.5595	0.2129	**0.7695**
	Macro	0.5644	0.4493	0.5423	0.5366	0.6447	0.6017	0.5913	0.5958	0.5223	0.1915	**0.6587**
	Micro	0.5637	0.2960	0.5813	0.5421	0.5196	0.4984	0.4901	0.4876	0.5828	0.2262	**0.8387**
Llama-3-8B-262k	Overall	0.7183	0.2703	0.5279	0.5825	0.6231	0.7150	0.7166	0.7147	0.6603	0.5038	**0.8285**
	Macro	0.6218	0.4198	0.5751	0.4652	0.5115	0.6189	0.6221	0.6183	0.5292	0.3529	**0.6928**
	Micro	0.7786	0.1770	0.4985	0.6559	0.6928	0.7752	0.7757	0.7750	0.7422	0.5982	**0.9401**
Vicuna-7B	Overall	0.1273	0.1794	0.1147	0.1041	0.1066	0.1276	0.1277	0.1277	0.0862	0.0345	**0.3704**
	Macro	0.1267	0.1575	0.1164	0.0877	0.0995	0.1266	0.1259	0.1263	0.0567	0.0409	**0.2678**
	Micro	0.1276	0.2016	0.1137	0.1144	0.1111	0.1282	0.1289	0.1285	0.1046	0.0306	**0.4346**

5 Evaluation

We conduct experiments with GraphInsight on two level graph understanding tasks: (i) macro understanding, (ii) micro understanding. Section 5.1 summarizes the experimental setup. Section 5.2 demonstrates the advantages of GraphInsight in enhancing LLMs' understanding of graphs. Ablation study and hyperparameter analysis are on Sections 5.3 and 5.4. Each reported value is the average of three runs.

5.1 Experimental Setup

5.1.1 Baselines.

We compare GraphInsight with 10 baselines: (a) raw sequence input method without any processing (Raw Seq.), (b) GNN encoding-based method[3] (GraphToken) (Perozzi et al., 2024), (c) prompting methods including Build-a-Graph Prompting (BAG) (Wang et al., 2024a), Chain-of-Thought (COT) (Wei et al., 2022), and Few-Shot (FS) (Brown et al., 2020), (d) reordering methods (Ge et al., 2024) based on breadth-first-search order (BFS), depth-first-search order (DFS), and shortest-path order (SP), and (e) structural methods including adjacency lists (AL) and adjacency matrices (AM).

5.1.2 Benchmarks and Evaluation Tasks.

Existing graph understanding benchmarks suffer from limited node coverage, unclear task definitions, a lack of structural diversity, and insufficient support for multi-step graph reasoning. To address these issues, we introduce GraphSQA, a comprehensive benchmark designed to evaluate the ca-

(a) Macro-level (Mistral-7B) (b) Macro-level (Vicuna-7B)

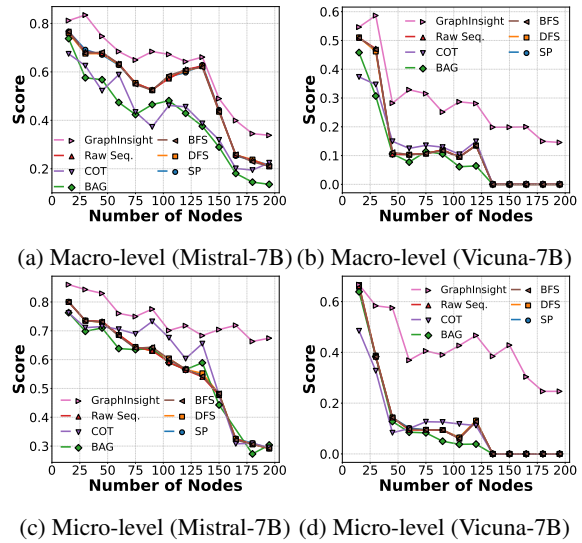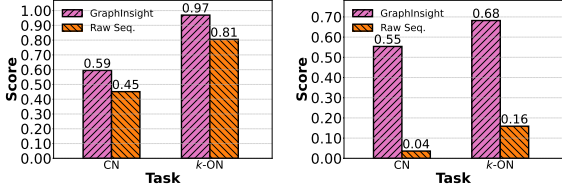(c) Micro-level (Mistral-7B) (d) Micro-level (Vicuna-7B)

Figure 4: Analysis on Graphs with Different $|V|$

pabilities of LLMs in understanding graph structures, which encompasses a wide range of task types found in existing benchmarks (Wang et al., 2024a; Yuan et al., 2024). GraphSQA includes a broad spectrum of node counts and features diverse graph structures, such as multi-edges and self-loops. It encompasses two types of graph understanding tasks: (1) 5 macro-level graph tasks and (2) 15 micro graph tasks including 7 composite tasks. Further details are provided in **Appendix A**. In our experiments, we evaluate the average performance of various methods and models on diverse graph tasks with $|V|$ ranging from 15 to 200 nodes, covering 10,400 tasks in total—2,600 macro-level and 7,800 micro-level.

5.1.3 Metric.

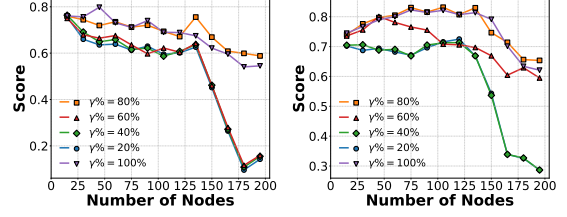GraphSQA employs three answer types: boolean, numerical, and set. For boolean answers, the score is 1 if correct, otherwise 0. For numerical answers, we use one minus the relative error (Wang et al.,

---

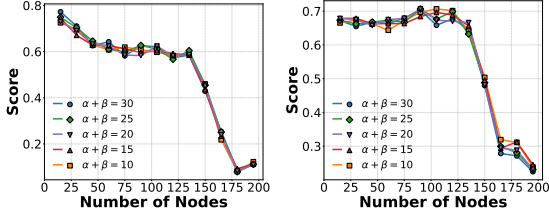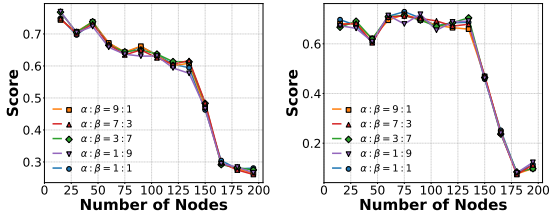[3]GNN encoding-based methods like GraphToken rely on implicit graph embeddings, which are orthogonal to GraphInsight's explicit graph description approach. Despite this, we include them to ensure a comprehensive comparison.

(a) Qwen2-7B (b) Vicuna-7B

Figure 5: Analysis on Composite Tasks

(a) Mistral-7B (b) Qwen2-7B

Figure 7: GraphRAG Base Rate $\gamma\%$.

Table 2: Ablation Study.

Methods	Models	Macro	Micro
Raw Seq.		0.5222	0.5635
w/ Reorganization	Mistral-7B	**0.5851**	0.5639
w/ Reorganization and GraphRAG		-	**0.7425**
Raw Seq.		0.4379	0.4597
w/ Reorganization	Llama-3-8B	**0.5029**	0.5298
w/ Reorganization and GraphRAG		-	**0.7605**
Raw Seq.		0.5644	0.5637
w/ Reorganization	Qwen2-7B	**0.6293**	0.5676
w/ Reorganization and GraphRAG		-	**0.8387**
Raw Seq.		0.6218	0.7786
w/ Reorganization	Llama-3-8B-262K	**0.6928**	0.8477
w/ Reorganization and GraphRAG		-	**0.9401**
Raw Seq.		0.1267	0.1276
w/ Reorganization	Vicuna-7B	**0.1571**	0.1588
w/ Reorganization and GraphRAG		-	**0.4346**

(a) Mistral-7B ($\frac{\alpha}{\beta} = \frac{3}{7}$) (b) Qwen2-7B ($\frac{\alpha}{\beta} = \frac{3}{7}$)

(c) Mistral-7B ($\alpha + \beta = 15$) (d) Qwen2-7B ($\alpha + \beta = 15$)

Figure 6: Head and Tail Strong Region Thresholds α and β.

2024a). For set answers, the score is the Jaccard similarity (Ji et al., 2013) between the answer and the ground truth.

5.1.4 Models.

We employed a diverse selection of open-source LLMs, encompassing both long-sequence models, such as Mistral-7B and Llama3-8B-256K (fine-tuned for long sequences), and their counterparts, including Llama3-8B, Qwen2-7B, and Vicuna-7B.

Given that different LLMs possess distinct capabilities and exhibit varied positional biases due to their distinct architectures, which influence their sensitivity to information within strong memory regions (e.g., head and tail) versus weak memory regions (e.g., middle portions) of input sequences. Therefore, we adjusted key hyperparameters for each LLM, specifically the strong region thresholds $\alpha\%$ and $\beta\%$, as well as the GraphRAG base rate $\gamma\%$ (Section 5.4), to optimize their individual performance.

5.2 Performance

5.2.1 Macro-level Tasks.

GraphInsight outperforms all other methods across all LLMs for macro-tasks, as shown in Table 1. Specifically, GraphInsight can achieve up to a $4.61\times$ increase in score compared to AM on Vicuna-7B. Structural methods show the smallest improvement for macro-level tasks. The other two types of methods, i.e., prompting and structural methods, result in only minimal improvements in macro-level tasks. For example, on Llama-3-8B, they can achieve at most a minimal improvement of 5.7%, from 0.4379 to 0.4404. However, GraphInsight can provide a substantial increase of 23.82%, from 0.4379 to 0.5422. Also, as shown in Figure 4(a)-(b), across macro-level tasks with varying $|V|$, GraphInsight consistently outperforms other methods. As $|V|$ increases, the understanding capability tends to decrease, but GraphInsight shows the smallest decline.

5.2.2 Micro-level Tasks.

For the micro-level tasks, GraphInsight still outperforms the others, as shown in Table 1. Notably, GraphInsight can achieve up to an $14.02\times$ increase in score compared to AL on Vicuna-7B. Similar to macro-level tasks, baseline methods offer minimal improvement in the large model's understand-

ing. For example, on Qwen2-7B, they can achieve at most a minimal improvement of 3.4%, from 0.5637 to 0.5828. However, GraphInsight can provide a substantial increase of 48.78%, from 0.5637 to 0.8387. In Figures 4(c)-(d), as $|V|$ increases, GraphInsight causes the least decline in the model's understanding. For example, on Mistral-7B, when $|V|$ increases from 135 to 195, the score decreases only slightly from 0.68 to 0.67. In contrast, the best baseline's score drops more significantly, from 0.54 to 0.29. We also conduct experiments on composite tasks, e.g., common neighbor finding (CN) and k-order neighbor finding (k-ON) shown in Figure 5. GraphInsight can provide an improvement of $12.75\times$ and $3.25\times$ on Vicuna, respectively.

Detailed task-level results and method performance rankings are provided in the Appendix (see Figures 8 to 12 and Table 6). We observe that different ordering methods (importance-based[4], BFS, DFS, and SP) show task-specific preferences, suggesting that adaptive method selection based on task characteristics could be a promising future research direction.

5.3 Ablation Study

We conduct the ablation study on graph description organization and GraphRAG to verify the effectiveness of them. As shown in Table 2, graph description organization is the optimization primarily for macro-level tasks as well as for micro-level tasks. For example, on Llama-3-8B-262K, reorganization can achieve an 11.4% increase for macro-level tasks and an 8.9% increase for micro-level tasks. On the other hand, GraphRAG focuses on micro-level tasks and can provide further notably improvements for micro-level tasks on top of the gains achieved by reorganization.

5.4 Hyperparameter Analysis

5.4.1 Strong Region Thresholds $\alpha\%$ and $\beta\%$.

We conduct experiments with $\alpha\% + \beta\% = 15\%$ (resp. $\frac{\alpha\%}{\beta\%} = \frac{3}{7}$) to assess how variations in $\frac{\alpha\%}{\beta\%}$ (resp. $\alpha\% + \beta\%$) affect LLMs' graph understanding. As shown in Figure 6, the performance remains stable in both cases, indicating insensitivity to these parameters. If model-specific adjustments are not made, we can set $\frac{\alpha\%}{\beta\%} = \frac{3}{7}$ and $\alpha\% + \beta\% = 15\%$ for simplicity.

---

[4]Different importance-based metrics (e.g., degree, specific centrality measures) may also have task-specific impacts.

5.4.2 GraphRAG Base Rate $\gamma\%$.

We examine the impact of $\gamma\%$ on the size of the GraphRAG base. As $\gamma\%$ increases, the base retains more critical graph information, improving micro-level understanding. However, as shown in Figure 6, the improvement diminishes when $\gamma\%$ exceeds 80%. We recommend setting $\gamma\%$ to 80%.

6 Conclusion

In this paper, we introduce GraphInsight, the first framework to address the challenge of graph structure comprehension in LLMs through their inherent "positional bias." By leveraging strong memory in certain graph sequence regions and compensating for weaker ones, GraphInsight enhances graph understanding. It employs two key techniques: importance-based reorganization and lightweight RAG, optimized for macro- and micro-level tasks. Experiments show that GraphInsight outperforms all baselines across graph sizes. Future work will focus on improving LLMs' understanding of more complex graph types, including those with labels and semantics.

Limitations

The proposed **GraphInsight** framework significantly enhances LLMs' ability to comprehend graph structures, yet it presents several limitations that could inspire future research directions.

First, the quantification of subgraph importance remains a task-dependent challenge, as different graph characteristics and task objectives require distinct importance measures. Future work could explore more task-aware strategies for subgraph importance quantification to improve the framework's adaptability across various graph-related tasks. Additionally, while the GraphRAG component addresses memory limitations effectively, its optimization in terms of retrieval efficiency and storage overhead remains an open area for improvement. Future research could focus on refining these aspects to enhance retrieval speed and reduce memory consumption. Moreover, the framework's approach contrasts with other orthogonal LLM-based methods, such as instruction fine-tuning or implicit embedding techniques. Integrating these methods with GraphInsight could complement its capabilities and foster further advancements in graph structure comprehension tasks.

Acknowledgements

References

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, and Jian-Guang Lou. 2024. Make your llm fully utilize the context. *arXiv preprint arXiv:2404.16811*.

Bojan Babic. 2023. Llms for social networks: Applications, challenges and solutions. In *CIKM Workshop*, volume 3532.

Jinheon Baek et al. 2023. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. In *ACL Workshop*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI*, pages 17682–17690.

Maciej Besta, Robert Gerstenberger, Emanuel Peter, Marc Fischer, Michał Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefler. 2023. Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. *ACM Computing Surveys*, 56(2).

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *NeurIPS*, 33:1877–1901.

Yukun Cao, Zengyi Gao, Zhiyang Li, Xike Xie, Kevin Zhou, and Jianliang Xu. 2024. Lego-graphrag: Modularizing graph-based retrieval-augmented generation for design space exploration. *arXiv preprint arXiv:2411.05844*.

Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. 2024a. Graphwiz: An instruction-following language model for graph computational problems. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 353–364.

Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, et al. 2024b. Exploring the potential of large language models (llms) in learning on graphs. *KDD*, 25(2):42–61.

Sudeep Das, Raghav Saboo, Chaitanya S. K. Vadrevu, Bruce Wang, and Steven Xu. 2024. Applications of llms in e-commerce search and product knowledge graph: The doordash case study. In *WSDM*. ACM.

Jingcheng Deng, Liang Pang, Huawei Shen, and Xueqi Cheng. 2023. Regavae: A retrieval-augmented gaussian mixture variational auto-encoder for language modeling. *Findings of EMNLP*.

Jingcheng Deng, Zihao Wei, Liang Pang, Hanxing Ding, Huawei Shen, and Xueqi Cheng. 2024. Everything is editable: Extend knowledge editing to unstructured data in large language models. *arXiv preprint arXiv:2405.15349*.

Fatemi et al. 2023. Talk like a graph: Encoding graphs for large language models. *arXiv preprint arXiv:2310.04560*.

Zengyi Gao, Yukun Cao, Hairu Wang, Ao Ke, Yuan Feng, Xike Xie, and S Kevin Zhou. 2025. FRAG: A flexible modular framework for retrieval-augmented generation based on knowledge graphs. *Findings of ACL*.

Yingqiang Ge, Wenyue Hua, Kai Mei, Jianchao Ji, Juntao Tan, Shuyuan Xu, Zelong Li, and Yongfeng Zhang. 2023. Openagi: When LLM meets domain experts. In *NeurIPS*.

Yuyao Ge, Shenghua Liu, Lingrui Mei, Lizhe Chen, and Xueqi Cheng. 2024. Sequential ordering in textual descriptions: Impact on spatial perception abilities of large language models. *arXiv preprint arXiv:2402.07140*.

Hao Geng, Deqing Wang, Fuzhen Zhuang, Xuehua Ming, Chenguang Du, Ting Jiang, Haolong Guo, and Rui Liu. 2022. Modeling dynamic heterogeneous graph and node importance for future citation prediction. In *CIKM*, pages 572–581. ACM.

Akash Ghosh, Arkadeep Acharya, Raghav Jain, Sriparna Saha, Aman Chadha, and Setu Sinha. 2024. Clipsyntel: CLIP and LLM synergy for multimodal question summarization in healthcare. In *AAAI*, pages 22031–22039.

Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. 2023. Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking. *arXiv preprint arXiv:2305.15066*.

Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. 2023. Tabllm: Few-shot classification of tabular data with large language models. In *AISTATS*, pages 5549–5581. PMLR.

Cheng-Yu Hsieh, Yung-Sung Chuang, Chun-Liang Li, Zifeng Wang, Long T Le, Abhishek Kumar, James Glass, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, et al. 2024. Found in the middle: Calibrating positional attention bias improves long context utilization. *arXiv preprint arXiv:2406.16008*.

Jianqiu Ji, Jianmin Li, Shuicheng Yan, Qi Tian, and Bo Zhang. 2013. Min-max hash for jaccard similarity. In *ICDM*, pages 301–309. IEEE.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, et al. 2023. Time-llm: Time series forecasting by reprogramming large language models. *arXiv preprint arXiv:2310.01728*.

James R Kirk, Robert E Wray, Peter Lindes, and John E Laird. 2024. Improving knowledge extraction from llms for task learning through agent analysis. In *AAAI*, volume 38, pages 18390–18398.

Hanbum Ko, Hongjun Yang, Sehui Han, Sungwoong Kim, Sungbin Lim, and Rodrigo Hormazabal. 2024. Filling in the gaps: Llm-based structured data generation from semi-structured scientific data. In *ICML Workshop*.

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. *NeurIPS*, 33:9459–9474.

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. 2022. A survey on deep learning for named entity recognition. *IEEE Trans. Knowl. Data Eng.*, 34(1):50–70.

Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. 2023. A survey of graph meets large language model: Progress and future directions. *arXiv preprint arXiv:2311.12399*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *TACL*, 12:157–173.

Shihu Liu and Haiyan Gao. 2023. The structure entropy-based node importance ranking method for graph data. *Entropy*, 25(6):941.

Zihan Luo, Xiran Song, Hong Huang, Jianxun Lian, Chenhao Zhang, Jinqi Jiang, Xing Xie, and Hai Jin. 2024. Graphinstruct: Empowering large language models with graph understanding and reasoning capability. *arXiv preprint arXiv:2403.04483*.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Nick Barnes, and Ajmal Mian. 2023. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *TKDE*, 36(7):3580–3599.

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*.

Joel Rorseth, Parke Godfrey, Lukasz Golab, Divesh Srivastava, and Jaroslaw Szlichta. 2024. RAGE against the machine: Retrieval-augmented LLM explanations. In *ICDE*, pages 5469–5472. IEEE.

Payel Santra. 2024. Leveraging llms for detecting and modeling the propagation of misinformation in social networks. In *SIGIR*, page 3073. ACM.

Wenbo Shang and Xin Huang. 2024. A survey of large language models on generative graph analytics: Query, learning, and applications. *arXiv preprint arXiv:2404.14809*.

Siqi Shen, Lajanugen Logeswaran, Moontae Lee, Honglak Lee, Soujanya Poria, and Rada Mihalcea. 2024. Understanding the capabilities and limitations of large language models for cultural commonsense. pages 5668–5680.

Daivik Sojitra, Raghav Jain, Sriparna Saha, Adam Jatowt, and Manish Gupta. 2024. Timeline summarization in the era of llms. In *SIGIR*, pages 2657–2661. ACM.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *WSDM*, pages 645–654.

Jianheng Tang, Qifan Zhang, Yuhan Li, and Jia Li. 2024. Grapharena: Benchmarking large language models on graph computational problems. *arXiv preprint arXiv:2407.00379*.

Raphael Tang, Xinyu Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Ture. 2023. Found in the middle: Permutation self-consistency improves listwise ranking in large language models. *arXiv preprint arXiv:2310.07712*.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2024a. Can language models solve graph problems in natural language? *NeurIPS*, 36.

Xiang Wang, Tinglin Huang, Dingxian Wang, Yancheng Yuan, Zhenguang Liu, Xiangnan He, and Tat-Seng Chua. 2021. Learning intents behind interactions with knowledge graph for recommendation. In *WWW*, pages 878–887.

Xindi Wang, Mahsa Salmani, Parsa Omidi, Xiangyu Ren, Mehdi Rezagholizadeh, and Armaghan Eshaghi. 2024b. Beyond the limits: A survey of techniques to extend the context length in large language models. *arXiv preprint arXiv:2402.02244*.

Yan Wang, Zhixuan Chu, Xin Ouyang, Simeng Wang, Hongyan Hao, Yue Shen, Jinjie Gu, Siqiao Xue, James Zhang, Qing Cui, Longfei Li, Jun Zhou, and Sheng Li. 2024c. LLMRG: improving recommendations through large language model reasoning graphs. In *AAAI*, pages 19189–19196.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 35:24824–24837.

Wu et al. 2024a. Never miss a beat: An efficient recipe for context window extension of large language models with consistent" middle" enhancement. *arXiv preprint arXiv:2406.07138*.

Qiming Wu, Zichen Chen, Will Corcoran, Misha Sra, and Ambuj K Singh. 2024b. Grapheval2000: Benchmarking and improving large language models on graph datasets. *arXiv preprint arXiv:2406.16176*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks.

Mengjia Xu. 2021. Understanding graph embedding methods and their applications. *SIAM Review*, 63(4):825–853.

Xinli Yu, Zheng Chen, Yuan Ling, Shujing Dong, Zongyi Liu, and Yanbin Lu. 2023. Temporal data meets llm–explainable financial time series forecasting. *arXiv preprint arXiv:2306.11025*.

Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. 2024. Gracore: Benchmarking graph comprehension and complex reasoning in large language models. *arXiv preprint arXiv:2407.02936*.

Zhang et al. 2024a. Attention instruction: Amplifying attention in the middle via prompting. *arXiv preprint arXiv:2406.17095*.

Zeyang Zhang, Xin Wang, Ziwei Zhang, Haoyang Li, Yijian Qin, Simin Wu, and Wenwu Zhu. 2024b. Llm4dyg: Can large language models solve problems on dynamic graphs? *KDD*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *AAAI*, volume 38, pages 19632–19642.

Guangming Zhu, Siyuan Wang, Tianci Wu, and Liang Zhang. 2024. Enhance sketch recognition's explainability via semantic component-level parsing. In *AAAI*, pages 7731–7738. AAAI Press.

A Benchmark

In order to systematically evaluate the ability of large language models (LLMs) to understand and process graph structures, we introduce a comprehensive benchmark GraphSQA, which not only broadens the scope of evaluation by including a wider variety of graph structures but also provides a more rigorous and detailed assessment of both macro- and micro-level graph understanding tasks.

A.1 Introduction to GraphSQA

Existing graph understanding benchmarks suffer from several critical limitations, including restricted node coverage, ambiguous task definitions, limited structural diversity, and inadequate support for multi-step graph reasoning. These deficiencies significantly impede a comprehensive evaluation of large language models (LLMs) in their ability to understand and process graph structures effectively. To address these challenges, we introduce **GraphSQA**, a benchmark meticulously designed to evaluate LLMs across multiple dimensions of graph understanding.

GraphSQA offers a standardized evaluation suite, enabling consistent and comprehensive assessment of LLMs' capabilities in understanding and reasoning about graph structures. Unlike existing benchmarks that primarily focus on small graphs containing up to 35 nodes, GraphSQA broadens the scope by encompassing a diverse spectrum of node quantities, ranging from 15 to 300 nodes, and includes various graph structures, such as multi-edges and self-loops. The benchmark is organized into two primary categories of graph understanding tasks: **macro-level tasks** and **micro-level tasks**, with the latter further subdivided into 15 tasks, including 7 composite tasks. This comprehensive structure provides a more challenging and extensive evaluation environment, better reflecting the complexities encountered in real-world graph-based applications. A comparative overview of existing benchmarks designed to evaluate the graph understanding capabilities of large language models (LLMs) is presented in Table 3.

A.2 Key Features of GraphSQA

GraphSQA is meticulously designed to provide a standardized evaluation suite for the consistent and comprehensive assessment of LLMs' capabilities in understanding and reasoning about graph structures. Unlike existing benchmarks that primarily fo-

cus on small graphs containing 10-35 nodes, GraphSQA broadens the scope by including a wider range of node sizes, extending up to 200 nodes. This expansion provides a more challenging and extensive evaluation environment, better reflecting the complexities encountered in real-world graph-based applications.

A.2.1 Macro-level Tasks

Macro-level tasks are designed to assess LLMs' capabilities in understanding the fundamental properties of entire graphs. These tasks involve evaluating the overall graph structure, such as determining the node count, assessing graph connectivity, detecting cycles, identifying the maximum edge weight, and pinpointing nodes with the highest degrees. Each task is carefully crafted to test different aspects of global graph understanding, presenting unique challenges to the models. In total, there are 5 tasks in this category, each contributing to a comprehensive evaluation of a model's ability to grasp the graph's macrostructure.

A.2.2 Micro-level Tasks

Micro-level tasks focus on understanding and reasoning about specific nodes or edges within a graph. These tasks include recognizing direct connections between nodes, calculating node degrees, identifying leaf nodes, checking for even degrees, and determining specific edge weights. This category comprises 15 tasks, including 8 composite tasks, each targeting a particular aspect of local graph structure. These tasks are designed to test the models' ability to navigate and reason about graph components on a more granular scale, thus assessing their proficiency in detailed graph analysis.

A.3 Overview of GraphSQA Categories

Macro-level Tasks: The macro-level tasks are aimed at evaluating a model's ability to comprehend the entire graph's structure. These tasks require the model to analyze the graph as a whole, focusing on its global properties rather than individual components.

- **Node Count Identification:** *How many nodes are in this graph?* This task evaluates the model's ability to accurately determine the total number of nodes present within a given graph. Understanding the node count is fundamental for analyzing the graph's overall size and structure, and serves as a crucial precursor to further global analysis.

	Task Types	Max Graph Size	Graph Structural Diversity	Multi-step Graph Reasoning
GraphQA (Fatemi et al., 2023)	6	20	×	×
GraphInstruct (Luo et al., 2024)	21	35	×	×
GraphInstruct(Graphwiz) (Chen et al., 2024a)	9	100	×	×
LLM4DyG (Zhang et al., 2024b)	9	20	×	×
GPT4Graph (Guo et al., 2023)	10	20	×	×
GraphSQA	20	200	✓	✓

Table 3: Comparison of Benchmarks for Evaluating Graph Understanding in Large Language Models (LLMs)

- **Graph Connectivity Determination:** *Is this graph a connected graph?* This task assesses whether the model can determine if the graph is fully connected, meaning there is a path between every pair of nodes. Understanding graph connectivity is essential for evaluating the graph's structural integrity and the inter-relationship between its components, which is critical for various applications in network analysis.

- **Cycle Detection:** *Does this graph contain a cycle?* This task examines the model's ability to detect the presence of cycles within the graph, a fundamental aspect of graph theory. Cycle detection is crucial for understanding complex graph behaviors and structures, such as feedback loops and circular dependencies.

- **Maximum Edge Weight Identification:** *What is the maximum weight of the edges in this graph?* This task measures the model's capability to identify the heaviest edge within the graph. In weighted graphs, where edge weights represent varying strengths of connections, identifying the maximum weight is important for understanding the distribution of relationships and the overall dynamics of the graph.

- **Top-Degree Nodes Identification:** *What are the nodes with the top N highest degrees in this graph?* This task evaluates the model's proficiency in ranking nodes based on their degrees, i.e., the number of edges connected to each node. Identifying top-degree nodes is important for understanding which nodes are most influential or central within the graph, playing key roles in its connectivity and structure.

Micro-level Tasks: Micro-level tasks require the model to analyze and understand specific elements or substructures within the graph, such as individual nodes or edges. These tasks demand a more detailed and focused approach to graph analysis.

- **Direct Connection Identification:** *Is there a direct connection between node U and node V?* This task tests the model's ability to recognize direct adjacency between two specific nodes within the graph. Understanding direct connections is fundamental to analyzing the local structure of the graph and how nodes are interrelated.

- **Node Degree Calculation:** *What is the degree of node N?* This task measures the model's ability to determine the degree of a specific node, which refers to the number of edges connected to it. Node degree is a key indicator of a node's importance and centrality within the graph.

- **Leaf Node Identification:** *Is node N a leaf node?* This task evaluates whether the model can correctly identify leaf nodes, which are nodes with only one connection. Leaf nodes are particularly significant in tree-like structures, where they represent terminal nodes or endpoints.

- **Even Degree Check:** *Does node N have an even degree?* This task assesses the model's ability to determine whether a given node has an even degree. Understanding degree parity is important for analyzing the graph's structure and potential for certain types of substructures.

- **Neighbor Identification:** *Who are the neighbors of node N?* This task tests the model's ability to accurately identify and enumerate the neighbors of a given node. Understanding the local neighborhood is crucial for analyzing the node's immediate environment and its influence within the graph.

- **Common Neighbors Identification:** *Do nodes U and V have any common neighbors?*

This task evaluates the model's capability to identify shared neighbors between two specific nodes. Understanding common neighbors is important for exploring the overlap in local neighborhoods, which has implications for clustering and community detection.

- **Degree Comparison:** *Is the degree of node U greater than the degree of node V?* This task assesses the model's comparative reasoning skills by requiring it to compare the degrees of two nodes. Understanding the relative degrees of nodes is essential for analyzing the distribution of connections and identifying influential nodes.

- **Edge Weight Determination:** *What is the weight of the edge between node U and node V?* This task tests the model's ability to determine the specific weight of an edge between two nodes in a weighted graph. Accurately identifying edge weights is crucial for understanding the graph's structure, especially in contexts where edge weights influence the graph's behavior.

- **Connected Edges Identification:** *Given the edge (u, v), find all edges connected to it. List the answers in the format of '[(1, 2), (3, 4), ...]'.* This task evaluates the model's ability to identify all edges that are connected to a given edge. Understanding local connectivity around a specified edge is essential for analyzing the graph's structure and how different parts of the graph are interlinked.

- **Subgraph Completeness Check:** *Given the nodes [n1, n2, ..., nk], determine if they form a complete subgraph. List the answer directly in the format of 'Yes' or 'No'.* This task assesses the model's ability to determine whether a specified set of nodes forms a complete subgraph, where every pair of nodes is connected by an edge. Understanding subgraph completeness is important for identifying densely connected clusters within the graph.

- **Highest-Degree Neighbor Identification:** *Given the node n, find the neighbor's neighbor with the highest degree. List the answer directly as the node id.* This task evaluates the model's ability to identify the neighbor of a given node's neighbor that has the highest degree. This task requires the model to traverse the graph's local neighborhood and perform degree comparisons, which is crucial for understanding local influence and connectivity.

- **Third-Order Neighbors Identification:** *Given the node n, find all its third-order neighbors. List the answers in the format of '[1, 2, ...]'.* This task tests the model's ability to identify third-order neighbors, which are nodes three steps away from the given node. Understanding extended neighborhoods is important for analyzing the broader context in which a node operates.

- **Direct Neighbor Connection Identification:** *Given the node n and a specified node m, find n's neighbors that are directly connected to m. List the answers in the format of '[1, 2, ...]'.* This task assesses the model's ability to identify which of a given node's neighbors are directly connected to another specified node. This task involves understanding both direct and indirect connections within the graph, which is important for analyzing the graph's local structure.

- **Triangles Identification:** *Given the node n, find all triangles (sets of three nodes that are mutually connected) it forms with its neighbors. List the answers in the format of '[(1, 2, 3), (4, 5, 6), ...]'.* This task evaluates the model's ability to identify all triangles that involve a given node and its neighbors. Understanding triangles is fundamental for analyzing local clustering, community structures, and the graph's overall connectivity.

- **Common Third-Order Neighbor Identification:** *Given nodes n and m, find all common third-order neighbors. List the answers in the format of '[1, 2, ...]'.* This task evaluates the model's ability to identify common third-order neighbors between two nodes. This involves analyzing the extended neighborhoods of both nodes and determining the overlap, which is crucial for understanding the broader structure and connectivity within the graph.

B   Experimental Settings

In this section, we detail the computing infrastructure used for conducting our experiments, including both hardware and software configurations.

Hardware: The experiments were conducted on Ubuntu 20.04.2 using four NVIDIA A100 GPUs, each with 80 GB of memory (PCIe interface).

Software: The software environment was configured with Python 3.11.9. The experiments were primarily implemented using PyTorch version 2.3.1. Additionally, we employed the vLLM library (version 0.5.3) for managing large-scale language model inference. The system ran on CUDA version 12.2, optimizing GPU performance for the computations.

C Hyperparameter Details

In this section, we list the final hyperparameters used for each model/algorithm in the experiments conducted in this paper, as in Table 4.

Table 4: Final hyperparameters suggested in the experiments.

Model/Algorithm	Hyperparameter	Value
GraphInsight	GraphRAG Base Rate $\gamma\%$	0.80
	$\alpha\%$ (Head Memory Region)	4.5%
	$\beta\%$ (Tail Memory Region)	10.5%
PageRank	Damping Factor λ	0.85
	Max Iterations	100

D Notations

This section summarizes all notations used throughout this paper, as in Table 5.

Table 5: Notations used throughout this paper.

Notations	Definitions or Descriptions
$G = (V, E)$	Graph with node set V and edge set E
V	Set of nodes in the graph
E	Set of edges in the graph
v_i, v_j	Nodes in the graph
e_{ij}	Edge between node v_i and node v_j
w_{ij}	Weight of edge e_{ij}
$\mathcal{T}$	Standard graph description sequence
$\mathcal{T}_s$	Subgraph descriptions sequence
t_i	Subgraph description
$\bar{\mathcal{T}}$	Reorganized graph description sequence
C_{LLM}	Capacity of an LLM for graph understanding
$C(\mathcal{T}, p)$	Comprehension ability at a specific position p within $\mathcal{T}$
$\Psi(p)$	Positional bias curve representing LLM's inherent comprehension at position p
$\Phi(p)$	Importance of position p in sequence $\mathcal{T}$
$\mathcal{I}(t_i)$	Importance of subgraph description t_i
D_{KL}	Kullback-Leibler divergence
G_i	Subgraph centered on node v_{c_i}
v_{c_i}	Central node of subgraph G_i
$\text{PR}(v)$	PageRank score of node v
$\text{InNb}(v)$	Set of nodes with edges directed towards node v
$\text{OutDeg}(u)$	Number of edges leaving node u
$\alpha\%, \beta\%$	Proportions defining the head and tail (strong memory regions) of the sequence

E Experimental Results for different task types

The experimental results for different task types on GraphSQA are illustrated in Figures 8 to 12. These figures analyze macro- and micro-level similarity metrics across various LLM configurations,

including Llama-3-8B-Instruct-262k, Meta-Llama-3-8B-Instruct, Mistral-7B-Instruct-v0.2, Qwen2-7B-Instruct, and Vicuna-7b-v1.1.

F Metric Analysis

In the GraphSQA evaluation framework, responses are classified into three distinct categories: boolean, numerical, and set-based. Each category necessitates a specific metric to accurately evaluate the correctness of the predictions.

For **boolean answers**, which are binary in nature (e.g., true/false), the evaluation is straightforward. A score of 1 is assigned if the predicted answer exactly matches the ground truth; otherwise, the score is 0. This binary scoring system provides an unambiguous measure of correctness, ensuring clarity in the assessment of such responses.

Numerical answers, on the other hand, require a more nuanced scoring approach due to the potential variability in magnitude. To address this, we employ a metric based on relative error, a widely recognized method in numerical evaluations. The score is calculated as one minus the relative error between the predicted value $\tilde{y}$ and the ground truth value y (Wang et al., 2024a), formally defined as:

$$\text{Score} = 1 - \frac{|\tilde{y} - y|}{\max(\tilde{y}, y)} \qquad (10)$$

This metric effectively penalizes larger deviations from the ground truth while allowing for partial credit when the prediction is reasonably close. As shown in Equation 10, it offers a graded evaluation that more accurately reflects the precision of the numerical predictions.

For **set-type answers**, which involve comparisons between predicted and ground truth sets (e.g., sets of nodes, edges, or other graph elements), we utilize the Jaccard similarity coefficient as the evaluation metric (Ji et al., 2013). The Jaccard similarity measures the degree of overlap between the predicted set A and the ground truth set B, defined as:

$$\text{Jaccard Similarity} = \frac{|A \cap B|}{|A \cup B|} \qquad (11)$$

As expressed in Equation 11, this similarity measure yields a score ranging from 0 to 1, where 1 indicates a perfect match between the predicted and ground truth sets, and 0 indicates no overlap. By considering both false positives and false negatives, the Jaccard similarity provides a balanced

Table 6: Analysis on Macro- and Micro-level Tasks with Different Baseline Methods and Models.

Models	Tasks	Raw Seq.	Prompting Methods			Reordering Methods			Structural Methods		GraphInsight
			COT	FS	BAG	BFS	DFS	SP	AL	AM	
Llama-3-8B-Instruct-262k	Overall	3	9	7	4	5	2	5	8	10	**1**
	Macro	3	7	4	8	5	2	6	9	10	**1**
	Micro	3	10	9	2	6	4	5	7	8	**1**
Meta-Llama-3-8B-Instruct	Overall	6	9	5	8	3	2	4	7	10	**1**
	Macro	5	8	6	9	2	3	1	7	10	4
	Micro	9	8	5	7	3	2	4	6	10	**1**
Mistral-7B-Instruct-v0.2	Overall	7	5	9	2	8	6	2	4	10	**1**
	Macro	5	9	2	6	7	4	1	3	10	**1**
	Micro	8	3	9	2	6	5	4	7	10	**1**
Qwen2-7B-Instruct	Overall	5	4	9	7	3	8	6	2	10	**1**
	Macro	7	8	9	6	1	5	3	4	10	**1**
	Micro	3	2	9	7	6	8	4	5	10	**1**
vicuna-7b-v1.1	Overall	5	5	8	7	4	3	2	9	10	**1**
	Macro	2	3	8	7	5	4	6	9	10	**1**
	Micro	8	6	9	5	4	3	2	7	10	**1**
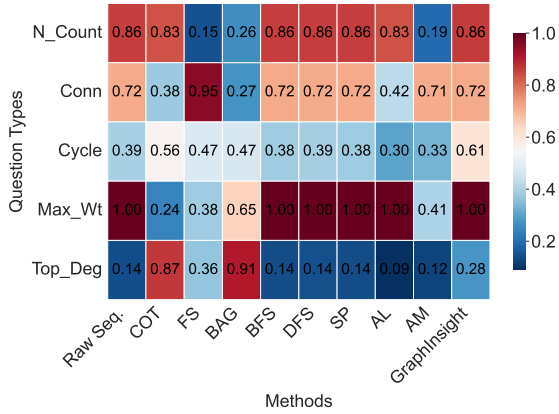
(a) Macro-level (Llama-3-8B-Instruct-262k)
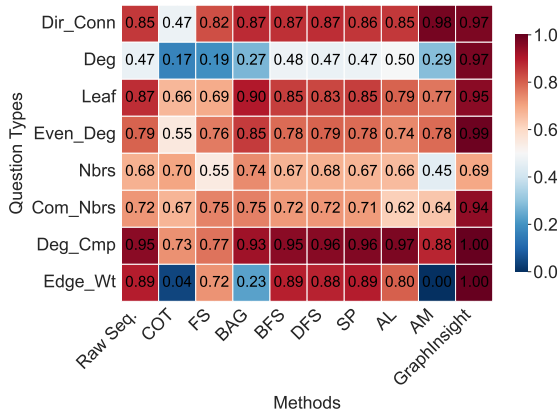
(b) Micro-level (Llama-3-8B-Instruct-262k)

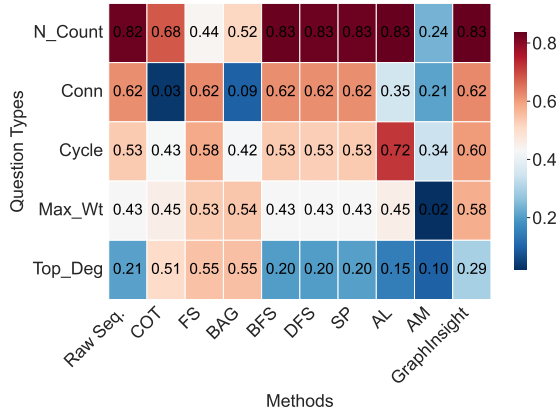Figure 8: Macro- and Micro-level Similarity Analysis for Llama-3-8B-Instruct-262k
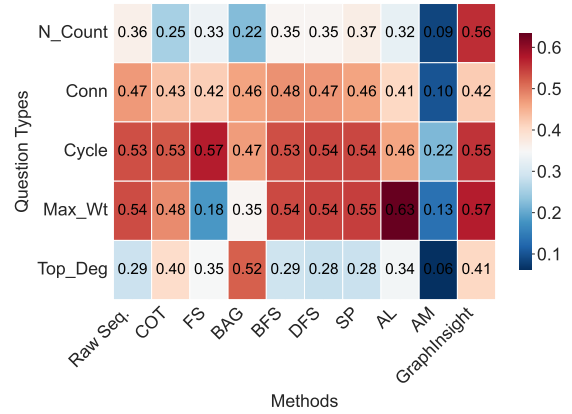
(a) Macro-level (Meta-Llama-3-8B-Instruct)
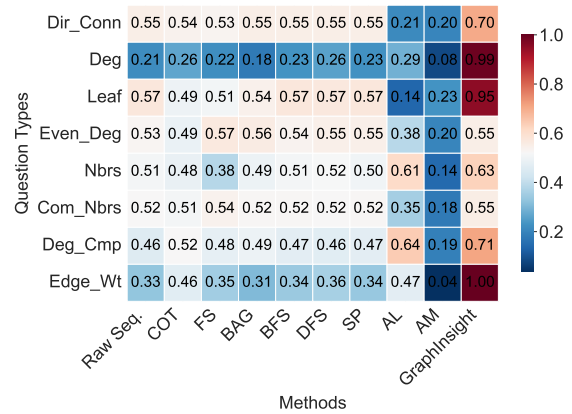
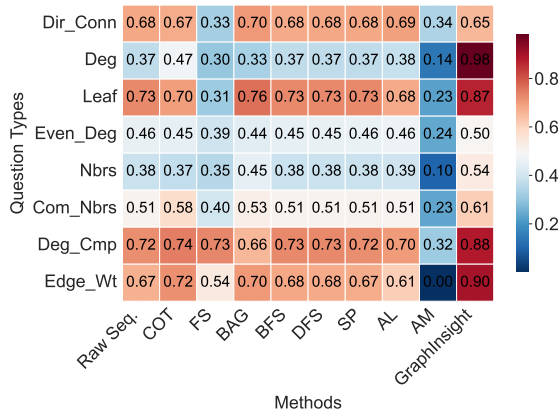

(b) Micro-level (Meta-Llama-3-8B-Instruct)

Figure 9: Macro- and Micro-level Similarity Analysis for Meta-Llama-3-8B-Instruct

(a) Macro-level (Mistral-7B-Instruct-v0.2)

Question Types	Raw Seq.	COT	FS	BAG	BFS	DFS	SP	AL	AM	GraphInsight
N_Count	0.82	0.68	0.44	0.52	0.83	0.83	0.83	0.83	0.24	0.83
Conn	0.62	0.03	0.62	0.09	0.62	0.62	0.62	0.35	0.21	0.62
Cycle	0.53	0.43	0.58	0.42	0.53	0.53	0.53	0.72	0.34	0.60
Max_Wt	0.43	0.45	0.53	0.54	0.43	0.43	0.43	0.45	0.02	0.58
Top_Deg	0.21	0.51	0.55	0.55	0.20	0.20	0.20	0.15	0.10	0.29

(a) Macro-level (Qwen2-7B-Instruct)

Question Types	Raw Seq.	COT	FS	BAG	BFS	DFS	SP	AL	AM	GraphInsight
N_Count	0.84	0.83	0.80	0.82	0.85	0.84	0.84	0.93	0.28	0.83
Conn	0.62	0.26	0.58	0.62	0.62	0.62	0.62	0.71	0.29	0.62
Cycle	0.41	0.39	0.39	0.56	0.46	0.43	0.44	0.70	0.34	0.57
Max_Wt	0.83	0.78	0.78	0.57	0.85	0.83	0.84	0.15	0.00	0.84
Top_Deg	0.12	0.45	0.12	0.66	0.23	0.23	0.23	0.12	0.05	0.29

(b) Micro-level (Mistral-7B-Instruct-v0.2)

Question Types	Raw Seq.	COT	FS	BAG	BFS	DFS	SP	AL	AM	GraphInsight
Dir_Conn	0.68	0.67	0.33	0.70	0.68	0.68	0.68	0.69	0.34	0.65
Deg	0.37	0.47	0.30	0.33	0.37	0.37	0.37	0.38	0.14	0.98
Leaf	0.73	0.70	0.31	0.76	0.73	0.73	0.73	0.68	0.23	0.87
Even_Deg	0.46	0.45	0.39	0.44	0.45	0.45	0.46	0.46	0.24	0.50
Nbrs	0.38	0.37	0.35	0.45	0.38	0.38	0.38	0.39	0.10	0.54
Com_Nbrs	0.51	0.58	0.40	0.53	0.51	0.51	0.51	0.51	0.23	0.61
Deg_Cmp	0.72	0.74	0.73	0.66	0.73	0.73	0.72	0.70	0.32	0.88
Edge_Wt	0.67	0.72	0.54	0.70	0.68	0.68	0.67	0.61	0.00	0.90

(b) Micro-level (Qwen2-7B-Instruct)

Question Types	Raw Seq.	COT	FS	BAG	BFS	DFS	SP	AL	AM	GraphInsight
Dir_Conn	0.73	0.74	0.69	0.75	0.47	0.45	0.45	0.84	0.36	0.91
Deg	0.47	0.50	0.45	0.40	0.45	0.43	0.43	0.42	0.22	0.92
Leaf	0.56	0.62	0.54	0.51	0.32	0.33	0.30	0.45	0.23	0.81
Even_Deg	0.48	0.49	0.48	0.44	0.46	0.44	0.44	0.56	0.26	0.58
Nbrs	0.42	0.41	0.42	0.47	0.51	0.50	0.50	0.45	0.28	0.96
Com_Nbrs	0.52	0.53	0.54	0.42	0.51	0.51	0.50	0.65	0.24	0.73
Deg_Cmp	0.54	0.55	0.47	0.61	0.50	0.50	0.51	0.55	0.24	0.76
Edge_Wt	0.80	0.80	0.75	0.56	0.77	0.77	0.77	0.75	0.00	1.00

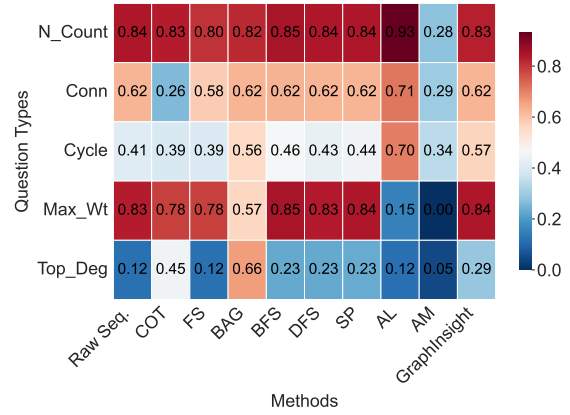Figure 10: Macro- and Micro-level Similarity Analysis for Mistral-7B-Instruct-v0.2

Figure 11: Macro- and Micro-level Similarity Analysis for Qwen2-7B-Instruct

and comprehensive evaluation for set-type answers, effectively capturing both precision and recall in the predictions.

are unlikely to be due to random chance, thereby validating the effectiveness of the proposed framework.
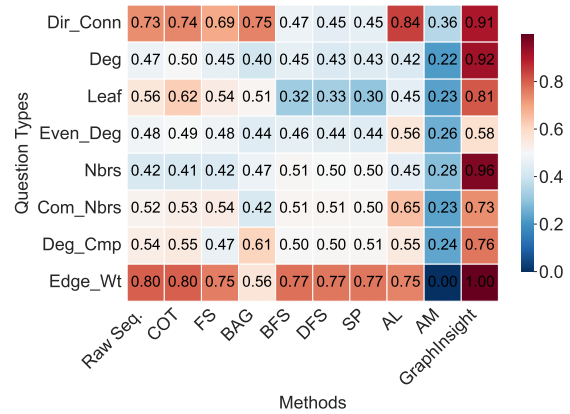
G Performance Analysis of the GraphInsight Framework

The significance of any observed improvement or decline in performance between the proposed GraphInsight framework and the baseline methods is evaluated using appropriate statistical tests. In this study, the Wilcoxon signed-rank test is employed, which is a non-parametric test suitable for comparing paired samples. This test assesses whether the median differences between pairs of observations are statistically significant.

The overall performance comparison between GraphInsight and the baseline method yields a Wilcoxon signed-rank test statistic of 0.0. The corresponding p-value is $6.103515625 \times 10^{-5}$, indicating a statistically significant improvement at the commonly accepted significance level (e.g., $\alpha = 0.05$). The low p-value strongly suggests that the improvements observed with GraphInsight
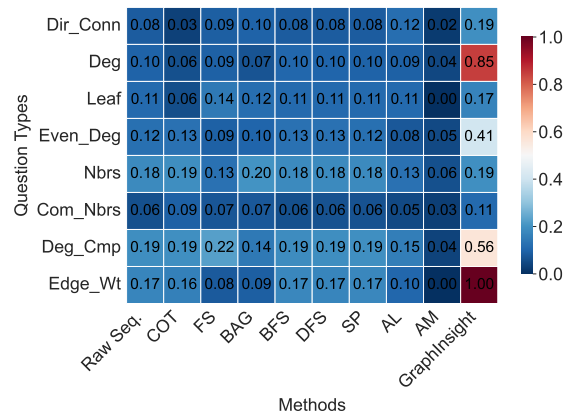
H Discussion

In this section, we discuss the implications of our framework and the results from our empirical evaluations. Our work highlights the potential of optimizing LLMs' graph comprehension through carefully designed graph description sequences, particularly when leveraging the sequential format for transforming graphs into a description form conducive to LLMs' understanding.

One of the key findings is the effectiveness of aligning the LLMs' comprehension distribution with the importance distribution of graph description sequences. As demonstrated in Section 3, by reorganizing graph descriptions to position the most important subgraphs in the strong memory regions of LLMs (i.e., the head and tail of the sequence), we significantly enhance the LLMs' performance in macro-level graph understanding tasks. This improvement stems from the better utilization
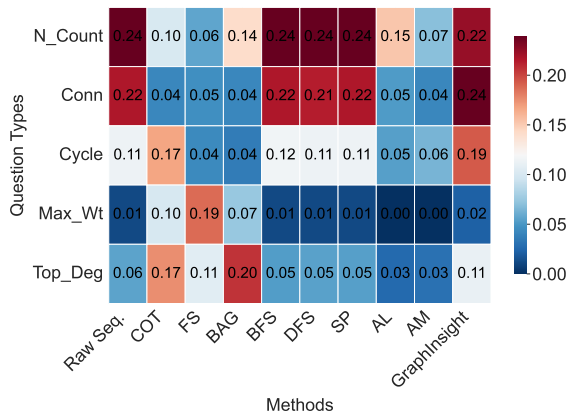
(a) Macro-level (Vicuna-7b-v1.1)

(b) Micro-level (Vicuna-7b-v1.1)

Figure 12: Macro- and Micro-level Similarity Analysis for Vicuna-7b-v1.1

of the LLMs' natural comprehension bias towards the head and tail of sequences, as illustrated by the U-shaped comprehension curve discussed in Definition 2.

Moreover, the decomposition of graph descriptions into subgraph descriptions, centered on high-importance nodes (as measured by PageRank), provides a systematic approach to structuring information in a manner that aligns with the LLMs' strengths. This method not only reinforces the LLMs' ability to process critical structural information but also opens avenues for more complex reasoning tasks that require a deep understanding of local graph structures.

However, several challenges remain. First, while our framework assumes a U-shaped comprehension curve, this may not hold universally across all LLMs or task types. Further research is needed to empirically validate the exact shape of this curve for different models and datasets. Additionally, the process of defining and quantifying the importance of graph structures, while effective in our study, is an open problem and could benefit from more sophisticated techniques beyond PageRank.

In conclusion, our findings underscore the importance of sequence organization in enhancing LLMs' graph comprehension capabilities. By strategically placing high-importance subgraph descriptions in positions that align with the LLMs' natural comprehension tendencies, we can significantly improve performance on a range of graph understanding tasks.

I Task Templates in GraphSQA

In this section, we present the task templates for each category in GraphSQA, as detailed below.

Node Count Identification

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: How many nodes are in this graph?
A: 14

Graph Connectivity

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Is this graph a connected graph?
A: No

Cycle Detection

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Does this graph contain a cycle?
A: Yes

Maximum Weight Identification

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: What is the maximum weight of the edges in this graph?
A: 5

Highest Degree Nodes Identification

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: What are the nodes with the top 3 highest degrees in this graph?
A: [(0, 6), (1, 6), (2, 6)]

Direct Connection Check

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Is there a direct connection between node 8 and node 2?
A: No

Node Degree Calculation

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: What is the degree of node 12?
A: 6

Leaf Node Check

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Is node 0 a leaf node?
A: No

Even Degree Check

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Does node 12 have an even degree?
A: Yes

Neighbor Nodes Identification

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Who are the neighbors of node 12?
A: [7, 8, 9, 10, 11, 13]

Common Neighbors Identification

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Do nodes 7 and 11 have any common neighbors?
A: Yes, node 8.

Degree Comparison

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Is the degree of node 2 greater than the degree of node 6?
A: No

Edge Weight Identification

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 4;
From node 0 to node 3 with weight 3;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 3;
From node 0 to node 6 with weight 5;
From node 1 to node 2 with weight 4;
From node 1 to node 3 with weight 5;
From node 1 to node 4 with weight 5;
From node 1 to node 5 with weight 2;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 3;
From node 2 to node 4 with weight 1;
From node 2 to node 5 with weight 2;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 2;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 3;
From node 4 to node 6 with weight 1;
From node 5 to node 6 with weight 5;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 3;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 4;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 3;
From node 8 to node 10 with weight 3;
From node 8 to node 11 with weight 3;
From node 8 to node 12 with weight 3;
From node 8 to node 13 with weight 2;
From node 9 to node 10 with weight 4;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 5;
From node 9 to node 13 with weight 3;
From node 10 to node 11 with weight 4;
From node 10 to node 12 with weight 3;
From node 10 to node 13 with weight 3;
From node 11 to node 12 with weight 1;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: What is the weight of the edge between node 12 and node 13?
A: 5

Find All Connected Edges

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 1;
From node 0 to node 3 with weight 2;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 4;
From node 0 to node 6 with weight 3;
From node 1 to node 2 with weight 1;
From node 1 to node 3 with weight 2;
From node 1 to node 4 with weight 2;
From node 1 to node 5 with weight 1;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 1;
From node 2 to node 4 with weight 4;
From node 2 to node 5 with weight 3;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 1;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 4;
From node 4 to node 6 with weight 2;
From node 5 to node 6 with weight 1;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 5;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 2;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 5;
From node 8 to node 10 with weight 1;
From node 8 to node 11 with weight 4;
From node 8 to node 12 with weight 5;
From node 8 to node 13 with weight 3;
From node 9 to node 10 with weight 3;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 2;
From node 9 to node 13 with weight 1;
From node 10 to node 11 with weight 1;
From node 10 to node 12 with weight 2;
From node 10 to node 13 with weight 4;
From node 11 to node 12 with weight 5;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Given the edge (8, 11), find all edges connected to it. List the answers in the format of '[(1, 2), (3, 4), ...]'.
A:
$$\begin{bmatrix} (8,7), (8,9), (8,10), (8,11), (8,12), (8,13), \\ (11,7), (11,8), (11,9), (11,10), (11,12), (11,13) \end{bmatrix}$$

Is Complete Subgraph

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 1;
From node 0 to node 3 with weight 2;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 4;
From node 0 to node 6 with weight 3;
From node 1 to node 2 with weight 1;
From node 1 to node 3 with weight 2;
From node 1 to node 4 with weight 2;
From node 1 to node 5 with weight 1;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 1;
From node 2 to node 4 with weight 4;
From node 2 to node 5 with weight 3;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 1;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 4;
From node 4 to node 6 with weight 2;
From node 5 to node 6 with weight 1;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 5;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 2;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 5;
From node 8 to node 10 with weight 1;
From node 8 to node 11 with weight 4;
From node 8 to node 12 with weight 5;
From node 8 to node 13 with weight 3;
From node 9 to node 10 with weight 3;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 2;
From node 9 to node 13 with weight 1;
From node 10 to node 11 with weight 1;
From node 10 to node 12 with weight 2;
From node 10 to node 13 with weight 4;
From node 11 to node 12 with weight 5;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Given the nodes [0, 1, 6], determine if they form a complete subgraph. List the answer directly in the format of 'Yes' or 'No'.
A: Yes

Find Highest Degree Neighbor of Neighbors

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 1;
From node 0 to node 3 with weight 2;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 4;
From node 0 to node 6 with weight 3;
From node 1 to node 2 with weight 1;
From node 1 to node 3 with weight 2;
From node 1 to node 4 with weight 2;
From node 1 to node 5 with weight 1;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 1;
From node 2 to node 4 with weight 4;
From node 2 to node 5 with weight 3;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 1;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 4;
From node 4 to node 6 with weight 2;
From node 5 to node 6 with weight 1;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 5;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 2;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 5;
From node 8 to node 10 with weight 1;
From node 8 to node 11 with weight 4;
From node 8 to node 12 with weight 5;
From node 8 to node 13 with weight 3;
From node 9 to node 10 with weight 3;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 2;
From node 9 to node 13 with weight 1;
From node 10 to node 11 with weight 1;
From node 10 to node 12 with weight 2;
From node 10 to node 13 with weight 4;
From node 11 to node 12 with weight 5;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Given the node 9, find the neighbor's neighbor with the highest degree. List the answer directly as the node id.
A: 8

Find K-Order Neighbors

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 1;
From node 0 to node 3 with weight 2;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 4;
From node 0 to node 6 with weight 3;
From node 1 to node 2 with weight 1;
From node 1 to node 3 with weight 2;
From node 1 to node 4 with weight 2;
From node 1 to node 5 with weight 1;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 1;
From node 2 to node 4 with weight 4;
From node 2 to node 5 with weight 3;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 1;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 4;
From node 4 to node 6 with weight 2;
From node 5 to node 6 with weight 1;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 5;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 2;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 5;
From node 8 to node 10 with weight 1;
From node 8 to node 11 with weight 4;
From node 8 to node 12 with weight 5;
From node 8 to node 13 with weight 3;
From node 9 to node 10 with weight 3;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 2;
From node 9 to node 13 with weight 1;
From node 10 to node 11 with weight 1;
From node 10 to node 12 with weight 2;
From node 10 to node 13 with weight 4;
From node 11 to node 12 with weight 5;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Given the node 7, find all its 2-order neighbors. Note that the 2-order neighbors do not include the 1-order neighbors, and so on. List the answers in the format of '[1, 2, ...]'.
A: []

Find Direct Neighbors of Specified Node

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 1;
From node 0 to node 3 with weight 2;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 4;
From node 0 to node 6 with weight 3;
From node 1 to node 2 with weight 1;
From node 1 to node 3 with weight 2;
From node 1 to node 4 with weight 2;
From node 1 to node 5 with weight 1;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 1;
From node 2 to node 4 with weight 4;
From node 2 to node 5 with weight 3;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 1;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 4;
From node 4 to node 6 with weight 2;
From node 5 to node 6 with weight 1;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 5;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 2;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 5;
From node 8 to node 10 with weight 1;
From node 8 to node 11 with weight 4;
From node 8 to node 12 with weight 5;
From node 8 to node 13 with weight 3;
From node 9 to node 10 with weight 3;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 2;
From node 9 to node 13 with weight 1;
From node 10 to node 11 with weight 1;
From node 10 to node 12 with weight 2;
From node 10 to node 13 with weight 4;
From node 11 to node 12 with weight 5;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Given the node 11, find its neighbors that are directly connected to node 3. List the answers in the format of '[1, 2, ...]'.
A: []

Find Connected Neighbor Pairs

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 1;
From node 0 to node 3 with weight 2;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 4;
From node 0 to node 6 with weight 3;
From node 1 to node 2 with weight 1;
From node 1 to node 3 with weight 2;
From node 1 to node 4 with weight 2;
From node 1 to node 5 with weight 1;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 1;
From node 2 to node 4 with weight 4;
From node 2 to node 5 with weight 3;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 1;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 4;
From node 4 to node 6 with weight 2;
From node 5 to node 6 with weight 1;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 5;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 2;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 5;
From node 8 to node 10 with weight 1;
From node 8 to node 11 with weight 4;
From node 8 to node 12 with weight 5;
From node 8 to node 13 with weight 3;
From node 9 to node 10 with weight 3;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 2;
From node 9 to node 13 with weight 1;
From node 10 to node 11 with weight 1;
From node 10 to node 12 with weight 2;
From node 10 to node 13 with weight 4;
From node 11 to node 12 with weight 5;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Given the node 2, find all connected pairs among its neighbors. List the answers in the format of '[(1, 2), (3, 4), ...]'.
A: [(0, 1), (0, 3), (0, 4), (0, 5), (0, 6), (1, 3), (1, 4), (1, 5), (1, 6), (3, 4), (3, 5), (3, 6), (4, 5), (4, 6), (5, 6)]

Find Common Neighbors of Edge Nodes

This is an undirected graph with the following edges:
From node 0 to node 1 with weight 4;
From node 0 to node 2 with weight 1;
From node 0 to node 3 with weight 2;
From node 0 to node 4 with weight 5;
From node 0 to node 5 with weight 4;
From node 0 to node 6 with weight 3;
From node 1 to node 2 with weight 1;
From node 1 to node 3 with weight 2;
From node 1 to node 4 with weight 2;
From node 1 to node 5 with weight 1;
From node 1 to node 6 with weight 1;
From node 2 to node 3 with weight 1;
From node 2 to node 4 with weight 4;
From node 2 to node 5 with weight 3;
From node 2 to node 6 with weight 4;
From node 3 to node 4 with weight 1;
From node 3 to node 5 with weight 1;
From node 3 to node 6 with weight 5;
From node 4 to node 5 with weight 4;
From node 4 to node 6 with weight 2;
From node 5 to node 6 with weight 1;
From node 7 to node 8 with weight 3;
From node 7 to node 9 with weight 5;
From node 7 to node 10 with weight 5;
From node 7 to node 11 with weight 2;
From node 7 to node 12 with weight 2;
From node 7 to node 13 with weight 4;
From node 8 to node 9 with weight 5;
From node 8 to node 10 with weight 1;
From node 8 to node 11 with weight 4;
From node 8 to node 12 with weight 5;
From node 8 to node 13 with weight 3;
From node 9 to node 10 with weight 3;
From node 9 to node 11 with weight 2;
From node 9 to node 12 with weight 2;
From node 9 to node 13 with weight 1;
From node 10 to node 11 with weight 1;
From node 10 to node 12 with weight 2;
From node 10 to node 13 with weight 4;
From node 11 to node 12 with weight 5;
From node 11 to node 13 with weight 5;
From node 12 to node 13 with weight 5;

Q: Given the edge (1, 5), find all common neighbors of its two end nodes. List the answers in the format of '[1, 2, ...]'.
A: [0, 2, 3, 4, 6]