Graph Reasoning with Large Language Models via Pseudo-code Prompting

Anonymous ACL submission

Abstract

Large language models (LLMs) have recently achieved remarkable success in various reasoning tasks in the field of natural language processing. This success of LLMs has also motivated their use in graph-related tasks. Among others, recent work has explored whether LLMs can solve graph problems such as counting the number of connected components of a graph or computing the shortest path distance between two nodes. Although LLMs possess preliminary graph reasoning abilities, they might still struggle to solve some seemingly simple problems. In this paper, we investigate whether prompting via pseudo-code instructions can improve the performance of LLMs in solving graph problems. This approach not only aligns the model's reasoning with algorithmic logic but also imposes a structured, modular approach to problem-solving that is inherently transparent and interpretable. Our experiments demonstrate that using pseudo-code instructions generally improves the performance of all considered LLMs. The graphs, pseudo-code prompts, and evaluation code are publicly available[1].

1 Introduction

Recently, the artificial intelligence community has witnessed great advancements in the field of large language models (LLMs) (Devlin et al., 2019; Brown et al., 2020; Ouyang et al., 2022). Those models have captured intense public and academic interest, while the success of LLMs in different domains such as in medicine (Thirunavukarasu et al., 2023) and in software engineering (Poesia et al., 2022) has boosted hopes that these models could potentially pave the way for the development of Artificial General Intelligence (Bubeck et al., 2023). These advancements have been made possible not only due to breakthroughs in the field of machine learning, such as the introduction of the Transformer (Vaswani et al., 2017), but also due to the availability of massive amounts of data and the increase of computational power.

While LLMs were originally designed for textual data, they have already been utilized in settings that go beyond their initial application context. In several of those settings, a graph structure is explicitly or implicitly involved. For example, in world modeling, LLMs are commonly employed to generate knowledge graphs for text games in order to improve an agent's ability to efficiently operate in complex environments (Ammanabrolu and Riedl, 2021). However, LLMs rely on unstructured text, and in those settings, they might fail to properly encode the different entities and their relationships. This might lead to different issues, e.g. the models might fail to deduce some logical entailments or they might hallucinate, i.e. generate plausible-sounding responses that are factually incorrect.

Despite the preliminary success of LLMs in the aforementioned settings, it is still not entirely clear whether those models exhibit fundamental limitations that might constrain their applicability in those domains. Some recent studies shed some light on this issue by investigating whether LLMs can actually reason with graphs (Wang et al., 2023a; Fatemi et al., 2024). In fact, those studies investigated whether LLMs can solve graph problems fed to them as natural language prompts. The two studies employed different LLMs, and the reported results are somewhat ambivalent. While in one study, it was shown that LLMs possess preliminary graph reasoning abilities (Wang et al., 2023a), in the other study, LLMs failed to solve basic graph tasks (*e.g.,* count the number of edges of a graph).

In this paper, we study whether prompt engineering can help us improve the performance of LLMs in solving graph algorithm problems. Natural language instructions can be ambiguous and underspecified, and this might prevent models from re-

[1] `https://anonymous.4open.science/r/graph-reasoning-llms-7D70`

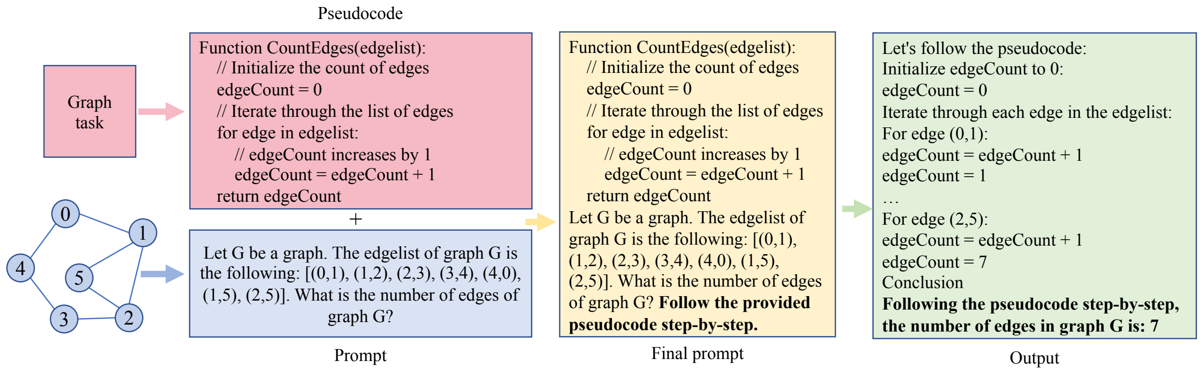

Figure 1: An illustration of our proposed method for graph reasoning using pseudo-code instructions.

turning the most accurate answer possible. Furthermore, very detailed instructions might increase the complexity of reasoning and harm the model's performance. Therefore, prompt engineering can significantly contribute to enhancing the capabilities of pre-trained LLMs. (Liu et al., 2023). Different prompting strategies have been developed so far. The idea to use prompts that encourage multi-step reasoning led to very successful methods such as the chain-of-thought (CoT) reasoning in few-shot settings (Wei et al., 2022), while it was also shown that LLMs can become decent zero-shot reasoners by just adding the prompt "Let's think step by step" (Kojima et al., 2022). Here, we investigate whether the use of pseudo-code instructions for prompting can enhance the performance of LLMs in solving graph algorithm problems. Pseudo-code can reduce the ambiguity present in natural language, but it also provides explicit and clear instructions on how to solve a problem. An example of the proposed approach for prompting with pseudo-code is illustrated in Figure 1. We study performance in 10 graph reasoning tasks on two LLM families (GPT and Mixtral). The obtained results indicate that the proposed method improves the performance of LLM mainly in tasks that they struggle to solve.

In summary, our paper makes the following contributions:

- We release a new benchmark dataset of pseudo-code prompts for different graph problems to test the reasoning abilities of LLMs.
- We study the impact of these prompts on the performance of three LLMs in 10 graph reasoning tasks.
- The experimental results demonstrate that augmenting prompts with pseudo-code can be useful for solving both simple, but also complex graph reasoning tasks.

2 Related Work

Large Language Models and graphs. Graph neural networks (GNNs) have been established as the standard neural architecture for performing machine learning on graphs since these models are invariant to permutations of the nodes of the input graph (Zhou et al., 2020). Other common architectures, such as the family of recurrent neural networks, do not enjoy this property. However, permutation-sensitive models such as the Transformer architecture (Vaswani et al., 2017) can also deal with graph learning problems. For example, it was shown in (Kim et al., 2022) that if we treat both nodes and edges as independent tokens, augment them with token embeddings, and feed them to a Transformer, we obtain a powerful graph learner. Some node classification datasets where the nodes are annotated with textual content have been treated as text classification datasets by ignoring the graph structure, and LLMs have been leveraged to classify the textual content (Chen et al., 2024). It was found that LLMs achieve good zero-shot performance on certain datasets. Similar conclusions were also reached by other works (Hu et al., 2023). Real-world data is noisy and this also applies to graphs. Thus, some works have leveraged LLMs to refine graphs (Sun et al., 2023; Guo et al., 2024). In the GraphEdit method, the LLM is responsible for identifying noisy connections between irrelevant nodes and for discovering implicit dependencies between nodes based on the textual data associated with them (Guo et al., 2024). Several works have investigated the potential of LLMs to enhance the performance of GNNs on text-attributed graphs (Duan et al., 2023; Chen et al., 2024; He et al., 2024). For instance, TAPE uses an LLM to extract predictions and explanations from the input text which serve as supplementary features for the downstream GNN model (He et al., 2024). The works closest to ours in this domain are the ones reported in (Wang et al., 2023a) and in (Fatemi et al., 2024), which investigate whether LLMs can solve graph algorithm problems in natural language. In this paper, we go one step further and study whether prompting with pseudo-code instructions can help LLMs better understand how to solve graph problems.

Not only LLMs have emerged as useful tools in graph learning tasks, but it turns out that the opposite is also true, *i.e.,* graphs can enhance LLMs (Pan et al., 2024). Even though LLMs have achieved great success in the past years, they still might suffer from different problems such as hallucinations, reduced factuality awareness, and limited explainability. Knowledge graphs can help LLMs deal with those issues since they store extensive high-quality and reliable factual knowledge. Therefore, to mitigate the aforementioned issues, knowledge graphs have been recently incorporated to improve the reasoning ability of LLMs (Guan et al., 2024; Luo et al., 2024).

Prompt engineering. Prompt engineering seeks for the best way to describe a task such that an LLM can solve the task using its autoregressive token-based mechanism for generating text. Prompt engineering is a resource-efficient approach in the sense that it does not require access to the internals of the model (*e.g.,* its parameters). We can thus provide the model with a task description and ask it to solve the task even if it has never been trained on it. Few-shot prompting aims to teach the language model how to solve a task by providing it with a small number of example tasks with solutions (Brown et al., 2020). The model then learns from these examples and can solve similar tasks. Chain-of-Thought (CoT) is a prompting technique, in which one includes a series of intermediate natural language reasoning steps that lead to the desired output (Wei et al., 2022). CoT was shown to significantly improve the capability of LLMs to solve problems. Zero-shot-CoT, another approach for prompting, simply adds the prompt "*Let's think step by step*" before each answer to facilitate step-by-step thinking (Kojima et al., 2022). Zero-shot-CoT turned out to be the strongest zero-shot baseline, while LLMs were shown to be decent zero-shot reasoners. However, Zero-shot-CoT might fail in some cases because of missing reasoning steps. Prompting via pseudo-code instructions has also been recently explored for solving natural language processing tasks (Mishra et al., 2023). Program-of-thoughts prompting generates code to solve a task (Chen et al., 2023). It uses Python code to describe reasoning steps, and the computation is accomplished by a Python interpreter. To improve the LLMs reasoning ability, some works have employed multiple rounds of prompting (Jung et al., 2022). For instance, least-to-most prompting teaches language models how to solve a complex problem by decomposing it into a series of simpler subproblems which are solved one after the other (Zhou et al., 2023). Self-Consistency is a scheme where multiple CoTs are generated and one of them is finally chosen (Wang et al., 2023b). Tree of Thoughts (ToT) (Yao et al., 2023) and Graph of Thoughts (GoT) (Besta et al., 2024) are two schemes that model the LLM reasoning process with a tree and a graph, respectively.

When LLMs are leveraged to solve graph tasks, different graph encoding schemes can be utilized to transform graph-structured data into text (*e.g.,* list of edges, adjacency matrix, graph description language, etc.). It was recently shown that input design indeed has a significant impact on the final result (Guo et al., 2023). GraphText constructs a graph-syntax tree from the input graph, and then, the traversal of the graph-syntax tree leads to a prompt in natural language which can be fed to the LLM to perform graph reasoning. (Zhao et al., 2023). More recently, continuous graph representations have been explored (Perozzi et al., 2024). The graph is mapped into a continuous vector via a GNN and this vector serves as input for the LLM.

3 Proposed Methodology

To investigate whether prompting with pseudo-code instructions can improve the capability of language models in reasoning with graphs, we focus on a wide range of graph tasks, we construct instances of those tasks and present them along with the pseudo-code that solves them as natural language queries to the language models. We next give more details about the different graph tasks we consider in this paper and how the different problem instances are generated.

Graph tasks. There exist many decision and optimization problems on graphs. Several of those problems are hard to solve (*e.g.,* finding a clique with the largest possible number of nodes is known

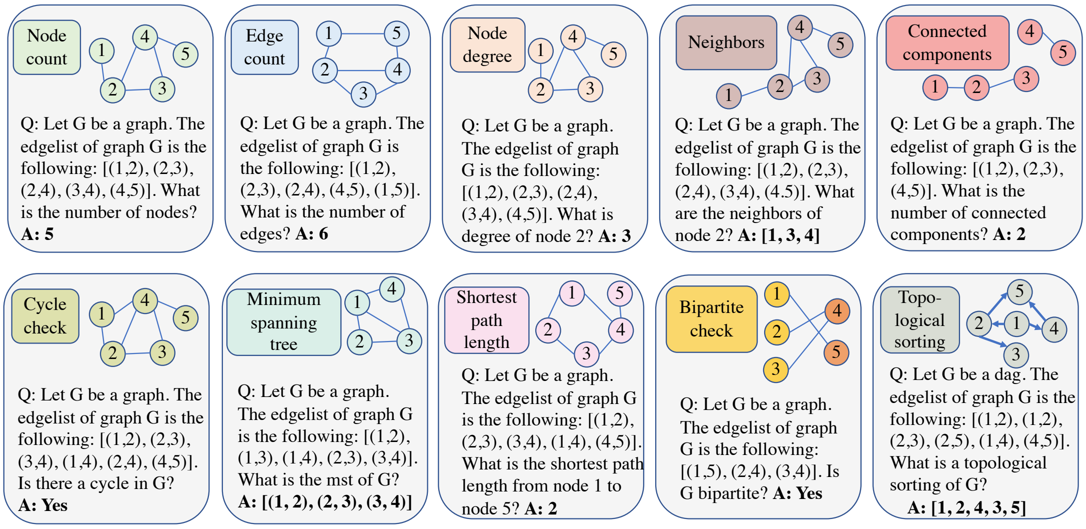

Figure 2: The proposed graph dataset.

to be an NP-hard problem). We cannot expect an LLM to be able to solve such problems in a short amount of time even when the input graphs are relatively small. Thus, here we focus on problems that can be solved in polynomial time in the worst case by some graph algorithm. We list below the 10 considered graph problems.

1. **Node count** - Count the number of nodes.
2. **Edge count** - Count the number of edges.
3. **Node degree** - Calculate the degree of a node.
4. **Neighbors** - Find all nodes that are adjacent to a given node.
5. **Connected components** - Count the number of connected components.
6. **Cycle check** - Check if a graph contains a cycle.
7. **Minimum spanning tree (MST)** - Find the minimum cardinality subset of edges of a given graph that connects all the vertices together, without any cycles.
8. **Shortest path** - Calculate the shortest path length between two nodes in a graph.
9. **Bipartite check** - Check if a graph is bipartite.
10. **Topological sorting** - Calculate a linear ordering of the nodes of a given directed acyclic graph such that for every directed edge (u, v) from node u to node v, u comes before v in the ordering.

Note that some tasks are easier, while others are more complex. For example, given the list of edges of the graph, *Edge count* requires just counting the number of elements of the list, while *Shortest path* is a more complex task since it generally requires further algorithmic steps to be performed to reach the solution.

Generated graphs. Even though the provided source code allows one to generate different types of graphs (*e.g.,* Erdős–Rényi graphs, Barabási–Albert graphs, star graphs, etc.), in this study, due to monetary costs, we focus mainly on Erdős–Rényi graphs. Therefore, for all tasks except *Bipartite check* and *Topological sorting*, the graphs about which the LLM is asked to reason are Erdős–Rényi graphs. To construct such a graph, we need to choose the number of nodes n and the edge probability p. As will be discussed later, we construct datasets of varying complexity, and the value of n depends on the type of the dataset. Hyperparameter p is sampled from $[0, 1]$ with uniform probability. For the *Topological sorting* task, we construct Erdős–Rényi graphs and we transform them into directed acyclic graphs. This is achieved by first mapping the nodes to integers, *i.e.,* $\{1, \ldots, n\}$, and then assigning direction to all edges such that they point from lower nodes to higher nodes. Finally, for *Bipartite check*, we either construct Erdős–Rényi graphs or random bipartite graphs. To construct a random bipartite graph, we create two sets of nodes such that no set is empty and such that the sum of their cardinalities is n, and then edges between nodes of one set and nodes of the other are included in the graph with probability p (where p is sampled from $[0, 1]$).

Generated problems. For each task, we construct three different datasets. The difference between those datasets lies in the number of nodes of the produced graphs. One dataset consists of small graphs, one consists of medium-sized graphs,

	Methods \ Tasks	Node count	Edge count	Node degree	Neighbors
S	0-SHOT	99	78	**75**	**90**
	1-SHOT	**100**	76	72	67
	BAG	67	57	73	78
	0-COT	82	67	70	77
	PSEUDO	87	**90**	56	75
	PSEUDO+1-SHOT	95	82	60	68
M	0-SHOT	88	16	24	42
	1-SHOT	**100**	22	28	29
	BAG	50	11	31	44
	0-COT	62	13	**46**	**51**
	PSEUDO	79	**34**	18	37
	PSEUDO+1-SHOT	63	18	43	30
L	0-SHOT	**100**	2	6	12
	1-SHOT	96	1	0	9
	BAG	72	0	7	**13**
	0-COT	7	2	**13**	12
	PSEUDO	62	**9**	6	**13**
	PSEUDO+1-SHOT	20	2	**13**	**13**

Table 1: Model GPT-3.5-Turbo-0125 results on simple tasks. Bold indicates best results.

and the last one consists of large graphs. We denote those three datasets by S, M, and L, respectively. The number of nodes of the graphs contained in those three datasets range between 5 and 11 nodes for S, 11 and 21 nodes for M, and 21 and 51 nodes for L. The different tasks do not share the same datasets of graphs. A different dataset is constructed for each task. Note that dataset L consists of graphs significantly larger than the ones considered in prior work (*i.e.,* all graphs had 5 and 35 nodes in (Wang et al., 2023a) and between 5 and 20 nodes in (Fatemi et al., 2024)). Our results thus also provide insights into the capabiblity of LLMs to perform reasoning tasks on *larger graphs* than the ones considered in previous studies. Once the graphs are generated, we create the prompts and we add to them pseudo-code instructions. We have created such instructions for all 10 considered tasks. An overview of the proposed graph reasoning tasks is shown in Figure 2.

Note that besides *Node degree*, *Neighbors* and *Shortest path*, the rest of the tasks correspond to graph-level properties. For each one of those seven tasks and for each graph size (*i.e.,* S, M or L), we construct 100 problems. This gives rise to $2,100$ problems in total. The *Node degree* and *Neighbors* tasks capture node-level properties of graphs. For those tasks and for each graph size, we create 100 graphs and from each one of those graphs, we randomly choose 5 nodes to create problems. This leads to $3,000$ more problems. Finally, the *Shortest path* task is defined between pairs of nodes. Once again, for each graph size, we create 100 graphs and from each one of those graphs, we randomly choose 5 pairs of nodes that both belong to the same connected component to create problems. This results into $1,500$ more problems. Overall, our dataset contains $6,600$ problems.

4 Experiments

Baselines. We compare the proposed method against the following three prompting approaches: (1) zero-shot prompting (0-SHOT); (2) one-shot in-context learning (1-SHOT) (Brown et al., 2020); (3) Build-a-Graph prompting (BAG) (Wang et al., 2023a); and (4) zero-shot chain-of-thought (0-COT) (Kojima et al., 2022). 0-SHOT constructs a prompt that describes the task and asks the LLM to solve the task, without any prior training on the task. Besides just a description of the task, 1-SHOT also provides the model with one example of the task, along with the desired output. BAG adds the sentence "Let's construct a graph with the nodes and edges first" to the task description. Last, 0-COT adds the sentence "Let's think step by step" to the task description to let the model generate its own Chain-of-Thoughts.

Models and Settings. We evaluate two popular LLMs, namely GPT-3.5-Turbo and Mixtral 7x8B, thus representing both proprietary and open source LLMs. For all baselines we set the parameter temperature = 0 in order to make results more deterministic and avoid randomness. As discussed above, we evaluate LLMs and various prompting techniques mainly on Erdős–Rényi graphs due to monetary costs, while we plan to evaluate the proposed method on other types of graphs in the future. We use two different variants of the proposed method. In the first variant (PSEUDO), we provide the LLM with the task description and the pseudocode to solve it, while in the second variant (PSEUDO + 1-SHOT), we also provide the model with one example of the task, along with the desired output. Previous works have found that graph encoding functions (*i.e.,* how to represent the graph in natural language) have a significant impact on the performance of LLMs in the different graph tasks (Guo et al., 2023; Fatemi et al., 2024). In this paper, we choose to represent each graph by its list of edges since it was shown that it outperforms other common representations (Guo et al., 2023).

Methods \ Tasks		Connected components	Cycle check	MST	Shortest path	Bipartite check	Topological sorting
S	0-SHOT	45	43	61	42	31	**88**
	1-SHOT	**86**	44	47	**73**	**61**	77
	BAG	4	23	19	18	48	**88**
	0-COT	30	47	16	25	51	62
	PSEUDO	76	76	61	50	52	72
	PSEUDO+1-SHOT	69	**79**	**64**	59	**61**	81
M	0-SHOT	57	7	23	15	**51**	**59**
	1-SHOT	**91**	46	6	**61**	**51**	33
	BAG	3	8	7	8	26	34
	0-COT	2	39	0	7	45	25
	PSEUDO	66	**47**	17	35	48	55
	PSEUDO+1-SHOT	47	**47**	**27**	51	42	36
L	0-SHOT	**85**	34	2	7	43	**28**
	1-SHOT	40	21	1	27	42	13
	BAG	2	1	4	14	17	8
	0-COT	0	6	0	2	48	6
	PSEUDO	49	**71**	10	22	**49**	14
	PSEUDO+1-SHOT	22	23	**27**	**34**	31	9

Table 2: Model GPT-3.5-Turbo-0125 results on the complex graph reasoning tasks. Results present accuracy in percentage (%). Bold indicates best results.

Evaluation metric. In all considered tasks, we are interested in finding whether the LLM provides the correct answer to a given query. We thus measure performance by computing the accuracy, *i.e.,* correct answers/total queries.

Performance on graph tasks. We first split the 10 different graph reasoning tasks into simpler tasks and more complex tasks. In the first part of our analysis, we focus on the simple tasks (*i.e., Node count*, *Edge count*, *Node degree* and *Neighbors*). We evaluate the different prompting approaches and initially employ GPT-3.5-Turbo-0125 as our LLM. Table 1 illustrates the results for these experiments. We observe that 0-SHOT and 1-SHOT prompting can accurately count the number of nodes of a graph even if the graph is large. Quite surprisingly, pseudo-code prompting fails to achieve similar levels of performance in this task. However, PSEUDO is the best-performing method in the *Edge count* task. In the *Node degree* and *Neighbors* tasks, no method outperforms consistently all the other methods. For small graphs, the LLM correctly answers more than half of the queries no matter the prompting technique. Besides the *Neighbors* task, 0-COT generally does not lead to improvements. As expected, the performance of the model decreases as the size of graphs increases. Overall, we observe that when the size of graphs is small, GPT3.5 performs quite well in the 4 simple reasoning tasks even when no examples or assistance is provided.

We next evaluate the GPT-3.5 model in the remaining 6 tasks (*i.e., Connected components*, *Cycle check*, *MST*, *Shortest path*, *Bipartite check* and *Topological sorting*). The results for these experiments are shown in Table 2. While one would expect the 0-SHOT approach to fail in all these tasks, we observe that it excels in *Topological sorting*. The example that the 1-SHOT method provides to the LLM seems to have a significant impact in some tasks, such as in identifying connected components and in computing shortest path distances. The 0-COT method is the worst-performing prompting technique, likely due to its inability to generate the actual reasoning steps needed to solve the problem. Incorporating pseudo-code into the prompt yields considerable improvements in some tasks, such as in computing shortest path lengths and in checking whether graph contain cycles where it provides the highest accuracy. The PSEUDO+1-SHOT approach is the best-performing prompting technique in the *MST* task and in computing shortest path lengths in large graphs. Surprisingly, in the *Connected components* and *Bipartite check* tasks, the size of the graphs does not seem to have any impact on the performance of the GPT-3.5 model.

We also experiment with the open-source Mixtral 7x8B model. The obtained results for the simpler tasks are shown in Figure 3. We observe

	Methods \ Tasks	Node count	Edge count	Node degree	Neighbors
S	0-SHOT	92	56	56	65
	1-SHOT	90	31	39	68
	BAG	92	51	64	60
	0-COT	88	42	70	**75**
	PSEUDO	89	83	63	63
	PSEUDO+1-SHOT	**97**	**99**	**73**	64
M	0-SHOT	89	8	23	27
	1-SHOT	88	9	7	31
	BAG	92	9	29	27
	0-COT	**93**	3	**34**	**37**
	PSEUDO	84	29	27	28
	PSEUDO+1-SHOT	81	**89**	31	27
L	0-SHOT	65	1	9	7
	1-SHOT	86	0	1	8
	BAG	**90**	0	**12**	7
	0-COT	83	2	11	**10**
	PSEUDO	80	7	7	7
	PSEUDO+1-SHOT	56	**14**	8	5

Table 3: Mixtral results on simple tasks. Results present accuracy in percentage (%). Bold indicates best results.

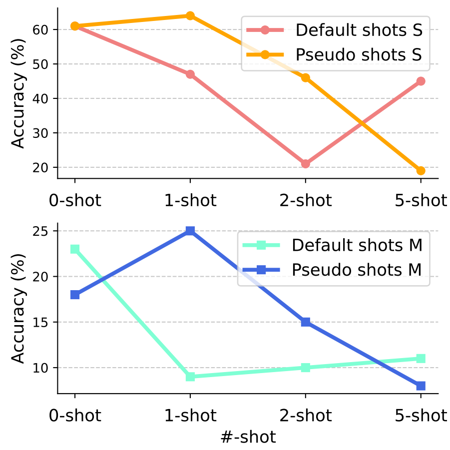

Figure 3: #-shot results in minimum spanning tree.

that no matter what prompting method we use, the model can always quite accurately count the number of nodes of the input graphs. However, in the rest of the tasks, 0-SHOT and 1-SHOT fail to achieve high levels of accuracy, especially for medium-sized and large graphs. In the *Edge count* task, these methods return a correct answer for less than 10% of the queries when the input graphs are not small. The results also suggest that 0-COT and BAG lead to performance improvements in most cases. Pseudo-code prompting also leads to significant performance gains in most cases. For example, PSEUDO+1-SHOT achieves the highest accuracy in the *Node count*, *Edge count*, and *Node degree* tasks, thus demonstrating how useful pseudo-code prompting is for less powerful LLMs. Specifically, in the *Edge count* and *Node degree* tasks and for small graphs, PSEUDO+1-SHOT led to a respective relative increase of 76.8% and 30.4% in accuracy over 0-SHOT. Furthermore, in the *Edge count* task and for medium-sized graphs, PSEUDO+1-SHOT resulted in an impressive relative increase of 1012.5% in accuracy. Finally, we should note that in most tasks, Mixtral's performance also decreases as the size of graphs increases.

The results for the more complex graph reasoning tasks are illustrated in Table 4. We observe that when pseudo-code is added to the prompt, it becomes harder for Mixtral to detect whether the input graph contains any cycle. However, the use of pseudo-code proves crucial for some other tasks such as *Connected components*. Interestingly, for small graphs, the PSEUDO+1-SHOT approach results in a relative increase of 114.3%, 75% and 16.7% in accuracy over 0-SHOT in the *Connected components*, *MST* and *Shortest path* tasks, respectively. Likewise, for medium-sized graphs, the use of pseudo-code use leads to a relative increase of 57.5% in accuracy over 0-SHOT in *Connected components*. These findings clearly indicate that augmenting the prompt with pseudo-code instructions and corresponding examples can significantly enhance accuracy in both simple and complex graph reasoning tasks.

Pseudo-code style. We next investigated what is the impact of the type of utilized pseudo-code on the performance of the LLM. Table 5 illustrates the results with different pseudo-code styles on Mixtral (1: Python, 2: Pseudo, 3: Complex). We observe that the results are mixed. Plain pseudo-code outperforms the rest in the *Neighbors* task, while Python code achieves the highest accuracy in the *MST* task. The pseudo-code that consists of multiple functions instead of a single one is the second-best method in both tasks. We should also mention that by examining the results, we observed that the LLM struggles when presented with nested loops and recursive functions.

	1	2	3
MST	**31**	24	29
Neighbors	40	**63**	46

Table 5: Results with different pseudo-code styles.

One vs. few examples. We also investigated whether we can obtain performance gains by increasing the number of examples provided to the

	Methods \ Tasks	Connected components	Cycle check	MST	Shortest path	Bipartite check	Topological sorting
S	0-SHOT	35	85	24	48	44	39
	1-SHOT	47	77	18	52	47	**57**
	BAG	**78**	82	19	28	39	51
	0-COT	70	**90**	27	55	**58**	35
	PSEUDO	62	33	24	50	47	40
	PSEUDO+1-SHOT	75	51	**42**	**56**	53	47
M	0-SHOT	40	**93**	6	30	42	6
	1-SHOT	31	75	7	**50**	50	11
	BAG	**65**	86	**8**	28	**53**	9
	0-COT	57	90	**8**	35	42	8
	PSEUDO	63	36	5	27	47	12
	PSEUDO+1-SHOT	42	40	5	40	48	**18**
L	0-SHOT	34	86	1	17	51	4
	1-SHOT	29	69	1	**25**	48	**11**
	BAG	25	77	1	10	45	2
	0-COT	27	**92**	1	15	**53**	2
	PSEUDO	**41**	31	1	13	44	5
	PSEUDO+1-SHOT	18	35	1	24	41	10

Table 4: Mixtral 8x7B results on the complex graph reasoning tasks. The results present accuracy in percentage (%). Bold indicates best results.

model. Figure 3 illustrates performance of GPT-3.5 in the small subset of the *MSE* task as a function of the number of examples. The results suggest that in case pseudo-code is present, a single example suffices. Unlike the 0-SHOT method, where adding more examples enhances the reasoning abilities of the LLM, our approach does not seem to benefit from multiple examples. Therefore, the proposed method appears to be more cost-efficient that other prompting techniques, as one example is enough to lead to performance improvements. Creating multiple examples, particularly in the context of graphs, can be time-consuming and resource-intensive.

Summary. We next present our main findings:

- **For most tasks, the size of the input graphs has a significant impact on the LLMs' performance.** With the exception of *Node count*, *Connected components* and *Bipartite check*, in all other tasks, performance decreases significantly as the size of the graphs increases.
- **LLMs can count nodes, but they cannot count edges.** While LLMs could quite accurately count the number of nodes of all graphs, no method achieved an accuracy greater than 14% in counting the number of edges of large graphs.
- **Pseudo-code is useful for tasks that LLMs struggle to solve.** Pseudo-code offered significant improvements in the *Edge count* and *MST* tasks, where the failure rate of LLMs is high.
- **There exist tasks where pseudo-code might improve the performance of one LLM, but lead another LLM to lower levels of performance.** PSEUDO significantly outperforms 0-SHOT in the *Cycle check* task when using GPT-3.5. On the other hand, PSEUDO is significantly outperformed by 0-SHOT in the same task with Mixtral.
- **Carefully designed prompting can improve the performance of LLMs.** In almost all our experiments, 0-SHOT was outperformed by the rest of the prompting techniques. This direction is computationally less demanding than fine-tuning pre-trained LLMs.
- **In the presence of pseudo-code, a single example is enough.** Even in complex graph reasoning tasks, prompting with pseudo-code does not need several examples to reach its full potential.

5 Conclusion

In this work, we explored whether prompting with pseudo-code instructions can enhance LLMs' reasoning on simple and complex graph tasks. Experiments with GPT-3.5 and Mixtral show that pseudo-code prompts improve performance across various graph tasks. However, performance declines as graph size increases. This highlights the need for further research on prompting techniques for large graphs. Our focus is on improving both reasoning and interpretability, showing LLMs can solve problems while making their reasoning steps explicit.

Limitations

Pseudo-code prompts need to be carefully designed or might not be available. To get the most out of pseudo-code, careful design is needed. Simple coding is preferred and complex structures such as nested loops or recursive functions should be avoided. We assume that pseudo-code is either directly available or there is access to the technical expertise required to write it.

Evaluation Automatically evaluating the performance of LLMs is by definition a hard task. In order to measure the performance, we search for the result in the LLM output. Therefore, some degree of ambiguity, variation in phrasing, and differences in reasoning approaches are inevitable. As a result, certain errors are expected when aligning the generated output with predefined answers or benchmarks."

References

Prithviraj Ammanabrolu and Mark Riedl. 2021. Learning Knowledge Graph-based World Models of Textual Environments. In *Advances in Neural Information Processing Systems*, pages 3720–3731.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*, pages 17682–17690.

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. In *Advances in Neural Information Processing Systems*, pages 1877–1901.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research*.

Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, et al. 2024. Exploring the Potential of Large Language Models (LLMs) in Learning on Graphs. *ACM SIGKDD Explorations Newsletter*, 25(2):42–61.

Jacob Devlin, Chang Ming-Wei, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Keyu Duan, Qian Liu, Tat-Seng Chua, Shuicheng Yan, Wei Tsang Ooi, Qizhe Xie, and Junxian He. 2023. SimTeG: A Frustratingly Simple Approach Improves Textual Graph Learning. *arXiv preprint arXiv:2308.02565*.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2024. Talk like a Graph: Encoding Graphs for Large Language Models. In *The 12th International Conference on Learning Representations*.

Xinyan Guan, Yanjiang Liu, Hongyu Lin, Yaojie Lu, Ben He, Xianpei Han, and Le Sun. 2024. Mitigating Large Language Model Hallucinations via Autonomous Knowledge Graph-Based Retrofitting. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*, pages 18126–18134.

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*.

Zirui Guo, Lianghao Xia, Yanhua Yu, Yuling Wang, Zixuan Yang, Wei Wei, Liang Pang, Tat-Seng Chua, and Chao Huang. 2024. GraphEdit: Large Language Models for Graph Structure Learning. *arXiv preprint arXiv:2402.15183*.

Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. 2024. Harnessing Explanations: LLM-to-LM Interpreter for Enhanced Text-Attributed Graph Representation Learning. In *The 12th International Conference on Learning Representations*.

Yuntong Hu, Zheng Zhang, and Liang Zhao. 2023. Beyond Text: A Deep Dive into Large Language Models' Ability on Understanding Graph Data. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*.

Jaehun Jung, Lianhui Qin, Sean Welleck, Faeze Brahman, Chandra Bhagavatula, Ronan Le Bras, and Yejin Choi. 2022. Maieutic Prompting: Logically Consistent Reasoning with Recursive Explanations. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1266–1279.

Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. 2022. Pure Transformers are Powerful Graph Learners. In *Advances in Neural Information Processing Systems*, pages 14582–14595.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. pages 22199–22213.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Computing Surveys*, 55(9):1–35.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024. Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning. In *The 12th International Conference on Learning Representations*.

Mayank Mishra, Prince Kumar, Riyaz Bhat, Rudra Murthy, Danish Contractor, and Srikanth Tamilselvam. 2023. Prompting with Pseudo-Code Instructions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15178–15197.

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. In *Advances in Neural Information Processing Systems*, pages 27730–27744.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering*.

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*.

Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Synchromesh: Reliable Code Generation from Pre-trained Language Models. In *The 10th International Conference on Learning Representations*.

Shengyin Sun, Yuxiang Ren, Chen Ma, and Xuecang Zhang. 2023. Large Language Models as Topological Structure Enhancers for Text-Attributed Graphs. *arXiv preprint arXiv:2311.14324*.

Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. 2023. Large language models in medicine. *Nature medicine*, 29(8):1930–1940.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2023a. Can Language Models Solve Graph Problems in Natural Language? In *Advances in Neural Information Processing Systems*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The 11th International Conference on Learning Representations*.

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. In *Advances in Neural Information Processing Systems*, pages 24824–24837.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems*.

Jianan Zhao, Le Zhuo, Yikang Shen, Meng Qu, Kai Liu, Michael Bronstein, Zhaocheng Zhu, and Jian Tang. 2023. GraphText: Graph Reasoning in Text Space. *arXiv preprint arXiv:2310.01089*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, et al. 2023. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *The 11th International Conference on Learning Representations*.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81.