Actions Speak Louder than Prompts: A Large-Scale Study of LLMs for Graph Inference

Ben Finkelshtein * University of Oxford
Silviu Cucerzan Microsoft Research
Sujay Kumar Jauhar Microsoft Research
Ryen White Microsoft Research

Abstract

Large language models (LLMs) are increasingly leveraged for text-rich graph machine learning tasks, with node classification standing out due to its high-impact application domains such as fraud detection and recommendation systems. Yet, despite a surge of interest, the field lacks a principled understanding of the capabilities of LLMs in processing graph data. In this work, we conduct a large-scale, controlled evaluation across the key axes of variability: the LLM-graph interaction mode, comparing prompting, tool-use, and code generation; dataset domains, spanning citation, web-link, e-commerce, and social networks; homophilic vs. heterophilic regimes; and short- vs. long-text features. We further analyze dependencies by independently truncating features, deleting edges, and removing labels to quantify reliance on input types. Our findings provide actionable guidance for both research and practice. (1) Code generation mode achieves the strongest overall performance, with especially large gains on long-text or high-degree graphs where prompting quickly exceeds the token budget. (2) All interaction strategies remain effective on heterophilic graphs, challenging the assumption that LLM-based methods collapse under low homophily. (3) Code generation mode is able to flexibly shift its reliance to the most informative input type, whether that be structure, features, or labels. Together, these results establish a clear picture of the strengths and limitations of current LLM–graph interaction modes and point to design principles for future methods.

1 Introduction

Large language models (LLMs) have rapidly evolved into versatile problem solvers with strong in-context learning, reasoning, and tool-use abilities [1–3]. Their capabilities extend across natural language [4], computer vision [5], and code completion and synthesis [6]. Much like in other domains, recent work in graph machine learning has explored leveraging LLMs for tasks such as node classification [7–9], graph property prediction [10] and knowledge graph reasoning [11], with node classification emerging as a dominant task.

This recent wave of interest is natural as many high-impact node classification applications are inherently text-rich and thus well suited to the linguistic processing capabilities of LLMs, such as information retrieval [12], fraud detection [13], and recommendation systems [14]. For instance, in fraud detection, accounts are nodes, transactions are edges, and the goal is to use textual and relational metadata to assign risk labels to nodes early to prevent financial losses with devastating implications.

Consequently, LLMs have emerged as a viable alternative to the dominant paradigm for graph understanding, Graph Neural Networks (GNNs) [15–17], and exhibit competitive performance on text-rich graphs [18]. While GNNs are typically trained per task and dataset and do not transfer

*Work performed while at Microsoft Research.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: New Perspectives in Advancing Graph Machine Learning.

across domains or label spaces [19], a key advantage of LLMs is their broad world knowledge [20], which can benefit long-text graph datasets, such as e-commerce, web-link, and social networks [21–25]. Furthermore, LLMs have many ways to process and reason over graph information: through linearization of text and prompt augmentation; to specialized tool usage for querying the underlying graph; to generating arbitrary code that operates over the graph. We refer to these different approaches as the LLM's interaction strategies/modes with respect to graph data.

However, despite rapid adoption of LLMs in graph understanding, most prior work focuses on optimizing performance for specific domains, graphs, or tasks. As a result, the field currently lacks a principled understanding of the capabilities of LLMs in processing graph information, and learnings that practitioners can leverage when integrating them into their scenarios or applications.

This principled understanding is especially important, since there are many axes of variability. Thus, in this paper we conduct a comprehensive, controlled, large-scale evaluation that factorizes the key axes of variability in using LLMs for graphs: (1) the **LLM-graph interaction mode**, comparing four prompting variants, two ReAct-style tool-using variants [26], and a programmatic Graph-as-Code medium; (2) **dataset domains**, spanning citation, web-link, e-commerce, and social networks; (3) **homophilic vs. heterophilic regimes**; (4) **textual feature length**, comparing short- versus long-text attributes; and (5) **model sources**, contrasting closed-source models (o4-mini, GPT-5) with open-source alternatives (DeepSeek R1). Without a comprehensive understanding of this combinatorial space, blind application of LLMs to graphs can lead to sub-optimal, even detrimental outcomes.

Next, we quantify the dependency of the leading interaction strategies on features, structure, and labels at inference time via controlled ablations that independently truncate textual features or remove known labels, while varying edge deletion, yielding response surfaces (heatmaps) per dataset that characterize each interaction mode's dependence on features, labels, and structure.

Findings. Our evaluation yields key insights and guidelines in applying LLMs to graph data:

- **Graph-as-Code achieves the strongest overall performance**, with especially large gains on long-text or high-degree graphs where prompting quickly exhausts the token budget.
- All LLM-graph interaction modes remain effective on heterophilic graphs, challenging the commonly held assumption that LLM-based methods collapse under low homophily [8].
- **Feature-structure dependencies align across modes** when prompts fit the context window.
- **Graph-as-Code is able to flexibly shift its reliance** to the most informative input type, whether that be structure, features, or labels.
- Graph-as-Code is consistently robust to feature truncation, edge deletion and label deletion, in contrast to the brittleness of Prompting.

Related works and additional experiments are presented in the appendix.

2 Axes of Variability

Our goal is to build a principled understanding of the capabilities of LLMs in processing graph information. In this section, we systematically factorize the key axes of variability, enabling controlled comparisons that isolate the influence of each factor. This approach allows us to identify the conditions under which each LLM-graph interaction mode succeeds or fails, and to clarify their dependencies on graph features, structure and labels.

Setup. The goal is to predict labels for a set of held-out nodes given (i) labels for nodes in the training and validation splits (test labels are withheld), (ii) the graph structure, and (iii) the textual features of all nodes. All datasets use the canonical train/validation/test split from their source. For test sets larger than 1,000 nodes, we randomly subsample 1,000 nodes per run.

Variability over LLM sources (Section B.3). We evaluate both closed-source (`o4-mini`, `GPT-5`) and open-source (`DeepSeek R1`) models, using `o4-mini` as the primary unless stated otherwise.

2.1 Variability over LLM-Graph Interaction Strategies

We study the following three LLM-graph interaction modes:

(1) Prompting. In this simple and widely-used mode [7, 8, 27], the entire context for the model is constructed and issued to the model as a single-turn inference. The prompt (i) states the task and

provides the class-index mapping, (ii) presents the target node's textual description and known label (if available), and (iii) serializes the k-hop neighborhood grouped by hop distance, specifying for each encountered node its description and label (or *None* for held-out nodes).

The hop number is a hyperparameter controlling how many hops of neighborhood information are included in the prompt. We experiment with four variants: *Self-prompt*, *1-hop prompt*, and *2-hop prompt*, corresponding to radii of 0, 1, and 2, respectively. To keep the context within a token budget for long-text datasets, we apply *budget prompt*, which caps the neighbors at each hop by subsampling.

(2) GraphTool. Motivated by ReAct [26], we frame node classification as an iterative *think–act–observe* loop. At each step, the LLM reasons about what is known and what remains missing, then issues a single action from a fixed tool set. The environment executes the action on the graph and returns the result, which is appended to the interaction history. The process repeats until the LLM decides to terminate the process and predict a label. This ReAct-style interaction encourages planning and targeted retrieval of graph structure and text, reducing irrelevant exposure and token usage.

In our basic variation, *GraphTool*, the following actions are available: (1) A topology-only action retrieves the neighbors of a specified node, enabling exploration without consuming feature tokens. (2) A feature-only action returns the textual description of a specified node. (3) A label-only action reveals the label of the requested node if in the training set (and *None* otherwise), allowing the model to anchor reasoning on known examples while avoiding leakage on held-out nodes. (0) The terminal action submits the final label. We also introduce *GraphTool+*, which extends the base GraphTool variant with additional exact-k hop retrieval actions: (4) retrieves the textual descriptions of all nodes exactly k hops away from a specified node; (5) retrieves their labels (or *None* for held-out nodes).

(3) Graph-as-Code. Building on LLMs' strong code completion capabilities [6, 28], we extend the ReAct paradigm beyond a fixed, predefined action set. In the *Graph-as-Code* mode, the graph data is represented as a typed table indexed by `node_id` with columns `features` (text), `neighbors` (list of node IDs), and `label` (integer or *None*). The LLM generates, executes, then reasons over the outputs of compact programs in an iterative fashion. The process repeats until the LLM decides to terminate the process and predict a label. This code-native mode enables compositional access to structure and features and can collapse multi-step tool sequences into a single query, improving step and token efficiency while remaining transparent and auditable.

2.2 Variability over Dataset Domains, Homophily Levels and Text Lengths

Dataset domains. We evaluate LLM performance across diverse graph domains, such as the citation network dataset *cora*, *pubmed*, and *arxiv*, where nodes are papers with titles as features [8]; the e-commerce graph datasets *products*, *computers* and *photo* [8, 9], where nodes are items with product title, descriptions or reviews; the web-link network datasets *cornell*, *texas*, *washington*, *wisconsin*, and *wiki-cs* [9], where nodes are webpages described by their page-level text; the social network datasets *reddit* and *instagram* with user profiles and comment snippets [9]. This domain variability enables a comprehensive assessment of LLM generalization and adaptation to different graph types.

Homophilic vs. heterophilic regimes. Homophily is the tendency of nodes to connect with others of the same class, whereas heterophily is the tendency to connect predominantly with nodes of different classes. In homophilic graphs such as *cora*, *pubmed*, *arxiv*, *products*, *computers*, *photo*, *wiki-cs*, *reddit*, and *instagram* render local label information highly important for for correct prediction. Conversely, heterophilic graphs such as *cornell*, *texas*, *washington*, and *wisconsin* challenge models to rely less on simple local label information and more on node features and graph structure. By evaluating performance across these regimes, we aim to uncover the varying dependencies of LLMs on graph features, structure, and labels.

Textual feature lengths. The datasets also vary in the richness and complexity of node textual features. Short-text datasets, such as *cora*, *pubmed*, *arxiv* and *products* provide only titles or product names, offering limited semantic signal for classification. In contrast, long-text datasets such as *computers*, *photo*, *reddit*, *instagram* and *wiki-cs* include detailed descriptions or user profiles, presenting both opportunities for deeper reasoning and challenges for efficient context processing by LLMs.

3 Experiments across Domains, Homophily Levels and Text Lengths

We evaluate LLMs across multiple axes of variability, including LLM-graph interaction strategies, dataset domains, homophilic vs. heterophilic regimes, textual feature lengths and model sources. For

Table 1: Accuracy of baselines and LLM-graph interaction modes Prompting, GraphTool, and Graph-as-Code on heterophilic datasets. Best per-dataset results are **bold**, runner-up underlined.

	cornell	texas	washington	wisconsin
# Nodes	191	187	229	265
# Classes	5	5	5	5
Hom. (%)	11.55	6.69	17.07	16.27
Avg. degree	1.53	1.66	1.72	1.89
Random	21.74±3.19	8.40±1.56	20.43±2.87	18.11±3.62
Majority label	42.43±1.56	58.70±1.40	45.94±3.64	44.15±2.20
Label propagation	41.74±1.06	**78.90**±1.67	15.07±4.21	14.21±2.69
Self prompt	81.57±1.80	53.20±3.19	80.14±2.54	84.78±2.86
1-hop prompt	81.39±0.99	71.40±2.07	81.74±1.80	88.81±1.43
2-hop prompt	84.17±1.43	TokenLimit	**84.35**±1.67	**91.45**±1.87
GraphTool	91.30±2.46	59.60±2.38	80.14±1.32	87.04±1.58
GraphTool+	91.13±2.97	63.70±2.36	80.41±0.94	87.42±1.60
Graph-as-Code	**92.70**±2.35	73.60±3.78	81.96±2.92	89.17±2.69

clarity, we split the experiments into three parts: short-text homophilic datasets (section B.1 in the Appendix), heterophilic datasets (section 3.1), and long-text homophilic datasets (section 3.2). In each setting, we introduce additional baselines to contextualize performance.

Baselines. We compare against several baselines: **Random**, which predicts labels uniformly at random, serving as a natural lower bound; **Majority Label**, which assigns the most frequent label from the training and validation sets to all test nodes; and the classic **Label Propagation** algorithm.

3.1 Heterophilic datasets

Finding 1. *All LLM-graph interaction modes are effective on heterophilic datasets.*

Finding 2. *In ReAct-based methods, more flexible variants perform better.*

Discussion. Table 1 challenges common assumptions and prior work [8] which suggests LLMs struggle on heterophilic graphs. In this setting, local label information can be non-predictive or even misleading, making it difficult for LLMs to rely on neighborhood cues for accurate classification. Despite low levels of homophily, all LLM-graph interaction modes achieve strong accuracy (see Finding 1), consistently outperforming classic baselines such as majority labeling and label propagation. This demonstrates that LLMs can exploit non-local or feature-based cues for classification, rather than relying solely on simple neighborhood voting heuristics. Here, the *context window token limit is reached, primarily due to the long textual features*, rather than graph degree (see Section B.1), which constrains the amount of context that can be included in prompts. Nevertheless, providing more context in prompting proves beneficial, contrary to popular belief, and similar to the homophilic setting.

Furthermore, ReAct-based variants exhibit a clear advantage with increased agency and adaptivity in interacting with the graph – moving from fixed tool invocation (GraphTool), to enhanced retrieval options (GraphTool+), and finally to the fully programmatic Graph-as-Code—classification, reinforcing Finding 2 in the heterophilic setting, with additional experiments in Section B.1 supporting the homophilic case. The Graph-as-Code variant in particular excels, likely due to its compositional access to both features and structure, which becomes especially advantageous when neighborhood labels are diverse or uninformative. Overall, Table 1 reveals patterns similar to those observed in homophilic graphs (Table 3), contrary to common assumptions and prior work [8].

3.2 Long-text datasets

Finding 3. *Graph-as-Code significantly outperforms Prompting and GraphTool on long-text datasets.*

Discussion. Table 2 suggests of a performance gaps, with Prompting performing worst and Graph-as-Code best. In this setting, Prompting is constrained by node feature length and neighborhood size, both quickly exhausting the model's token budget. Notably, similar token-limit issues appeared in other sections—for the *products* dataset in Section B.1 and for the *texas* and *washington* datasets in Section 3.1 when features were long or neighborhoods large.

While LLM context windows are steadily increasing, relying solely on ever-larger windows is not a sustainable solution. Real-world graphs such as social networks are not only vast and feature-rich,

Table 2: Accuracy of baselines and LLM-graph interaction modes Prompting, GraphTool, and Graph-as-Code on long-text homophilic datasets. Best per-dataset results are **bold**, runner-up underlined.

	citeseer	reddit	computer	photo	instagram	wiki-cs
# Nodes	3,186	33,434	87,229	48,362	11,339	11,701
# Classes	6	2	10	12	2	10
Hom. (%)	72.93	55.52	85.28	78.50	63.35	68.67
Avg. degree	1.34	5.93	8.27	10.36	12.70	18.45
Random	16.80±2.05	51.60±2.97	10.00±2.21	8.20±3.03	50.30±3.07	9.90±2.13
Majority label	21.90±2.10	52.80±2.97	24.20±2.89	42.30±2.25	**65.10**±2.90	21.50±3.89
Label propagation	37.30±5.03	40.50±1.66	74.70±1.60	75.30±0.91	52.00±2.98	71.90±2.19
Self prompt	68.20±2.77	47.90±3.86	65.50±3.32	69.80±3.15	48.00±3.66	74.00±3.10
1-hop prompt	68.30±2.59	59.30±4.12	86.10±2.59	85.80±1.92	56.10±2.30	TokenLimit
2-hop prompt	69.40±2.10	TokenLimit	TokenLimit	TokenLimit	TokenLimit	TokenLimit
2-hop budget prompt	70.20±2.83	54.40±3.85	86.00±2.41	85.60±1.63	54.50±4.45	80.80±3.48
GraphTool	68.30±1.15	56.25±1.84	80.80±4.61	77.00±2.32	47.80±4.09	76.27±4.18
GraphTool+	68.70±2.31	**61.80**±1.15	83.10±2.63	81.30±0.97	48.20±2.92	80.50±3.10
Graph-as-Code	**71.80**±2.22	61.60±2.36	**86.20**±3.55	**86.40**±2.65	56.40±2.56	**82.20**±3.63

but are also continuously growing in size, much like the context windows themselves. Consequently, nodes with extremely large neighborhoods or feature sets may still exceed available context limits. Moreover, larger windows incur greater computational cost and latency, and do not address the fundamental challenge of prioritizing and aggregating relevant information. Thus, approaches that intelligently manage or sample context, rather than indiscriminately expanding it, remain relevant.

To mitigate these constraints, we introduce the 2-hop budget prompting variant, which caps the number of neighbors per node via sampling. While this adjustment helps avoid hitting the token limit and allows the model to reason over a sampled subset of context, yet Prompting still remains the least effective variant, with Graph-as-Code the most effective (see Finding 2). This is likely due to the noise and information loss introduced by sampling, which can obscure important neighborhood signals. **These results demonstrate that Graph-as-Code can offer substantial advantages for LLM-based node classification in dense or feature-rich graphs**, which represent a large proportion of real-world networks such as e-commerce and recommendation networks.

ReAct-based variants consistently outperform Prompting by enabling LLMs to dynamically and iteratively access relevant graph structure and node features, rather than relying on a fixed serialized input. Among them, Graph-as-Code offers the greatest flexibility by composing function calls over both structural and textual information without being constrained by context-window limits, thereby achieving the highest performance (see Finding 2). This finding aligns with the trend observed on short-text homophilic datasets, where increased flexibility in ReAct variants yields stronger performance. Ultimately, this adaptability allows ReAct-based methods to fully exploit information even in large, verbose graphs, while Prompting remains fundamentally bottlenecked by input length.

4 Experiments on features, structure, and labels dependencies

In this section, we examine whether the widely used Prompting and the best performing Graph-as-Code LLM-graph interaction modes rely on node features, graph structure, and labels in similar ways. While previous sections compared overall accuracy across settings, the analysis in this section sheds light on the inner workings of LLM-based approaches. By isolating the contributions of features, labels, and structure, practitioners can identify which LLM-graph interaction strategies leverage specific types of information most effectively. This helps guide the selection of LLM-graph interaction modes that align with the particular characteristics (feature length, homophily levels) of their datasets, rather than relying on opaque, one-size-fits-all solutions.

We organize the analysis into two parts: (i) the effect (or dependency) of removing node features and edges (section B.2), and (ii) the effect of removing labels and edges (section 4). To visualize these effects, we generate 2D heatmaps showing model accuracy as a function of the removal rates.

Additional setup. In addition to the setup described in Section 2 in Sections B.2 and 4 we run partial-deletion experiments: edges and labels are removed uniformly at random, while feature deletion is implemented by truncating each node's text to the fixed percentage of tokens.

Labels vs. structure dependencies

Finding 4. *Prompting and Graph-as-Code exhibit different dependencies on labels and structure.*

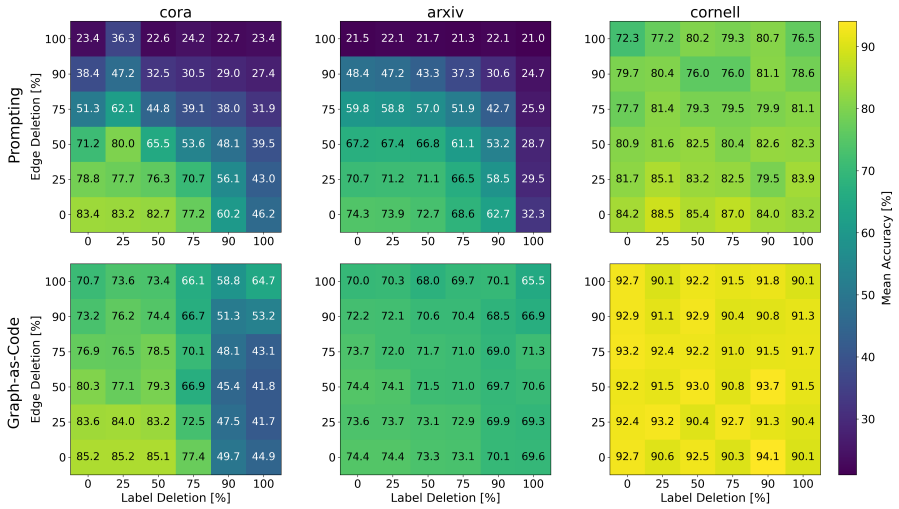

Figure 1: Accuracy of 2-hop prompting and Graph-as-Code on the cora, arxiv, and cornell datasets under varying ratios of randomly removed edges and known labels.

Finding 5. *Graph-as-Code is able to flexibly shift its reliance to the most informative input type, whether that be structure, features, or labels.*

Finding 6. *Graph-as-Code is more robust than Prompting to feature, structure and label deletion.*

Discussion. Figure 1 demonstrates a stark contrast between the dependence patterns of Prompting and Graph-as-Code when subjected to edge and label deletion (see Finding 4). This behavior is in contrast to the alignment on feature and structure dependencies observed in Section B.2.

For Prompting, the results across all datasets reveal that accuracy degrades rapidly along both axes, confirming that Prompting requires both structure and labels to perform reasonably. Graph-as-Code by comparison, displays a notably different pattern. Its accuracy remains nearly constant as edges are deleted, as long as either features or labels are present. This might suggest that Graph-as-Code ignores structure entirely; however, as observed in Section B.2, structural information becomes crucial only when features are truncated. Thus, Graph-as-Code does not disregard structure, but instead leverages it only when it is more informative relative to other available signals.

This leads to a key insight: Graph-as-Code can flexibly shift its reliance to the most informative input type, whether that is structure, features, or labels, and by doing so, it is only vulnerable when multiple sources of information are heavily degraded (Finding 5). This adaptive behavior contrasts sharply with the brittleness of Prompting and re-emphasizes the robustness of Graph-as-Code (Finding 6).

5 Conclusions

In this work, we conducted the first comprehensive, controlled evaluation of LLMs for node classification across key axes of variability: LLM-graph interaction mode (Prompting, ReAct-style tool use, and Graph-as-Code), dataset domain (citation, web-link, e-commerce, social), homophily regime (homophilic vs. heterophilic), textual feature length (short vs. long), and model source (closed vs. open). Our large-scale study reveals that the Graph-as-Code method, which leverages LLMs' coding capabilities, achieves the best performance, especially on graphs with long-text features or dense nodes, where the widely-used prompting method quickly becomes infeasible due to context window limitations. We also find all LLM-graph interaction methods to be effective on heterophilic graphs, challenging commonly held assumption that LLM-based methods fail in low-homophily [8].

Through a series of controlled dependency analyses, we independently truncate features, delete edges, and remove labels to quantify reliance on different input types. Experiments show that Graph-as-Code flexibly adapts its reliance to the most informative signal, be it structure, features, or labels.

Our findings provide actionable guidance for both practitioners and researchers: (1) Code generation should be the preferred LLM-graph interaction mode, particularly as graphs grow in size and complexity; (2) LLMs remain effective on heterophilic graphs; (3) Exploit Graph-as-Code's adaptive reliance to handle noisy or partially missing data, where different signals may be degraded.

References

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

[2] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

[3] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

[4] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020.

[5] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *arXiv*, 2023.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv*, 2021.

[7] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In *ICLR*, 2024.

[8] Jin Huang, Xingjian Zhang, Qiaozhu Mei, and Jiaqi Ma. Can llms effectively leverage graph structural information through prompts, and why? In *TMLR*, 2024.

[9] Xixi Wu, Yifei Shen, Fangzhou Ge, Caihua Shan, Yizhu Jiao, Xiangguo Sun, and Hong Cheng. When do llms help with node classification? a comprehensive analysis. In *ICML*, 2025.

[10] Taicheng Guo, Kehan Guo, Bozhao Nan, Zhenwen Liang, Zhichun Guo, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. What can large language models do in chemistry? a comprehensive benchmark on eight tasks. In *NeurIPS*, 2023.

[11] Yuqi Zhu, Xiaohan Wang, Jing Chen, Shuofei Qiao, Yixin Ou, Yunzhi Yao, Shumin Deng, Huajun Chen, and Ningyu Zhang. Llms for knowledge graph construction and reasoning: Recent capabilities and future opportunities. In *arXiv*, 2023.

[12] Weihang Su, Qingyao Ai, Xiangsheng Li, Jia Chen, Yiqun Liu, Xiaolong Wu, and Shengluan Hou. Wikiformer: Pre-training with structured information of wikipedia for ad-hoc retrieval, 2024.

[13] Chengdong Yang, Hongrui Liu, Daixin Wang, Zhiqiang Zhang, Cheng Yang, and Chuan Shi. Flag: Fraud detection with llm-enhanced graph neural network. In *SIGKDD*, 2025.

[14] Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan E. Lenssen, Yiwen Yuan, Zecheng Zhang, Xinwei He, and Jure Leskovec. Relbench: A benchmark for deep learning on relational databases, 2024.

[15] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[16] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[17] Ben Finkelshtein, Xingyue Huang, Michael Bronstein, and İsmail İlkan Ceylan. Cooperative graph neural networks. In *ICML*, 2024.

[18] Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. Language is all a graph needs. In *ACL*, 2024.

[19] Ben Finkelshtein, İsmail İlkan Ceylan, Michael Bronstein, and Ron Levie. Equivariance everywhere all at once: A recipe for graph foundation models. *arXiv*, 2025.

[20] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *EMNLP*, 2020.

[21] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation, 2019.

[22] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020.

[23] Hongbin Pei, Bingzhe Wei, Kevin C.-C. Chang, Yizhou Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2020.

[24] Peter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv*, 2020.

[25] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[26] Shunyu Yao, Jeffrey Zhao, Dian Yu, Prithviraj Ammanabrolu, Matthew Hausknecht, and Karthik Narasimhan. React: Synergizing reasoning and acting in language models. *arXiv*, 2023.

[27] Zhong Guan, Likang Wu, Hongke Zhao, Ming He, and Jianpin Fan. Attention mechanisms perspective: Exploring llm processing of graph-structured data. In *ICML*, 2025.

[28] Xiangyan Liu, Bo Lan, Zhiyuan Hu, Yang Liu, Zhicheng Zhang, Fei Wang, Michael Shieh, and Wenmeng Zhou. Codexgraph: Bridging large language models and code repositories via code graph databases. In *ACL*, 2025.

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

[30] Xin Li, Weize Chen, Qizhi Chu, Haopeng Li, Zhaojun Sun, Ran Li, Chen Qian, Yiwei Wei, Chuan Shi, Zhiyuan Liu, et al. Can large language models analyze graphs like professionals? a benchmark, datasets and models. In *NeurIPS*, 2024.

[31] Xinnan Dai, Haohao Qu, Yifen Shen, Bohang Zhang, Qihao Wen, Wenqi Fan, Dongsheng Li, Jiliang Tang, and Caihua Shan. How do large language models understand graph patterns? a benchmark for graph pattern comprehension. In *ICLR*, 2025.

[32] Jianan Zhao, Le Zhuo, Yikang Shen, Meng Qu, Kai Liu, Michael M Bronstein, Zhaocheng Zhu, and Jian Tang. Graphtext: Graph learning in text space. *arXiv*, 2024.

[33] Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. Let your graph do the talking: Encoding structured data for llms. *arXiv*, 2024.

[34] Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. *SIGIR*, 2024.

[35] Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. Grapharena: Evaluating and exploring large language models on graph computation. *ICLR*, 2025.

[36] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv*, 2023.

[37] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *NeurIPS*, 2023.

[38] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI*, 2024.

[39] Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, et al. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. *arXiv*, 2024.

[40] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *NeurIPS*, 2024.

[41] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv*, 2023.

[42] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *NeurIPS*, 2023.

[43] Jiawei Zhang. Graph-toolformer: To empower llms with graph reasoning ability via prompt augmented by chatgpt. *arXiv*, 2023.

[44] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv*, 2024.

[45] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv*, 2021.

[46] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *AAAS*, 2022.

[47] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv*, 2023.

[48] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, et al. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *SIGKDD*, 2023.

[49] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *ICML*, 2023.

[50] Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. Paper2code: Automating code generation from scientific papers in machine learning. *arXiv*, 2025.

[51] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv*, 2023.

[52] Qiaolong Cai, Zhaowei Wang, Shizhe Diao, James Kwok, and Yangqiu Song. Codegraph: Enhancing graph reasoning of llms with code. *arXiv*, 2024.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The introduction states the main claims about comparing LLM interaction modes and their dependencies, and these are directly supported by the experiments and findings. The scope is clearly described, and the results match the stated contributions.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations related to the LLM-graph interaction modes presented, such as context window size, token budget constraints, and the challenges of scaling to large or feature-rich graphs. It also notes that increasing context windows is not a sustainable solution and addresses how sampling can introduce noise and information loss, impacting results.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: The paper does not include theoretical results.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper fully disclose all the information needed to reproduce the main experimental results. Specifically, we disclose all the exact prompt templates used throughout the work along with the dataset sources and split.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility.

In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: We do not disclose the code, as it is easily reproducible using the detailed prompt templates, specified LLM models, data sources, and splits.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: This work does not involve training any models. Instead, we provide comprehensive details on the prompt templates, specified LLM models, data sources, and data splits to ensure full reproducibility.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report the standard deviation in all tables.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies that all experiments were conducted using 8 Intel Xeon Platinum 8370C cpus.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research fully complies with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

Justification: The paper does not include a discussion of societal impacts, either positive or negative. Its focus is foundational research on LLMs for node classification, with does not have societal implications.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: Our work poses no such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the relevant data sources and credit them for every dataset used.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We do not release any new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The core focus of the paper is evaluating and comparing different modes of LLM usage for node classification on graphs. The methodology and experiments explicitly detail how LLMs are used, including prompting, ReAct-style tool use, and programmatic Graph-as-Code.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Related Work

Textualization and prompting of graphs. Motivated by LLMs' cross-domain performance, early work encodes graphs as text for LLMs, benchmarking varying encoding styles such as adjacency lists, edge lists, shortest-path descriptions, and narrative-style encodings [7]. Subsequent studies employ these textualizations to evaluate LLMs on node classification using prompting setups [8, 29–31, 18, 27]. They find that carefully designed prompts can allow LLMs to compete with GNNs [18], that performance often hinges on neighborhood homophily [8], and that their abilities remain brittle and sensitive to input data and formatting [29]. More detailed analyses of attention patterns further suggest that LLMs may mirror prompt format rather than execute explicit graph computation [27].

Concurrently, natural alternatives to textualization have emerged, either by tokenizing each node based on local structure and features [32] or by introducing learnable components that encode structure and features [33]. However, none have replaced prompt-based approaches in practice and most recent work still relies on prompting for graph tasks [9, 34, 35, 31, 27]. We challenge the prompting interaction mode by proposing GraphTool and Graph-as-Code, which we show to achieve competitive performance and often surpasses prompting on graphs with large textual features or high-degree nodes. Contrary to conclusions drawn from prompting alone [8], we also find all modes viable on heterophilous graphs, with the added advantage that Graph-as-Code shows reduced brittleness.

Tool-calling for graph reasoning. Recent advances in LLM orchestration have introduced tool-calling and ReAct-style paradigms, enabling LLMs to interface with external APIs or reasoning modules for enhanced problem-solving [26, 3]. These approaches interleave natural language reasoning with calls to task-specific tools, allowing LLMs to retrieve, manipulate, or compute over structured data. Variants such as Plan-and-Execute [36], Reflexion [37], and Graph-of-Thought [38] have shown that LLMs can decompose complex tasks into sequences of actions and adapt their plans via feedback. Tool-calling has been extensively benchmarked for real-world utility across domains—planning [39], code API usage [40], mathematical reasoning [41], and multi-modal reasoning [42]. In the context of graph data, LLMs have been combined with tool-calling to perform graph classification, knowledge graph reasoning and node classification, achieving improved performance [43, 44]. However, these works focus on specialized workflows or proof-of-concept demonstrations. Our work extends this line of research by systematically evaluating varying tool-calling paradigms for node classification across diverse datasets and graph regimes, highlighting their strengths, limitations and dependencies on the features, labels and structure.

LLMs coding capabilities. Code generation has become a native capability of modern LLMs, with early systems demonstrating high-quality programs from natural language prompts [45], even reaching competition-level performance [46] and fueling the development of strong open-source models such as StarCoder [47] and CodeGeeX [48]. Beyond pure code synthesis, program-aided approaches have leveraged code as a medium for reasoning—enabling LLMs to generate and execute short programs for mathematical problem solving [49], automate scientific workflows [50], and enhance reliability through self-debugging and iterative refinement [51]. Recently, these capabilities have been extended to graph domains, where LLMs are prompted to generate code for solving simple graph problems such as cycle detection, connectivity analysis, and node degree estimation [52]. We extend this further with Graph-as-Code: the LLM iteratively writes and executes concise programs over a standardized API to flexibly combine structural queries, label propagation, and textual feature processing. We then compare Graph-as-Code to prompting and tool-calling across datasets and asses its dependencies on the input feature, structure and labels.

B Additional Experiments

In this section, we present additional experiments and findings that provide deeper insights into LLM-based node classification. We first examine performance across short-text homophilic datasets (see Section B.1). Next, we investigate the effect (or dependency) of removing node features and edges on LLM-graph interaction strategies (Section B.2. Finally, we compare open-source and closed-source LLM models, to assess the impact of model sources on our findings (Section B.3).

B.1 Short-text homophilic datasets

Finding 7. *Prompting and Graph-as-Code are closely competitive on short-text homophilic datasets.*

Table 3: Accuracy of baselines and LLM-graph interaction modes Prompting, GraphTool, and Graph-as-Code on short-text homophilic datasets. Best per-dataset results are **bold**, runner-up underlined.

	cora	pubmed	arxiv	products
# Nodes	2,708	19,717	169,343	2,449,029
# Classes	7	3	40	47
Hom. (%)	82.52	79.24	65.53	63.84
Avg. degree	4.92	6.30	13.64	61.37
Random	14.13±1.06	33.10±1.19	2.44±0.23	2.33±0.25
Majority label	29.00±0.89	41.90±4.08	5.90±1.39	26.10±2.49
Label propagation	76.61±1.94	80.80±2.93	68.00±1.66	70.40±1.64
Self prompt	64.17±0.68	89.20±1.89	68.10±3.11	70.00±5.82
1-hop prompt	81.92±1.86	91.30±2.02	73.80 ±1.92	82.20±3.98
2-hop prompt	83.43±2.25	91.80±2.17	74.30±2.53	TokenLimit
GraphTool	74.02±1.18	89.50±2.32	67.50±5.50	75.30±3.06
GraphTool+	81.40±3.08	**91.90**±2.16	73.30±2.86	78.50±3.43
Graph-as-Code	**85.16**±1.47	89.90±1.85	**74.40**±3.02	**82.70**±2.66

Discussion. Foremost, Table 3 reaffirms prior work by showing that all LLM-based approaches substantially outperform trivial baselines such as random guessing and majority label assignment. This confirms that LLMs leverage both textual node features and graph structure for classification in homophilic regimes. Furthermore, within the Prompting interaction mode, accuracy increases with the inclusion of neighborhood context, moving from self to 1-hop and 2-hop prompt variants, which is consistent with established findings [8, 9]. However, *on graphs with high average degree, context token limits are reached quickly*, restricting possible gains from additional neighborhood information.

Furthermore, we find that Prompting and Graph-as-Code perform competitively on short-text homophilic graphs (see Finding 7), positioning Graph-as-Code as a viable alternative to Prompting. Notably, the ReAct-based interaction strategies exhibits a clear trend from GraphTool to GraphTool+ to Graph-as-Code. As the LLM is given greater agency and adaptivity in interacting with the graph – moving from fixed tool invocation (GraphTool), to enhanced retrieval options (GraphTool+), and finally to the fully programmatic Graph-as-Code—classification, accuracy improves (see Finding 2). We believe that adaptivity is valuable in homophilic settings, as local neighborhood labels are highly informative but the optimal aggregation strategy may also depend on node degree and graph topology. Thus, increased agency empowers the LLM to tailor its reasoning and retrieval to the specific structure of each instance, resulting in stronger overall performance.

B.2 Features vs. structure dependencies

Finding 8. *Prompting and Graph-as-Code exhibit comparable use of node features and structure.*

Finding 9. *When the prompt size reaches the token limit, the behavior of Graph-as-Code and Prompting diverges, with Graph-as-Code performing significantly better.*

Discussion. Figure 2 shows that, for all three datasets, the two panels have nearly identical characteristics, indicating that Prompting and Graph-as-Code share the same dependence on features and structure. On *cora* and *arxiv*, accuracy drops mainly with edge deletion, while on *cornell* it is driven primarily by feature deletion (Finding 8). This alignment arises from the inherent characteristics of the datasets. Both *cora* and *arxiv* are highly homophilic, meaning nodes of the same class are densely interconnected. In these settings, structural information, specifically the local label context provided by edges, has high impact on accuracy, so removing edges disrupts information flow and leads to a decrease in accuracy for both methods. Conversely, *cornell* is a heterophilic dataset, where nodes of different classes are more likely to be connected and the graph structure is less informative. Here, node features are more discriminative than the sparse and less meaningful edge connections, making feature deletion the dominant factor impacting performance.

Beyond this shared dependency, Graph-as-Code consistently outperforms Prompting and is more resilient to perturbations (Finding 6). When structure is completely removed but features are intact, Graph-as-Code preserves high accuracy on all datasets, whereas Prompting collapses. This difference

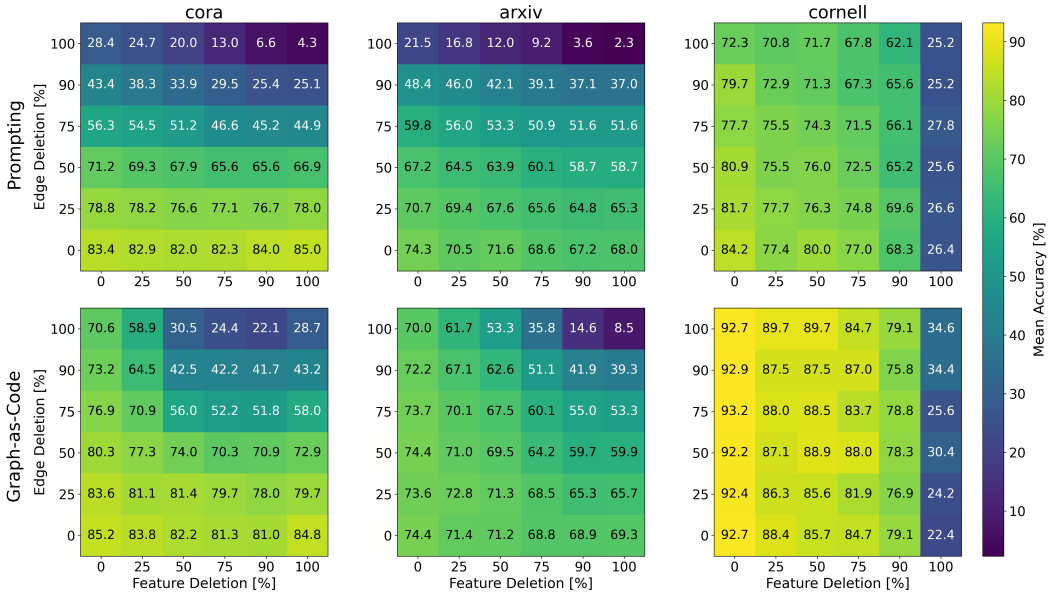

Figure 2: Accuracy of 2-hop prompting and Graph-as-Code on the cora, arxiv, and cornell datasets under varying ratios of randomly removed edges and truncated text features.

arises because Graph-as-Code can access feature and label information of other nodes even when edges are absent, whereas Prompting depends on edge connections to retrieve this information.

We further challenge Finding 8 by searching for a setting where the two methods might not align. Long-text homophilic graphs are precisely such a case because 2-hop prompts are prone to hitting the token limit. Figure 3 shows that on *photo* we indeed observe a divergence in behavior between the two methods (Finding 9). This divergence on *photo* is expected given the dataset's characteristics. *Photo* is highly homophilic and contains nodes with rich, lengthy feature descriptions. In homophilic graphs, nodes are densely connected to others of the same class, so 2-hop prompts accumulate a substantial amount of feature text from numerous neighbors. This quickly exceeds the LLM's context window, leading to a significant drop in accuracy for Prompting. In contrast, Graph-as-Code is designed to selectively retrieve and compose only the necessary structure and features for each query. This allows it to avoid exceeding the token limit and maintain high accuracy, even in the presence of long node descriptions and dense connectivity.

This result ties back to our long-text homophilic experiments and findings in Section 3.2. There we observed a large gap in favor of Graph-as-Code; the *photo* ablation reveals the same characteristic: Prompting is fundamentally bottlenecked by the context window and can even benefit from discarding feature text to fit within it, whereas Graph-as-Code retrieves and composes the needed structure and features without exceeding the token budget. Thus, the two methods share the same dependence on features and structure when prompts fit the context window, but once the

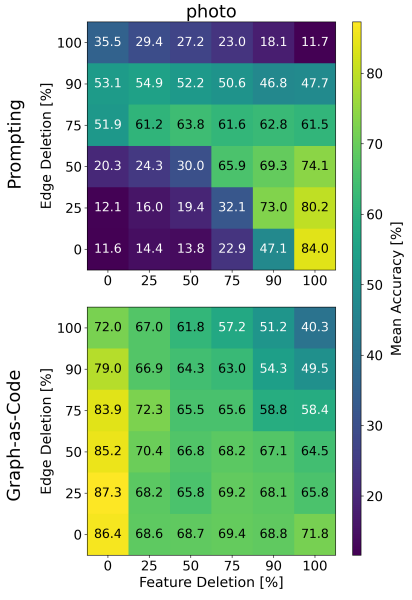

Figure 3: Accuracy of 2-hop prompting and Graph-as-Code on the photo dataset under varying ratios of randomly removed edges and truncated features.

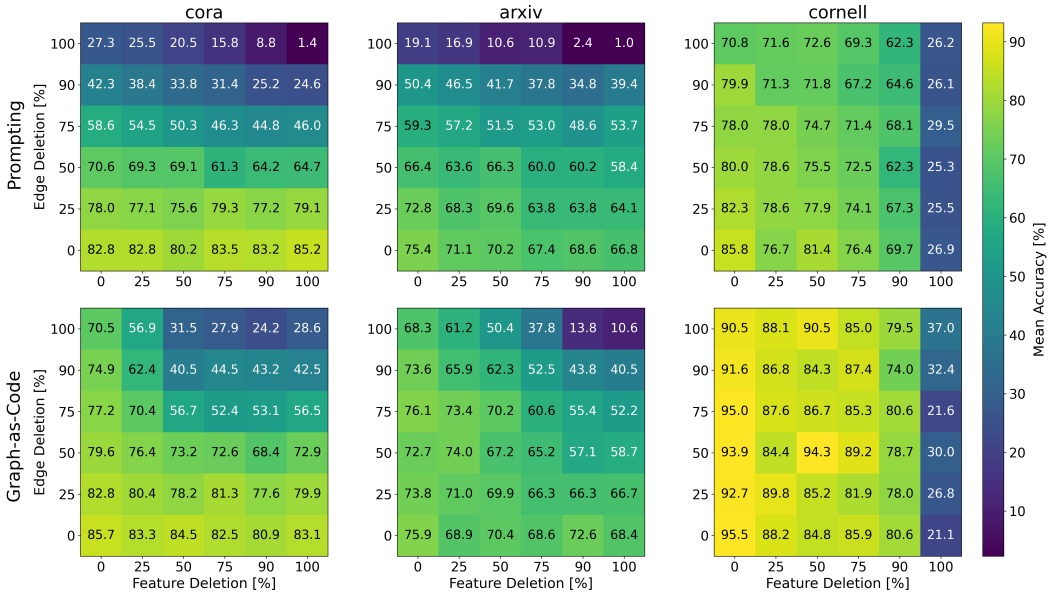

Figure 4: Accuracy of 2-hop prompting and Graph-as-Code on the cora, arxiv, and cornell datasets under varying ratios of randomly removed edges and truncated text features, evaluated using the close-source `GPT-5`.

token limit becomes a limiting factor, the behaviors diverge and Graph-as-Code becomes decisively superior. Practitioners should therefore assess the graph density and average feature length before choosing an LLM-graph interaction mode, prioritizing adaptive methods such as Graph-as-Code for cases with high density or long-text features.

B.3 Model sources

We adopt the experimental protocol outlined in Section B.2 to evaluate the performance of the closed-source `GPT-5` and the open-source `DeepSeek R1`.

Finding 10. *Open- and closed-source LLMs show comparable accuracy trends.*

Discussion. Figures 4 and 5 show that the patterns observed in section B.2 and the associated findings, such as the superior robustness of Graph-as-Code compared to Prompting (Finding 6) and the similar dependence of both methods on features and structure (Finding 8), hold consistently across both open- and closed-source LLMs.

Overall, these results suggest that the choice between open-source and closed-source LLMs does not substantially alter the qualitative behavior of LLM-graph interaction modes (Finding 10). Consequently, it is reasonable to assume that the majority of our key findings generalize to both open- and closed-source models. This indicates that practitioners can expect similar trends and trade-offs when deploying either type of model for node classification.

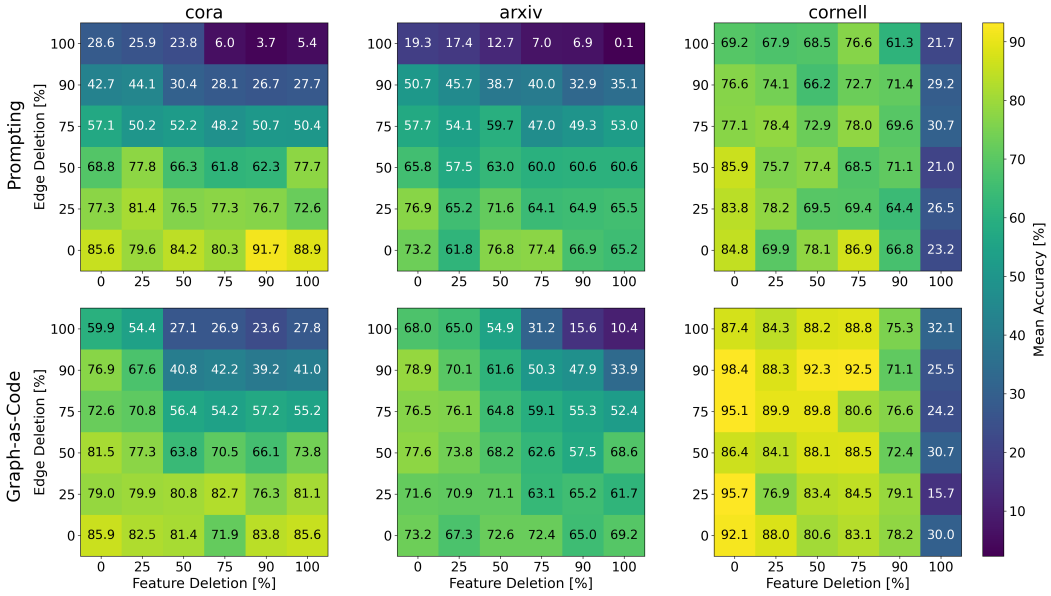

Figure 5: Accuracy of 2-hop prompting and Graph-as-Code on the cora, arxiv, and cornell datasets under varying ratios of randomly removed edges and truncated text features, evaluated using the open-source `DeepSeek R1`.

C Details on LLM-graph Interaction modes

In this section, we provide the full prompt templates used for each LLM-graph interaction mode evaluated in our experiments. These templates specify the instructions, available actions, and response formats provided to the LLM during node classification tasks. Specifically, we include the exact text used for the Prompting, GraphTool+, and Graph-as-Code modes, enabling reproducibility and facilitating future methodological comparisons.

Template 1 – Prompting

Task: You are solving a node-based task. Your goal is to determine the label for node `{node_id}`.
The final answer must be submitted as an integer corresponding to a class label. Below is the mapping from each integer index to its associated label.
Available class labels:
`0: {text description of label 0}`
`1: {text description of label 1}`
...
Node `{node_id}` has the textual description `{feat_id}` and belongs to label class `{label_id}` (or `None`).
Node `{node_id}` has the following neighbors 1-hop away:
Node `{n_1}` has the textual description `{feat_n1}` and belongs to label class `{label_n1}` (or `None`).
Node `{n_2}` has the textual description `{feat_n2}` and belongs to label class `{label_n2}` (or `None`).
...
Node `{node_id}` has the following neighbors 2-hops away:
Node `{m_1}` has the textual description `{feat_m1}` and belongs to label class `{label_m1}` (or `None`).
...
Think and end your response with: `Answer: [class_id]`.

Template 2 – GraphTool+

Task: You are solving a node-based reasoning task using interleaved steps. Your goal is to determine the label for node `{node_id}`. At each step, you may choose one of several available actions to gather information or submit your final prediction.
Instructions: Always begin with reasoning. You may take as many steps as needed, but aim to solve the task efficiently using the fewest necessary actions. Before each action, assess what information is available, what's missing, which action is most appropriate next, and how many steps likely remain. Then, on a new line, specify your chosen action using one of the formats below. It must be the final non-empty line of your response.
Available actions:
– `Action 0, answer class_id`: Submit your final answer as an integer label.
– `Action 1, node node_id`: Retrieve the list of neighboring nodes connected to the specified node.
– `Action 2, node node_id`: Retrieve the textual description (features) of the specified node.
– `Action 3, node node_id`: Retrieve the label of the specified node if it is in the training set; otherwise, return `None`.
– `Action 4, node node_id, hop num_hop`: Retrieve the textual descriptions (features) of all nodes that are exactly `num_hop` hops away from the specified node.
– `Action 5, node node_id, hop num_hop`: Retrieve the labels (or `None`) of all nodes that are exactly `num_hop` hops away from the specified node.
Available class labels:
`0: {text description of label 0}`
...
Now begin your reasoning in the Scratchpad below:

Template 3 – Graph-as-Code

Task: You are solving a node-based reasoning... for node `{node_id}`. You have a pandas DataFrame `df` where each row corresponds to a node, indexed by its `node_id`.
Instructions: Always begin with reasoning...
Schema structure:
– The DataFrame index is the node id. Access a row by node id with: `df.loc[node_id]`.
– The column `features` stores each node's textual description: `df.loc[node_id, 'features']`.
– The column `neighbors` stores a list of neighbor node IDs: `df.loc[node_id, 'neighbors']`.
– The column `label` contains the integer node label if it belongs to the training set; otherwise `None`.
You may query ANY column(s) of `df` using any valid pandas command that applies to a DataFrame named `df`. You may also use `pd.*` utilities with `df` as input. The dataframe can be long, so you may want to avoid commands that print the entire table.
Response format:
– For intermediate steps: reason then on the final line output a *single* valid pandas expression.
– To finish: reason then on the final line respond exactly as: `Answer [class_id]`.
Available class labels:
`0: {text description of label 0}`
...

All experiments were conducted using 8 Intel Xeon Platinum 8370C cpus.

D Dataset Statistics

The statistics of all datasets can be found in Table 4.

Table 4: Statistics of all datasets

	Dataset	#Nodes	#Edges	#Classes	Train/Val/Test (%)
Short-text homophilic	cora	2,708	5,429	7	60/20/20
	pubmed	19,717	44,338	3	60/20/20
	arxiv	169,343	1,166,243	40	53.7/17.6/28.7
	products	2,449,029	61,859,140	47	8.0/1.6/90.4
Heterophilic	cornell	191	292	5	60/20/20
	texas	187	310	5	60/20/20
	washington	229	394	5	60/20/20
	wisconsin	265	510	5	60/20/20
Long-text homophilic	citeseer	3,186	4,277	6	60/20/20
	reddit	33,434	198,448	2	60/20/20
	computer	87,229	721,081	10	60/20/20
	photo	48,362	500,928	12	60/20/20
	instagram	11,339	144,010	2	60/20/20
	wiki-cs	11,701	215,863	10	60/20/20