Can Graph Descriptive Order Affect Solving Graph Problems with LLMs?

Anonymous ACL submission

Abstract

Large language models (LLMs) have achieved significant success in reasoning tasks, including mathematical reasoning and logical deduction. Among these reasoning tasks, graph problems stand out due to their complexity and unique structural characteristics, attracting considerable attention from researchers. Previous studies have explored LLMs' graph reasoning abilities through various techniques, such as different encoding methods for graph structures and the use of carefully designed prompts. However, a critical factor has been mostly overlooked: the prompt sequential order in which graph descriptions are presented to the models. In this study, we present the first comprehensive analysis of how the order of graph descriptions impacts LLM performance. Specifically, we comprehensively evaluate four graph description orders across six graph problems using six mainstream LLMs. The results reveal that: (1) ordered graph descriptions significantly improve LLMs' comprehension of graph structures; (2) the robustness of LLMs to graph description order varies across different tasks; and (3) the impact of graph order on performance is closely related to the inherent characteristics of tasks. This study provides a critical advancement in the application of LLMs for solving graph-related problems, paving the way for future research to optimize model performance through strategic graph description ordering.

1 Introduction

Large language models (LLMs) have made remarkable progress, showing unprecedented capabilities in NLP (Vaswani et al., 2017; Devlin et al., 2018; Brown et al., 2020; Ouyang et al., 2022). Leveraging advancements in NLP, LLMs excel in reasoning tasks, which has drawn considerable interest from researchers. As a type of complex reasoning problem, graph problems have also attracted substantial attention. For instance, Wang et al. (2023a) represented graphs in natural language and validated the effectiveness of prompts for graph reasoning tasks. Fatemi et al. (2023) map pure graphs in a real-world scenario to understand how LLMs' learned representations are leveraged in graph tasks. More recently, Skianis et al. (2024) explore the use of pseudo-code instructions to enhance LLMs' ability to solve graph problems.

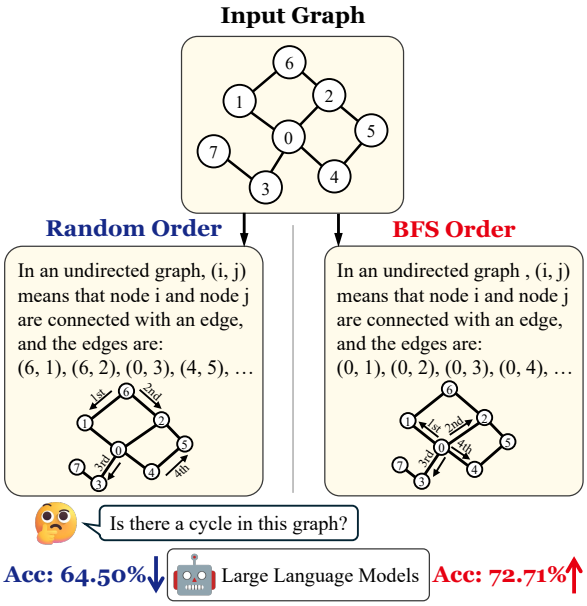


Figure 1: The order in which graphs are described significantly affects LLMs' ability to understand and solve graph problems. For instance, in the cycle detection task, graphs described in BFS order achieved an average accuracy improvement of 12.73% over those described in random order.

Despite the significant contributions of previous researchers, one key issue remains overlooked: **the order of graph descriptions may affect LLMs' performance in solving graph problems**. They typically employed randomly arranged graph descriptions, overlooking the critical role that description order may play. While graphs have no fixed textual representation, the order in which their components are expressed may affect the model's rea-

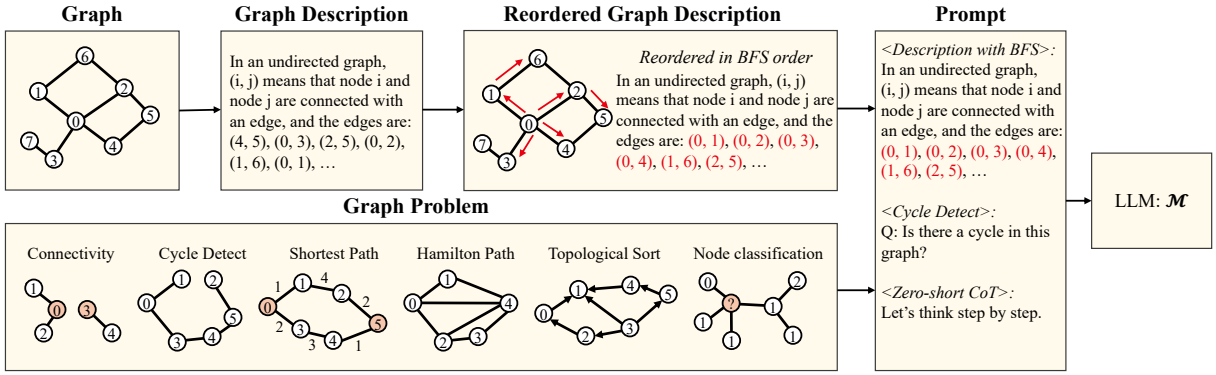


Figure 2: Overview of our framework for solving graph problems with LLMs. *In node classification task, node labels no longer represent identifiers; instead, they indicate the categories the nodes belong to.*

soning process. Different orders of descriptions may emphasize specific paths or parts, providing LLMs with different perspectives on the same graph structure.

In this work, we explore the impact of the order of graph description in solving graph problems with LLMs. For comprehensive study, we design four graph description orders and categorized them into graph traversal-based orders, including BFS and DFS, and probability distribution-based orders, including PageRank and Personalized PageRank. These orders were carefully chosen to provide LLMs different perspectives on graph understanding: BFS provides a hierarchical traversal, DFS offers a deep traversal, PageRank delivers a global probability distribution of node importance, and Personalized PageRank focuses on a localized probability distribution. Given that certain orders may influence specific graph reasoning tasks, we design six graph tasks spanning varying levels of complexity, and we conducted experiments on six mainstream LLMs.

Our main contributions are as follows:

- We are the first to demonstrate that the order of graph descriptions significantly affects the graph reasoning performance of LLMs.
- Through extensive experimentation, we analyzed the differential impact of description orders on LLMs' performance across diverse graph reasoning tasks.
- We introduced the GraphDO (**Graph Description with Order**), a novel dataset consisting of a set of graphs, corresponding prompts, and predefined description orders, which aims to advance the community's understanding of how graph description impacts reasoning in LLMs.

2 Preliminary

2.1 Prompt Engineering for Graph

Prompt engineering involves strategically designing task-specific instructions, referred to as prompts, to guide model output without altering parameters (Sahoo et al., 2024; Collobert and Weston, 2008; Mikolov et al., 2013; Sutskever et al., 2014).

In this work, we consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of nodes and $\mathcal{E}$ the set of edges. For encoding graph to text, we define the graph encoding function $g(\mathcal{G}, o)$, which maps a graph $\mathcal{G}$ and a description order $o \in \mathcal{O}$ to a description in natural language. Additionally, let $q(T)$ be a function that generates a question Q based on a graph task T, such that $q : T \mapsto Q$, where Q has a standard answer Y. Graph prompting engineering can be formally expressed as an optimization problem focused on determining the optimal order o that maximizes the LLM $\mathcal{M}$'s scoring function $\mathcal{S}$, and is formulated as:

$$\max_{o \in \mathcal{O}} \mathbb{E}_{\mathcal{G},T,Y \in D}\, \mathcal{S}(\mathcal{M}(p, g(\mathcal{G}, o), q(T)), Y) \quad (1)$$

where $p \in \mathcal{P}$ represents the prompt style, D is the dataset consisting of triples $(\mathcal{G}, T, Y)$.

To ensure the completeness of our experiments, we set five prompt styles to offer varying levels of heuristic reasoning to LLMs, with more details in Appendix A.1.

2.2 Graph Problems

We designed six graph reasoning tasks, covering various levels of reasoning complexity and categories.

T1 Connectivity In an undirected graph, LLMs need to determine whether a path exists between

two arbitrary nodes $u, v \in \mathcal{V}$, which is a binary classification problem.

T2 Cycle In an undirected graph, LLMs need to determine if a non-empty path exists where the starting and ending nodes are the same, which is a binary classification problem.

T3 Hamilton Path A Hamilton path visits each node in $\mathcal{V}$ exactly once. The LLMs need to determine whether such a path exists in an undirected graph $\mathcal{G}$ and, if such a path exists, answer the path.

T4 Shortest Path In an undirected graph, LLMs need to answer the complete shortest path between two nodes $u, v \in \mathcal{V}$.

T5 Topological Sort In a directed graph $\mathcal{G}$, LLMs need to generate a linear ordering of the nodes such that for every directed edge $(u, v) \in \mathcal{E}$, node u precedes node v in the ordering. This task requires finding any valid topological sort of the graph, and multiple correct solutions may exist.

T6 Node Classification In an undirected graph composed of nodes with labels, LLMs need to predict the label of a certain node which is labeled as '?' based on the labels of its neighbouring nodes. There is only one node labeled as '?' in each graph.

The T1-T5 tasks focus on pure graph structures to evaluate LLMs' understanding of graphs. Specifically, T1 and T2 assess local reasoning, while T3 to T5 examine global graph understanding. T6 shifts the focus to graph attribute learning, making it more relevant for real-world applications.

3 Graph Description Generation

3.1 Graph Encoder

Graphs can be described in text through multiple encoding methods. Fatemi et al. (2023) compared several approaches for converting graph data into text sequences. Given that our research focuses on the order of graph descriptions, we adopted the adjacency format, which uses edge lists to represent graphs and can be applied to both pure and attributed graphs, making it ideal for our study.

To encode an edge list $\mathcal{L}_o$ into a graph description in adjacency format, we use a template function $\mathcal{T}(\cdot)$. The process is formalized as:

$$g(\mathcal{G}, o) = \mathcal{T}(\mathcal{G}, \mathcal{L}_o), o \in \mathcal{O} \quad (2)$$

For pure unweighted graphs (*i.e.*, cycle detection, connectivity detection, shortest path, Hamilton path), the definition of $\mathcal{T}(\cdot)$ function is as follows:

Prompt Template for Unweighted Graphs

In an undirected/directed graph, (i, j) means that node i and node j are connected with an edge, and the edges are: [(0, 1), (1, 3), (3, 5), ...].

For pure weighted graphs (*i.e.*, shortest path), we define the $\mathcal{T}(\cdot)$ as:

Prompt Template for Weighted Graphs

In an undirected/directed graph, (i, j, w) means that node i and node j are connected by an edge with weight w, and the edges are: [(1, 3, 2), (0, 3, 1), (0, 1, 4), ...].

For graphs composed of nodes with labels (*i.e.*, node classification), inspired by Das et al. (2023), we define the $\mathcal{T}(\cdot)$ as:

Prompt Template for Node Classification Task

Adjacency list: [(1758, 2217), (2217, 2645), ...]
Node to label mapping: node 1758: label 3 | node 2217: label 2 | node 2645: label ? | ...

3.2 Graph Description Ordering

Beyond the random order, we designed four additional graph description orders for our main experiment. Furthermore, two more description orders were employed in the deeper exploration, which will be discussed in the corresponding section.

Random Order In random order, the edges $\mathcal{E}$ of the graph are shuffled randomly. In previous works, researchers have commonly employed random graph description orders (*e.g.*, (Wang et al., 2023a; Fatemi et al., 2023; Das et al., 2023)).

Breadth-First Search (BFS) Order Starting from a random root node $v_0 \in \mathcal{V}$, BFS generates a sequence of edges by exploring the graph level by level. At each level, for each node v, the edges (v, u), where $u \in \mathcal{N}(v)$, are added to the sequence before moving to the next level.

Depth-First Search (DFS) Order Starting from a node $v_0 \in \mathcal{V}$, DFS follows a recursive strategy, generating a sequence of edges by traversing as deeply as possible before backtracking.

PageRank (PR) Order In PR order, nodes $v \in \mathcal{V}$ are sorted in descending order by their PageRank scores $PR(v)$, where $PR(v) = \alpha \sum_{u \in \mathcal{N}^{-1}(v)} \frac{PR(u)}{|\mathcal{N}(u)|} + (1 - \alpha)$, with $\alpha = 0.85$ as the damping factor and $\mathcal{N}^{-1}(v)$ as the nodes linking to v. For each node, starting with the highest-ranked, edges to its neighbors $u \in \mathcal{N}(v)$ are added

to the edge list $\mathcal{L}_{PR}$. If an edge (v, u) or (u, v) is already in $\mathcal{L}_{PR}$, it is skipped.

Personalized PageRank (PPR) Order PPR introduces a personalization vector mechanism that prioritizes proximity to specific target nodes. The ranking is computed as $PR_S(v) = \alpha \sum_{u \in \mathcal{N}^{-1}(v)} \frac{PR_S(u)}{|\mathcal{N}(u)|} + (1-\alpha) \cdot e_v$, where the parameter e_v is task-specific and its definition can be found in the Appendix A.2. The subsequent computations follow the same process as in PageRank.

Note. For DFS and BFS order, when traversing the graph $\mathcal{G}$, it is not guaranteed that all edges $e \in \mathcal{E}$ will be included in the edge list $\mathcal{L}$. To avoid this, we perform a traversal on the dual graph $\mathcal{G}^*$ of $\mathcal{G}$ to ensure that the resulting edge list includes all the edges in $\mathcal{G}$. For a disconnected graph, the root node will be reselected randomly until the graph is fully described. This method does not alter the topology of $\mathcal{G}$; it merely serves as a means of obtaining $\mathcal{L}$.

4 Experiments

4.1 Experimental Settings

Datasets Our experiments are conducted on the GraphDO dataset, introduced in this paper, which includes six graph tasks. GraphDO consists of 8,500 cases, with each case containing a graph description, a question, and an answer. Each graph description is generated in a specific order. For traditional graph tasks (e.g., cycle detection, connectivity detection, shortest path, Hamilton path, topological sort), we employ the Erdős-Rényi (ER) graph generation method. We apply a filtering process to the generated graphs to ensure that each case has a valid and well-defined solution. For the graph learning task (e.g., node classification), we conduct experiments on attributed graphs using three widely recognized datasets: CORA (McCallum et al., 2000), Citeseer (Giles et al., 1998), and Pubmed (Sen et al., 2008). Since the sizes of these real-world citation graphs exceed the input limits of LLMs, we employ graph sampling methods, including ego-graph (Ego) and forest fire sampling (FF). Since the node classification task requires fewer reasoning steps than traditional graph tasks, we set the default prompt style to zero-shot. Additional details about the GraphDO dataset can be found in Appendix B.

Models and Settings We use the GPT-3.5-TURBO-0613 as the default model (Brown et al., 2020). To ensure the generality of our conclusions, we also conducte experiments on other models, including LLAMA2-7B-CHAT, LLAMA2-13B-CHAT (Touvron et al., 2023a), QWEN2-7B (Yang et al., 2024), MISTRAL-7B (Jiang et al., 2023), and VICUNA-7B-V1.5 (Zheng et al., 2023). The decoding temperature is set to zero.

Metric Performance is measured by accuracy, defined as:

$$\text{Acc} = \frac{\#\textit{correct answers}}{\#\textit{total questions}} \tag{3}$$

where # represents the number of instances.

More judgment details are in Appendix A.3.

Baseline We use the random order graph description as a baseline to facilitate comparisons with ordered descriptions.

4.2 Main Result

(Q1) Does the order of graph description impact the LLM's performance in solving graph problems? As presented in Table 1, ordered graph descriptions consistently outperform the random baseline across all traditional graph tasks and prompt configurations. For instance, in the connectivity task, the BFS order achieves an average accuracy of 89.43%, significantly higher than the random order's 78.36%. Similarly, in the cycle detection task, the BFS order reaches an accuracy of 72.71%, compared to the random order's 64.50%. Figure 3 further illustrates that the random order consistently yields the lowest accuracy across tasks.

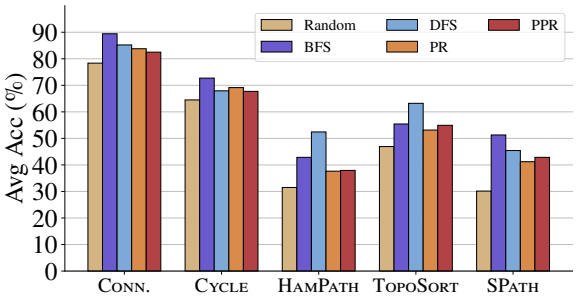


Figure 3: The LLM's average accuracy in solving various tasks across different orders.

As presented in Table 2, ordered descriptions also consistently outperform the baseline in node classification task. Specifically, in the CORA dataset with ego-graph sampling, PR order achieves 75.33% accuracy, compared to 70.00% for the random order. Similarly, in the Pubmed dataset, PR order reaches 82.67%, which significantly surpasses the random order's 72.00%.

Task	Order	Zero-shot	Zero-shot CoT	Few-shot	CoT	CoT-BAG	Avg.
Conn.	Random	$73.93_{(-)}$	$70.71_{(-)}$	$81.07_{(-)}$	$83.93_{(-)}$	$82.14_{(-)}$	$78.36_{(-)}$
	BFS	$82.14_{(\uparrow 11.11)}$	$87.50_{(\uparrow 23.74)}$	$89.29_{(\uparrow 10.14)}$	$92.50_{(\uparrow 10.21)}$	$95.71_{(\uparrow 16.52)}$	$89.43_{(\uparrow 14.13)}$
	DFS	$79.29_{(\uparrow 7.25)}$	$82.14_{(\uparrow 16.16)}$	$87.14_{(\uparrow 7.49)}$	$88.21_{(\uparrow 5.10)}$	$89.29_{(\uparrow 8.70)}$	$85.21_{(\uparrow 8.75)}$
	PR	$77.86_{(\uparrow 5.32)}$	$83.57_{(\uparrow 18.19)}$	$85.71_{(\uparrow 5.72)}$	$84.29_{(\uparrow 0.43)}$	$87.50_{(\uparrow 6.53)}$	$83.79_{(\uparrow 6.93)}$
	PPR	$76.79_{(\uparrow 3.87)}$	$81.07_{(\uparrow 14.65)}$	$83.93_{(\uparrow 3.53)}$	$84.64_{(\uparrow 0.85)}$	$86.07_{(\uparrow 4.78)}$	$82.50_{(\uparrow 5.29)}$
Cycle	Random	$51.79_{(-)}$	$53.57_{(-)}$	$65.36_{(-)}$	$75.71_{(-)}$	$76.07_{(-)}$	$64.50_{(-)}$
	BFS	$55.71_{(\uparrow 7.57)}$	$56.07_{(\uparrow 4.67)}$	$79.29_{(\uparrow 21.31)}$	$86.07_{(\uparrow 13.68)}$	$86.43_{(\uparrow 13.62)}$	$72.71_{(\uparrow 12.73)}$
	DFS	$52.14_{(\uparrow 0.68)}$	$53.93_{(\uparrow 0.67)}$	$73.21_{(\uparrow 12.01)}$	$79.29_{(\uparrow 4.73)}$	$81.07_{(\uparrow 6.57)}$	$67.93_{(\uparrow 5.31)}$
	PR	$55.36_{(\uparrow 6.89)}$	$56.43_{(\uparrow 5.33)}$	$70.36_{(\uparrow 7.65)}$	$80.36_{(\uparrow 6.14)}$	$83.21_{(\uparrow 9.39)}$	$69.14_{(\uparrow 7.20)}$
	PPR	$54.29_{(\uparrow 4.83)}$	$55.00_{(\uparrow 2.67)}$	$70.00_{(\uparrow 7.10)}$	$79.29_{(\uparrow 4.73)}$	$80.00_{(\uparrow 5.17)}$	$67.72_{(\uparrow 4.99)}$
HamPath	Random	$10.71_{(-)}$	$15.36_{(-)}$	$40.00_{(-)}$	$46.07_{(-)}$	$45.36_{(-)}$	$31.50_{(-)}$
	BFS	$20.00_{(\uparrow 86.74)}$	$20.71_{(\uparrow 34.83)}$	$57.86_{(\uparrow 44.65)}$	$58.57_{(\uparrow 27.13)}$	$57.14_{(\uparrow 25.97)}$	$42.86_{(\uparrow 36.05)}$
	DFS	$33.93_{(\uparrow 216.81)}$	$37.50_{(\uparrow 144.14)}$	$67.50_{(\uparrow 68.75)}$	$63.93_{(\uparrow 38.77)}$	$59.29_{(\uparrow 30.71)}$	$52.43_{(\uparrow 66.44)}$
	PR	$15.00_{(\uparrow 40.06)}$	$19.29_{(\uparrow 25.59)}$	$48.93_{(\uparrow 22.32)}$	$55.00_{(\uparrow 19.38)}$	$50.00_{(\uparrow 10.23)}$	$37.64_{(\uparrow 19.50)}$
	PPR	$16.43_{(\uparrow 53.41)}$	$18.93_{(\uparrow 23.24)}$	$50.00_{(\uparrow 25.00)}$	$53.93_{(\uparrow 17.06)}$	$50.36_{(\uparrow 11.02)}$	$37.93_{(\uparrow 20.41)}$
TopoSort	Random	$28.93_{(-)}$	$31.07_{(-)}$	$58.21_{(-)}$	$56.07_{(-)}$	$60.36_{(-)}$	$46.93_{(-)}$
	BFS	$43.21_{(\uparrow 49.36)}$	$40.36_{(\uparrow 29.90)}$	$67.14_{(\uparrow 15.34)}$	$61.43_{(\uparrow 9.56)}$	$65.00_{(\uparrow 7.69)}$	$55.43_{(\uparrow 18.11)}$
	DFS	$42.14_{(\uparrow 45.66)}$	$48.93_{(\uparrow 57.48)}$	$77.86_{(\uparrow 33.76)}$	$74.29_{(\uparrow 32.50)}$	$72.86_{(\uparrow 20.71)}$	$63.21_{(\uparrow 34.71)}$
	PR	$35.36_{(\uparrow 22.23)}$	$35.71_{(\uparrow 14.93)}$	$71.07_{(\uparrow 22.09)}$	$58.21_{(\uparrow 3.82)}$	$65.36_{(\uparrow 8.28)}$	$53.14_{(\uparrow 13.24)}$
	PPR	$37.14_{(\uparrow 28.38)}$	$39.64_{(\uparrow 27.58)}$	$72.50_{(\uparrow 24.55)}$	$58.93_{(\uparrow 5.10)}$	$66.43_{(\uparrow 10.06)}$	$54.93_{(\uparrow 17.05)}$
SPath	Random	$20.00_{(-)}$	$25.00_{(-)}$	$26.07_{(-)}$	$38.93_{(-)}$	$40.71_{(-)}$	$30.14_{(-)}$
	BFS	$35.36_{(\uparrow 76.80)}$	$42.50_{(\uparrow 70.00)}$	$45.36_{(\uparrow 73.99)}$	$67.50_{(\uparrow 73.39)}$	$65.71_{(\uparrow 61.41)}$	$51.29_{(\uparrow 70.15)}$
	DFS	$32.14_{(\uparrow 60.70)}$	$34.29_{(\uparrow 37.16)}$	$45.00_{(\uparrow 72.61)}$	$58.57_{(\uparrow 50.45)}$	$57.14_{(\uparrow 40.36)}$	$45.43_{(\uparrow 50.71)}$
	PR	$30.36_{(\uparrow 51.80)}$	$43.93_{(\uparrow 75.72)}$	$38.93_{(\uparrow 49.33)}$	$43.93_{(\uparrow 12.84)}$	$48.93_{(\uparrow 20.19)}$	$41.21_{(\uparrow 36.74)}$
	PPR	$32.50_{(\uparrow 62.50)}$	$44.64_{(\uparrow 78.56)}$	$42.14_{(\uparrow 61.64)}$	$45.36_{(\uparrow 16.52)}$	$49.64_{(\uparrow 21.94)}$	$42.86_{(\uparrow 42.18)}$

Table 1: Results of the performance of various orders on different graph tasks. (↑) indicates the improvement compared to the baseline under the same setting.

However, the improvements in node classification task is generally less pronounced than those observed in traditional graph tasks. For example, in the Pubmed dataset with ego-graph sampling, the PR order improves accuracy by 14.82%, which is the largest improvment in node classification tasks. This suggests that, while ordered descriptions are indeed advantageous, their relatively smaller impact on node classification tasks may be attributed to inherent differences in the complexity and reasoning patterns between these tasks.

We hypothesize that LLMs' improved performance with ordered descriptions stems from limitations in positional encoding and attention mechanisms—what we call attention bias. Positional encodings, which are meant to provide sequence information in transformer models, may not effectively capture the structural complexity of graph data when the input sequence is unordered. Guided by positional encodings, attention mechanisms can give undue priority to certain sections of the input based on their order in the sequence, leading to over-dependence on the graph description.

(Q2) Is the robustness of LLM to graph description order consistent across different tasks? As presented in Figure 4, the variance in LLM performance across different graph description orders reveals a clear pattern: simpler tasks, such as connectivity and cycle detection, consistently exhibit low variance, indicating greater robustness in LLM reasoning. In contrast, more complex tasks like Hamilton path, topological sort, and shortest path show significantly higher variance, reflecting their increased sensitivity to graph description order. Notably, the shortest path task has the highest variance, as it is the only task that requires reasoning on weighted graphs, where changes in graph descrip-

Sampling	Order	CORA		Citeseer		Pubmed	
		Acc.	Δ	Acc.	Δ	Acc.	Δ
Ego	Random	70.00	-	67.33	-	72.00	-
	BFS	72.00	↑ 2.86	68.67	↑ 1.99	74.00	↑ 2.78
	DFS	71.33	↑ 1.90	68.66	↑ 1.98	77.33	↑ 7.40
	PR	75.33	↑ 7.61	71.33	↑ 5.94	82.67	↑ 14.82
	PPR	73.33	↑ 4.76	69.33	↑ 2.97	77.33	↑ 7.40
Forest Fire	Random	79.33	-	68.67	-	69.99	-
	BFS	82.67	↑ 4.21	71.33	↑ 3.87	74.00	↑ 5.73
	DFS	81.33	↑ 2.52	70.00	↑ 1.94	76.00	↑ 8.59
	PR	83.33	↑ 5.04	71.33	↑ 3.87	76.00	↑ 8.59
	PPR	82.00	↑ 3.36	70.67	↑ 2.91	74.67	↑ 6.69

Table 2: The accuracy of the LLM in solving node classification task across various orders, datasets, and sampling methods. ↑ indicates the improvement compared to the baseline under the same setting.

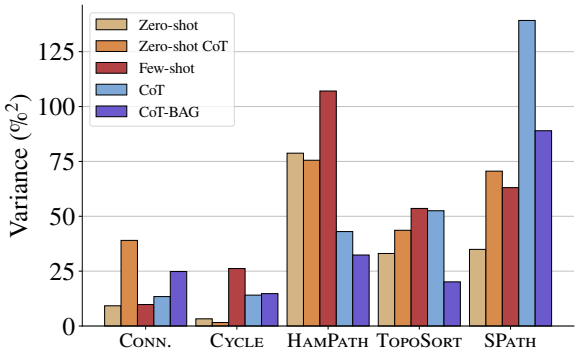


Figure 4: Variance of LLM accuracy across different graph tasks with varying description orders. The variance for each task is computed as $\sigma^2 = \frac{1}{|\mathcal{O}|}\sum_{o\in\mathcal{O}}(\mathcal{S}_o - \mu)^2$, where $\mathcal{S}_o$ is the accuracy for order o, μ is the mean accuracy across all orders.

tion greatly affect task complexity.

The robustness in simpler tasks likely stems from their reliance on local substructures, minimizing the need for global reasoning and allowing LLMs to focus on individual graph components without considering the overall structure. As a result, even when the graph description order varies, the model can extract the necessary information with minimal disruption. In contrast, the higher variance in more complex tasks can be attributed to the need for global reasoning, which is heavily influenced by the order of the input sequence. Counterintuitively, even when using CoT to encourage LLMs to engage in rational thinking and provide reasoning steps, the variance did not significantly decrease. A reason for this phenomenon could be that CoT encourages the LLMs for "slow thinking" about the question, but does not necessarily mitigate the attention bias to graph structure during CoT reasoning steps (Wang et al., 2023b).

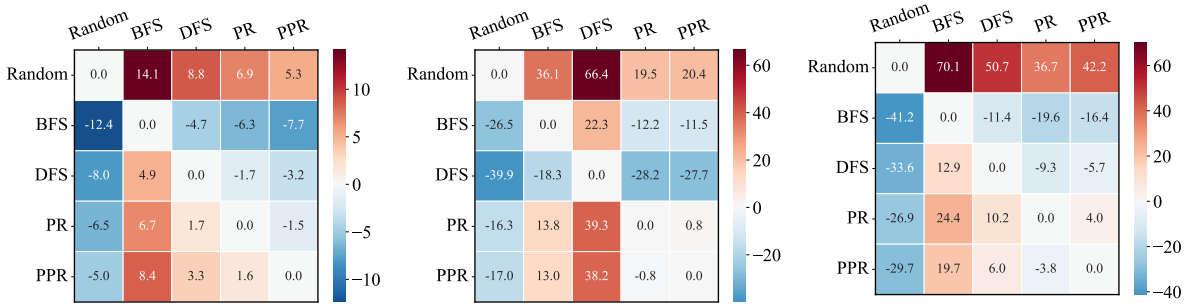
(Q3) Does a specific graph description order favor certain graph tasks? As presented in Table 1, BFS generally outperforms DFS in tasks like cycle detection, connectivity detection, and shortest path. For example, BFS achieves 72.71% accuracy in cycle detection, 7.04% higher than DFS achieves 67.93%. Conversely, for tasks requiring deeper exploration, such as topological sort and Hamilton path, DFS performs better.

Figure 5 provides further insights. In connectivity, BFS exceeds the random baseline by 14.1%, while DFS improves upon it by 8.8%. In the shortest path task, BFS improves accuracy by 70.1%, outperforming DFS by 12.9%. Conversely, in the Hamilton path task, DFS surpasses the random order by 66.4% and outperforms BFS by 22.3%.

Tasks like cycle and connectivity detection, which focus on local connectivity, benefit from BFS's level-wise traversal, allowing the LLM to efficiently extract adjacent connections and form accurate local representations. For shortest path problems, BFS ensures the shortest path is found once the target node is reached. In contrast, tasks like Hamilton path and topological sort require a deeper understanding of global structures, where DFS excels by thoroughly exploring paths and capturing global dependencies.

For node classification task, as demonstrated in Table 2, the PR order consistently outperforms the PPR across all datasets, while PPR generally performs better than traversal-based orders.

We hypothesize that when LLMs reason based on PR-ordered graph descriptions, their focus on local features may lead to overfitting to the local



(a) Connectivity (b) Hamilton Path (c) Shortest Path

Figure 5: The improvement of average accuracy (calculated as the mean across all prompt types) of the LLM between a graph description in one order (horizontal axis) and its average accuracy on graph descriptions in other orders (vertical axis).

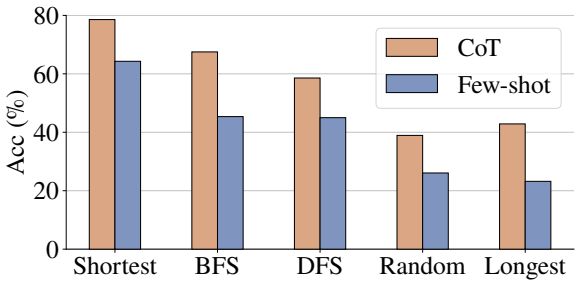


Figure 6: Results of the accuracy of various orders on shortest path task.

neighborhood, thereby limiting their ability to capture broader patterns. Additionally, node classification task often require a more comprehensive understanding of the global graph structure, as nodes within the same category may be distributed across multiple regions, making local information inadequate for accurate classification. As for why probability distribution-based orders outperform traversal-based orders, we believe it is because the classification of the query node is influenced by its neighboring nodes, making it less suitable to infer from traversal-based graph descriptions.

5 Deeper Exploration

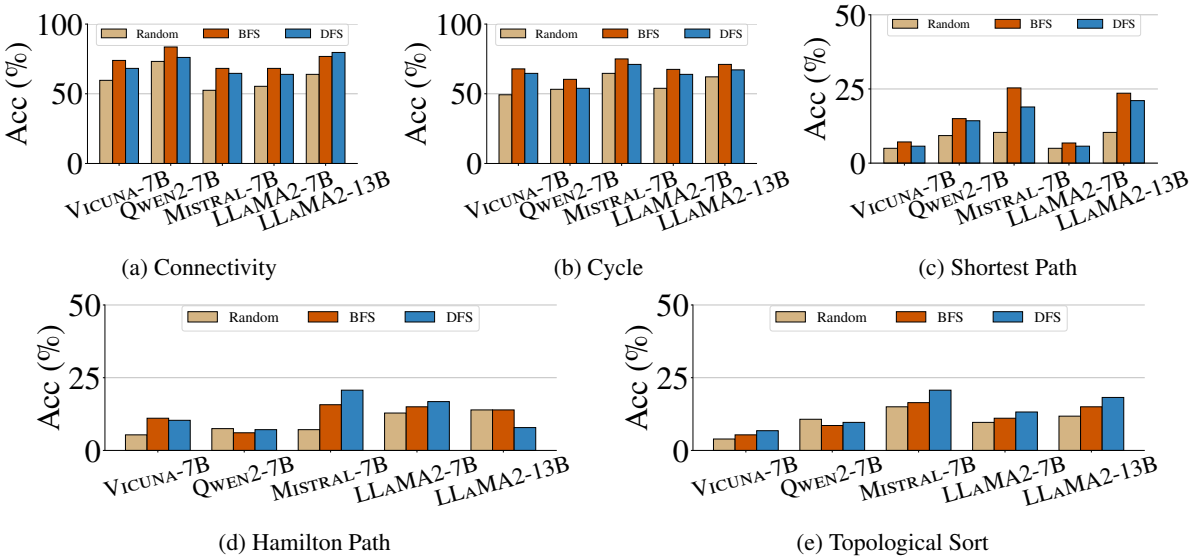
Better graph understanding or just more overlap with the answer? Inspired by the finding of Q3, we considered whether the improved LLM performance on graph problems is due to the ordered graph descriptions containing all or part of the ground truth. For example, in the shortest path problem, the BFS and DFS edge lists may partially overlap with the correct shortest path.

To validate this hypothesis, We designed two extreme orders for shortest path task:

- **Shortest Path Order**: Edges are ordered based on the shortest path from the root node v_0 to the target node v_t.
- **Longest Path Order**: Edges are ordered according to the longest path from v_0 to v_t.

We test the two orders on a subset of GraphDO, and the results are shown in Figure 6. The shortest path order, which has the highest overlap with the answer, achieves 78.57% accuracy with the CoT prompt, a 16.4% improvement over BFS. In contrast, the accuracy of the longest path order is nearly identical to the random order. Although the shortest path order shows significant improvement, it still falls significantly short of 100%, indicating that while overlap with the answer has some influence, it is not the sole factor. This confirms that ordered graph descriptions can indeed enhance LLMs' understanding of graphs.

Model Comparison Study We repeat a subset of the experiments from Table 1 on five open-source LLMs to test the generalizability of our findings. As demonstrated in Figure 7, similar patterns emerged, consistent with previous results, though the effects were less pronounced than with GPT-3.5-TURBO-0613. We attribute this to the relatively weaker reasoning abilities of these open-source models and their limited capacity to map textual graph descriptions to conceptual spaces (Patel et al., 2021). Consequently, graph order has a smaller impact on these models compared to GPT-3.5-TURBO-0613. Additionally, some models demonstrate superior performance in specific tasks. For example, although QWEN2-7B does not excel in other tasks, it shows outstanding performance in the connectivity task, even surpassing LLAMA2-13B-CHAT with larger capacity.



(a) Connectivity (b) Cycle (c) Shortest Path

(d) Hamilton Path (e) Topological Sort

Figure 7: The impact of model differences on solving graph reasoning problems.

6 Related Work

LLMs Reasoning LLMs' reasoning and common-sense skills are applied to decision-making and action tasks in various domains. Yao et al. (2022) proposed ReAct, a prompting method that synergizes thinking and action, making it particularly effective for more complex problems involving autonomous planning and exploration. Madaan et al. (2023) introduced Self-Refine, a feedback-driven iterative refinement approach to rectify the mistakes and hallucinations during the inference time of LLM reasoning. Shinn et al. (2023) proposed Reflexion, which uses linguistic feedback stored in episodic memory to enhance decision-making in language agents. Sun et al. (2023) presented AdaPlanner, a closed-loop approach that enables LLM agents to refine their plans adaptively in response to environmental feedback, integrating in-plan and out-of-plan strategies to improve sequential decision-making. More recently, Zhou et al. (2023) introduced Language Agent Tree Search (LATS), a framework that combines Monte Carlo Tree Search with LM-powered value functions and self-reflections, enabling more deliberate and adaptive decision by integrating reasoning, acting, and planning.

Graph Reasoning with LLMs Wang et al. (2023a) introduced NLGraph, a benchmark of graph problems in natural language, and proposed Build-a-Graph and Algorithmic prompting to improve LLM performance. Fatemi et al. (2023) conducted the first comprehensive study on encoding graph-structured data as text for LLMs, revealing that task performance depends on encoding methods, graph tasks, and graph structure. Zhao et al. (2023) presented graphtext, a framework that converts graphs into natural language using a graph-syntax tree, enabling facilitating interactive communication between humans and LLMs. Wei et al. (2024) proposed GITA, an end-to-end framework that integrates visual graphs into general graph reasoning. Das et al. (2023) encode a graph with diverse modalities to enhance LLM efficiency in processing complex graph structures.

7 Conclusion

In this work, we conduct the first comprehensive analysis of how graph description order affects LLM performance in solving graph problems. Our findings demonstrate that ordered graph descriptions significantly enhance LLMs' ability to comprehend and reason about graph structures, a crucial discovery that could reshape the way we approach graph reasoning tasks. Additionally, through detailed analysis of various graph description orders, we observe that the impact of order on performance is closely tied to the intrinsic characteristics of each task. We believe that the over-reliance on graph descriptions stems from limitations in positional encoding—what we refer to as attention bias. Lastly, we int roduce the GraphDO dataset, which aims to advance the community's understanding of how graph descriptions influence reasoning in LLMs, providing a valuable benchmark for future research in this area.

Limitations

While our work demonstrates the critical role that the order of graph descriptions plays in LLMs' understanding of graphs, we have not explored the impact of this order on different graph structures and types in greater depth. Additionally, although we conducted some in-depth analyses, we did not provide a rigorous mathematical and theoretical explanation for the phenomena observed in this paper, which warrants further experimental investigation.

Ethics Statement

In conducting our research, we place paramount importance on ethical standards to ensure integrity and contribute positively to the scientific community. We exclusively utilize open-source datasets, ensuring that our work is built upon accessible and transparent resources. Our methods employ models that are either open-source or have gained wide recognition for their reliability and ethical use within the academic community. Furthermore, we have meticulously designed our methodology to prevent the generation of harmful or misleading information, thereby safeguarding the integrity of our findings.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Baolong Bi, Shenghua Liu, Lingrui Mei, Yiwei Wang, Pengliang Ji, and Xueqi Cheng. 2024a. Decoding by contrasting knowledge: Enhancing llms' confidence on edited facts. *arXiv preprint arXiv:2405.11613*.

Baolong Bi, Shenghua Liu, Yiwei Wang, Lingrui Mei, and Xueqi Cheng. 2024b. Lpnl: Scalable link prediction with large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3615–3625.

Baolong Bi, Shenghua Liu, Yiwei Wang, Lingrui Mei, Hongcheng Gao, Junfeng Fang, and Xueqi Cheng. 2024c. Struedit: Structured outputs enable the fast and accurate knowledge editing for large language models. *arXiv preprint arXiv:2409.10132*.

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. *Advances in neural information processing systems*, 33:1877–1901.

Yukun Cao, Shuo Han, Zengyi Gao, Zezhong Ding, Xike Xie, and S. Kevin Zhou. 2024. Graphinsight: Unlocking insights in large language models for graph structure understanding. *Preprint*, arXiv:2409.03258.

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. 2023. Graphllm: Boosting graph reasoning ability of large language model. *arXiv preprint arXiv:2310.05845*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167.

Maxwell Crouse, Pavan Kapanipathi, Subhajit Chaudhury, Tahira Naseem, Ramon Astudillo, Achille Fokoue, and Tim Klinger. 2023. Laziness is a virtue when it comes to compositionality in neural semantic parsing. *arXiv preprint arXiv:2305.04346*.

Debarati Das, Ishaan Gupta, Jaideep Srivastava, and Dongyeop Kang. 2023. Which modality should i use–text, motif, or image?: Understanding graphs with large language models. *arXiv preprint arXiv:2311.09862*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*.

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

Yunfan Gao, Tao Sheng, Youlin Xiang, Yun Xiong, Haofen Wang, and Jiawei Zhang. 2023. Chatrec: Towards interactive and explainable llms-augmented recommender system. *arXiv preprint arXiv:2303.14524*.

Yuyao Ge, Zhongguo Yang, Lizhe Chen, Yiming Wang, and Chengyang Li. 2023. Attack based on data: a novel perspective to attack sensitive points directly. *Cybersecurity*, 6(1):43.

C Lee Giles, Kurt D Bollacker, and Steve Lawrence. 1998. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98.

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

William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. 2019. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*.

Aric Hagberg, Pieter J Swart, and Daniel A Schult. 2008. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States).

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. 2023. Large language models on graphs: A comprehensive survey. *arXiv preprint arXiv:2312.02783*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163.

Lingrui Mei, Shenghua Liu, Yiwei Wang, Baolong Bi, and Xueqi Chen. 2024a. Slang: New concept comprehension of large language models. *arXiv preprint arXiv:2401.12585*.

Lingrui Mei, Shenghua Liu, Yiwei Wang, Baolong Bi, Jiayi Mao, and Xueqi Cheng. 2024b. " not aligned" is not" malicious": Being careful about hallucinations of large language models' jailbreak. *arXiv preprint arXiv:2406.11668*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Muhammad Ferjad Naeem, Muhammad Gul Zain Ali Khan, Yongqin Xian, Muhammad Zeshan Afzal, Didier Stricker, Luc Van Gool, and Federico Tombari. 2023. I2mvformer: Large language model generated multi-view document supervision for zero-shot image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15169–15179.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. 2024. The prompt report: A systematic survey of prompting techniques. *Preprint*, arXiv:2406.06608.

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

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Konstantinos Skianis, Giannis Nikolentzos, and Michalis Vazirgiannis. 2024. Graph reasoning with large language models via pseudo-code prompting. *Preprint*, arXiv:2409.17906.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Ruixiang Tang, Dehan Kong, Longtao Huang, and Hui Xue. 2023. Large language models can be lazy learners: Analyze shortcuts in in-context learning. *arXiv preprint arXiv:2305.17256*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023a. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2023a. Can language models solve graph problems in natural language? *arXiv preprint arXiv:2305.10037*.

Yiwei Wang, Yujun Cai, Muhao Chen, Yuxuan Liang, and Bryan Hooi. 2023b. Primacy effect of chatgpt. *arXiv preprint arXiv:2310.13206*.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023c. Describe, explain, plan and select: interactive planning with llms enables open-world multi-task agents. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Yanbin Wei, Shuai Fu, Weisen Jiang, Zejian Zhang, Zhixiong Zeng, Qi Wu, James T. Kwok, and Yu Zhang. 2024. Gita: Graph to visual and textual integration for vision-language graph reasoning. *Preprint*, arXiv:2402.02130.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Guangzi Zhang, Lizhe Chen, Yu Zhang, Yan Liu, Yuyao Ge, and Xingquan Cai. 2024. Translating words to worlds: Zero-shot synthesis of 3d terrain from textual descriptions using large language models. *Applied Sciences*, 14(8):3257.

Zeyang Zhang, Xin Wang, Ziwei Zhang, Haoyang Li, Yijian Qin, Simin Wu, and Wenwu Zhu. 2023. Llm4dyg: Can large language models solve problems on dynamic graphs? *arXiv preprint arXiv:2310.17110*.

Jianan Zhao, Le Zhuo, Yikang Shen, Meng Qu, Kai Liu, Michael Bronstein, Zhaocheng Zhu, and Jian Tang. 2023. Graphtext: Graph reasoning in text space. *Preprint*, arXiv:2310.01089.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

A Additional experimental details

A.1 Prompt Design

We employe various prompting methods in our experiments. We briefly introduce these methods here:

- **Zero-shot**: Zero-shot prompting only provides the task description and uses zero exemplars, requiring the model to generate the desired output.
- **Zero-shot CoT** (Wei et al., 2022): Zero-shot CoT prompting involves appending a thought inducing phrase "Let's think step by step."
- **Few-shot**(Brown et al., 2020) : Few-shot prompting provides the LLM with a few exemplars, including task descriptions and expected outputs, to guide its learning.
- **Chain-of-Thought (CoT)** (Wei et al., 2022): CoT prompting provides the model with a series of exemplars, each demonstrating a step-by-step approach to solving the task. It encourages the LLM to articulate its reasoning process before presenting the final answer.
- **CoT-BAG**(Wang et al., 2023a): Similar to CoT, but CoT-BAG prompting appends the phrase "Let's construct a graph with the nodes and edges first" at the end of the text to guide the model's reasoning process.

A more detailed explanation can be found in Table 3.

A.2 Personalization Vector

For the Personalized PageRank (PPR) order, the personalization vector e_v is defined in a task-specific manner. The definitions for various tasks are as follows:

T1 Connectivity For the two queried nodes u and v, the personalization vector is defined as:

$$e_u = e_v = 0.5 \quad \text{and} \quad e_w = 0 \quad \forall w \in \mathcal{V} \setminus \{u, v\}.$$

T2 Cycle If a cycle $\mathcal{C}$ exists, the personalization vector e_v is uniformly distributed across the nodes forming the cycle, with the sum equal to 1:

$$e_v = \frac{1}{|\mathcal{C}|} \quad \forall v \in \mathcal{C}, \quad e_w = 0 \quad \forall w \in \mathcal{V} \setminus \mathcal{C}.$$

If no cycle exists, e_v is uniformly distributed across all nodes:

$$e_v = \frac{1}{|\mathcal{V}|} \quad \forall v \in \mathcal{V}.$$

T3 Hamiltonian Path For the nodes along the Hamiltonian path $\mathcal{H}$, the personalization vector is uniformly distributed, with the sum equal to 1:

$$e_v = \frac{1}{|\mathcal{H}|} \quad \forall v \in \mathcal{H}, \quad e_w = 0 \quad \forall w \in \mathcal{V} \setminus \mathcal{H}.$$

Here, $\mathcal{H}$ represents the set of nodes on the Hamiltonian path.

T4 Shortest Path For the nodes along the shortest path $\mathcal{P}_{uv}$ between two nodes u and v, the personalization vector is uniformly distributed, with the sum equal to 1:

$$e_v = \frac{1}{|\mathcal{P}_{uv}|} \quad \forall v \in \mathcal{P}_{uv}, \quad e_w = 0 \quad \forall w \in \mathcal{V} \setminus \mathcal{P}_{uv}.$$

Here, $\mathcal{P}_{uv}$ represents the set of nodes on the shortest path between nodes u and v.

T5 Topological Sort For the nodes with in-degree 0 in a directed acyclic graph, the personalization vector is uniformly distributed, with the sum equal to 1:

$$e_v = \frac{1}{|\mathcal{V}_0|} \quad \forall v \in \mathcal{V}_0, \quad e_w = 0 \quad \forall w \in \mathcal{V} \setminus \mathcal{V}_0,$$

where $\mathcal{V}_0$ represents the set of nodes with in-degree 0.

T6 Node Classification The personalization vector is defined based on the shortest path distance $\delta(v)$ from the target node v_0 to each node v, with the formula:

$$e_v = \frac{\Delta - \delta(v) + 1}{\sum_{u \in \mathcal{V}} (\Delta - \delta(u) + 1)},$$

where $\delta(v)$ is the shortest path distance from node v_0 to node v, and Δ represents the maximum shortest path distance from v_0 to any node in the graph.

A.3 Response Parser

For GPT series models, we utilize string matching to parse responses. In binary classification tasks, such as cycle detection and connectivity detection, we extract answers by matching specific keywords like "there is a cycle" or "there is no cycle" to assess correctness. Path-related tasks are more complex: we first locate the approximate position of the path by matching keywords such as "the shortest path from x to x," then use numerical matching to extract the node indices and evaluate whether the path satisfies the task requirements.

Prompt Style	Prompt Template
Zero-shot	Graph: <Graph Description> \n Question: <Question> \n Answer:
Zero-shot CoT	Graph: <Graph Description> \n Question: <Question> Let's think step by step. \n Answer:
Few-shot	Graph: <Example Graph Description> \n Question: <Example question> Answer: <Example Answer> \n ... (more few-shot examples) \n Graph: <Graph Description> \n Question: <Question> \n Answer:
CoT	Graph: textcolorblack<Example Graph Description> \n Question: <Example Question> Answer: <Example Answer with CoT> \n ... (more examples with CoT) \n Graph: <Graph Description> \n Question: <Question> \n Answer:
CoT-BAG	Graph: <Example Graph Description> \n Question: <Example Question> Answer: <Answer with CoT> \n ... (more examples with CoT) \n Graph: <Graph Description> \n Question: <Question> \n Let's construct a graph with the nodes and edges first \n Answer:

Table 3: Prompt styles and their corresponding templates for graph reasoning tasks.

However, for open-source LLMs, the weaker in-context learning abilities of smaller models compared to GPT make it difficult for them to replicate the provided example responses, complicating answer extraction through string matching. Additionally, differences in training data and methods often result in distinct response styles across LLMs, further hindering the use of parsers. Designing a custom parser for each LLM would significantly increase the workload. Therefore, we employe GPT to verify whether the responses of open-source LLMs align with the correct answers.

B GraphDO

B.1 Graph Generation

For traditional graph tasks, we employe the Erdős-Rényi (ER) graph generation method. Specifically, we set the number of nodes n and a connection probability p, where any two nodes are connected with probability p. Edges can be directed or undirected based on the task. In our experiments, we chose n between 5 and 15, as previous studies have shown that LLMs demonstrate more consistent reasoning abilities on graphs of this size, making them suitable for detecting patterns in LLM performance (Wang et al., 2023a; Cao et al., 2024). The connection probability p was fixed at 0.3 to ensure that the graphs had a moderate level of sparsity, which is crucial for evaluating the reasoning capabilities of LLMs without generating overly dense or trivial graph structures. Additionally, given the diversity of the tasks, we filtered the generated graphs to guarantee that each instance had a valid and well-defined solution. For example, in the shortest path problem, we ensure that a valid path always exists between the start and end nodes.

B.2 Sampling

For node classification task, given that the data is sourced from real-world datasets, graph sampling is necessary. Below, we outline the two sampling methods used.

Ego Starting from a selected central node, we expand the subgraph by including neighboring nodes within a fixed hop distance of 3, ensuring that the resulting subgraph contains 50 nodes.

Forest Fire Propagating from a selected seed node to neighboring nodes with a probability of 0.3 continues until 50 nodes are included or no further propagation is possible.

For the sampled subgraphs, the graphs are converted into text-based descriptions using various orders.

B.3 Statistics

Using the above methodology, we collecte a total of 1,700 graphs ($280 \times 5 + 50 \times 3 \times 2$). We encode the graphs in adjacency format and then sorted the edges according to different orders, converting them into text sequences in natural language. This resulted in 8,500 graph cases, with each case consisting of a graph description, a question, and an answer.

C Implementation Details

We deploy the open-source LLMs for our experiments on a 4 × A800 server. The decoding temperature was set to zero. We use the NetworkX library (Hagberg et al., 2008) to generate the random graphs and to find the answers to the graph problems. The node classification experiments presented in Table 2 were repeated three times due to the limited sample size and the prompt is set as zero-shot. When calculating PageRank score, the

default weight of edges in an unweighted graph is set to 1.

D Case Analysis

As shown in Table 4 and Table 5, although GPT answered most of the questions correctly, its imitation of the CoT provided in the prompt was incomplete, demonstrating limited in-context learning ability. In the erroneous examples presented in Table 5, GPT exhibited reasoning that appeared logically sound, but factual errors were made, which can be attributed to the phenomenon of hallucination in LLMs.

Question	Answer of gpt-3.5-0613
Graph: In an undirected graph, (i, j) means that node i and node j are connected with an edge, and the edges are: (0,1) (0,2) (1,5) (1,2) (1,3) (2,5). Question: Determine if there is a path between two nodes in the graph. Is there a path between node 2 and node 3? Answer: Node 2 is connected to node 1, node 1 is connected to node 3. We can follow the path: 2->1->3, so the answer is yes. Graph: In an undirected graph, (i, j) means that node i and node j are connected with an edge, and the edges are: (0,2) (0,5) (0,6) (1,3) (1,4) (2,5) (2,6) (3,4) (5,6). Question: Determine if there is a path between two nodes in the graph. Is there a path between node 5 and node 4? Answer: Node 5 is in the connected block consisted of node 0, node 2, node 5, and node 6. Node 4 is in the connected block consisting of node 1, node 3, and node 4. Node 5 and node 4 are not in the same connected block, so the answer is no. Graph: In an undirected graph, (i, j) means that node i and node j are connected with an edge, and the edges are: (4,0) (4,1) (4,2). Question: Determine if there is a path between two nodes in the graph. Is there a path between node 0 and node 2? Answer:	Node 0 is connected to node 4, and node 4 is connected to node 2. We can follow the path: 0->4->2, so the answer is yes. ✓
Graph: In an undirected graph, (i,j) means that node i and node j are connected with an edge, and the edges are: (3,4) (3,5) (1,0) (2,5) (2,0). Question: Is there a cycle in this graph? Answer: No, there is no cycle in this graph. Graph: In an undirected graph, (i,j) means that node i and node j are connected with an edge, and the edges are: (3,5) (1,0) (3,0) (3,4) (4,1) (2,3). Question: Is there a cycle in this graph? Answer: The edges (3,0), (1,0), (4,1), (3,4) form a cycle, so yes, there is a cycle in this graph. Graph: In an undirected graph, (i,j) means that node i and node j are connected with an edge, and the edges are: (1,3) (1,0) (4,0) (4,2). Question: Is there a cycle in this graph? Answer:	No, there is no cycle in this graph. The graph forms a tree-like structure without any closed loops. ✓
Graph: In an undirected graph, (i, j, w) means that nodes i and j are connected by an edge with weight w, and the edges are: (0, 1, 2) (1, 4, 1) (0, 2, 1) (2, 3, 1) (3, 4, 2). Question: Give the shortest path from node 0 to node 4. Answer: All the paths from node 0 to node 4 are: 0,1,4 with a total weight of 2 + 1 = 3, 0,2,3,4 with a total weight of 1 + 1 + 2 = 4. The weight of path 0,1,4 is the smallest, so the shortest path from node 0 to node 4 is 0,1,4 with a total weight of 3. Graph: In an undirected graph, (i, j, w) means that nodes i and j are connected by an edge with weight w, and the edges are: (0, 3, 2) (0, 4, 1) (0, 2, 1) (4, 1, 2) (2, 1, 1) (3, 2, 4) (2, 4, 1) (3, 4, 2). Question: Give the shortest path from node 3 to node 1. Answer: All the paths from node 3 to node 1 are: 3,2,1 with a total weight of 4 + 1 = 5, 3,2,4,1 with a total weight of 4 + 1 + 2 = 7, 3,4,1 with a total weight of 2 + 2 = 4, 3,4,2,1 with a total weight of 2 + 1 + 1 = 4, 3,0,4,1 with a total weight of 2 + 1 + 2 = 5, 3,0,2,1 with a total weight of 2 + 1 + 1 = 4, 3,4,2,4,1 with a total weight of 2 + 1 + 1 + 2 = 6. The weight of path 3,4,1 is the smallest, so the shortest path from node 3 to node 1 is 3,4,1 with a total weight of 4. Graph: In an undirected graph, (i, j, w) means that nodes i and j are connected by an edge with weight w, and the edges are: (0, 4, 4) (0, 3, 3) (0, 1, 3) (1, 4, 4) (2, 4, 2) (2, 3, 1) (3, 4, 1). Question: Give the shortest path from node 0 to node 2.	To find the shortest path from node 0 to node 2 in the given graph, we can use Dijkstra's algorithm. Let's calculate the shortest path step by step: * Start at node 0. * From node 0, we can go to node 1 with weight 3 and to node 3 with weight 3. * From node 1, we can go to node 4 with weight 4. * From node 3, we can go to node 2 with weight 1. Therefore, the shortest path from node 0 to node 2 is $0 \rightarrow 3 \rightarrow 2$ with a total weight of 3 + 1 = 4. ✓

Table 4: Cases of the questions and the responses of GPT-3.5-TURBO-0613.

Question	Answer of gpt-3.5-0613
Graph: In a directed graph, (i, j) means that node i and node j are connected with an edge, and the edges are: (0, 4), (0, 1), (2, 1), (3, 2). Question: Give any topological sorting of the graph. Answer: First, I need to find the nodes with an in-degree of 0: 0,3. Then, I will traverse these nodes in sequence: starting from node 0, I traverse nodes 4, 1. Since node 4 and node 1 depend on node 0, after node 0 is visited, nodes 4 and 1 can be visited. Starting from node 3, I traverse node 2. Since node 2 depends on node 3, after node 3 is visited, node 2 can be visited. So the topological sequence of the graph is 0, 3, 4, 2, 1. Graph: In a directed graph, (i, j) means that node i and node j are connected with an edge, and the edges are: (0, 1), (1, 2), (3, 1), (3, 2), (4, 3), (4, 0), (4, 1). Question: Give any topological sorting of the graph. Answer: First, I need to find the nodes with an in-degree of 0: 4. Then, I will traverse these nodes in sequence: starting from node 4, I traverse nodes 0, 1, 3. Since node 0 and node 3 both depend on node 4, after node 4 is visited, nodes 0 and 3 can be visited. Since visiting node 1 requires visiting nodes 0 and 3 first, after node 0 and node 3 are visited, node 1 can be visited. Since visiting node 2 requires visiting nodes 1 and 3 first, after node 1 and node 3 are visited, node 2 can be visited. So the topological sequence of the graph is 4, 0, 3, 1, 2. Graph: In a directed graph, (i, j) means that node i and node j are connected with an edge, and the edges are: (0, 3), (1, 2), (3, 2), (4, 1), (4, 2), (5, 1), (5, 2). Question: Give any topological sorting of the graph. Answer:	To find a topological sorting of the given directed graph, we can follow the steps of Kahn's algorithm. Here's how we can do this for the provided graph: * Find nodes with in-degree 0. * Start with these nodes and remove their outgoing edges. * Repeat until all nodes are visited. * For the given graph with edges: (0, 3), (1, 2), (3, 2), (4, 1), (4, 2), (5, 1), (5, 2), the topological sorting could be as follows: 0, 4, 5, 1, 3, 2 This sequence satisfies the topological order where each node appears after its dependencies. ✓
Graph: In an undirected graph, (i, j) means that node i and node j are connected with an edge, and the edges are: (4, 2), (0, 4), (4, 3), (0, 1), (0, 2), (4, 1), (2, 3). Question: Is there a path in this graph that visits every node exactly once? If yes, give the path. Note that in a path, adjacent nodes must be connected with edges. Answer: Yes. We can start at node 0. As node 0 is connected with node 1, and node 1 is not visited, we can then visit node 1. As node 1 is connected with node 4, and node 4 is not visited, we can then visit node 4. As node 4 is connected with node 2, and node 2 is not visited, we can then visit node 2. As node 2 is connected with node 3, and node 3 is not visited, we can then visit node 3. Therefore, the path can be: 0, 1, 4, 2, 3. Graph: In an undirected graph, (i, j) means that node i and node j are connected with an edge, and the edges are: (0, 4), (1, 5), (3, 1), (4, 2), (3, 5), (2, 1), (1, 4), (2, 5). Question: Is there a path in this graph that visits every node exactly once? If yes, give the path. Note that in a path, adjacent nodes must be connected with edges. Answer: Yes. We can start at node 1. As node 1 is connected with node 3, and node 3 is not visited, we can then visit node 3. As node 3 is connected with node 5, and node 5 is not visited, we can then visit node 5. As node 5 is connected with node 2, and node 2 is not visited, we can then visit node 2. As node 2 is connected with node 4, and node 4 is not visited, we can then visit node 4. As node 4 is connected with node 0, and node 0 is not visited, we can then visit node 0. The path can be: 1, 3, 5, 2, 4, 0. Graph: In an undirected graph, (i, j) means that node i and node j are connected with an edge, and the edges are: (0, 2), (0, 4), (0, 1), (1, 3), (1, 2), (3, 4). Question: Is there a path in this graph that visits every node exactly once? If yes, give the path. Note that in a path, adjacent nodes must be connected with edges. Answer:	Yes, there is a path that visits every node exactly once in this graph. We can start at node 0. Then we can visit node 1, as it is connected to node 0. Next, we move to node 3 since it is connected to node 1. After that, we visit node 4 which is connected to node 3. Finally, we can move to node 2 from node 4, completing the path. Therefore, the path that visits every node exactly once is: 0, 1, 3, 4, 2. ✗

Table 5: Cases of the questions and the responses of GPT-3.5-TURBO-0613.