Are Large Language Models Suitable for Graph Computation? Progress and Prospects

Yuting Zhang[1] **Yi Han**[1] **Kai Wang**[2] * **Wei Ni**[3] **Angela Bonifati**[4] **Wenjie Zhang**[1]
[1]University of New South Wales
[2]Antai College of Economics and Management, Shanghai Jiao Tong University
[3]Edith Cowan University [4]Lyon 1 University
{yutingz, zhangw}@cse.unsw.edu.au yi.han.1@student.unsw.edu.au
w.kai@sjtu.edu.cn w.ni@ecu.edu.au angela.bonifati@univ-lyon1.fr

Abstract

Large language models (LLMs) have been increasingly explored for graph computation, where tasks require reasoning over structured relationships and algorithmic operations. Yet, it remains unclear when LLMs can reliably support such computation and how they should be incorporated into graph-solving pipelines. Existing surveys at the intersection of LLMs and graphs primarily focus on graph learning, text-attributed graphs, or graph-language modeling. To bridge this gap, we provide a comprehensive review of LLMs for graph computation through a role-based taxonomy. Specifically, we identify two major paradigms: i) LLMs as executors, where models directly solve graph tasks from graph descriptions and instructions; and ii) LLMs as planners, where models formulate problems, decompose reasoning steps, and invoke external tools or agents for execution. Based on this taxonomy, we analyze the strengths and limitations of current methods. Our review indicates that LLMs are promising for simple tasks on small graphs but remain unreliable for large graphs and tasks requiring exact results. Finally, we summarize available datasets and suggest four future directions.

1 Introduction

Graphs provide a natural representation for relationships in many real-world systems, such as social connections (Breuer et al., 2020), protein-protein interactions (Neyshabur et al., 2013), and paper citations (Das Gollapalli and Caragea, 2014). Analyzing such graph-structured data often requires graph computation, which derives explicit structural results or quantities from a given graph, such as shortest paths (Dijkstra, 1959), triangle counts (Azad et al., 2015), and cycle detection (Qiu et al., 2018). Such computation underlies a wide range of applications, including social network analysis (Newman, 2003), fraud detection (Lyu et al., 2020), and recommendation systems (He et al., 2017).

However, the reliance on specialized knowledge of graph computation techniques creates a significant barrier to applying graph-based analytics to real-world problems. Therefore, there is a need for approaches that can leverage LLMs to interpret user intent and translate it into computational solutions, lowering the barrier to graph computation (Liu et al., 2024a). Despite the needs and growing attention to the use of LLMs in addressing graph computational tasks (Guo et al., 2025), whether LLMs are fundamentally suitable for graph computation remains controversial. Graph computation often requires exact structural reasoning and faithful tracking of vertices and edges or multi-step algorithmic execution, which differ substantially from the probabilistic next-token prediction objective of LLMs. Existing studies have shown that LLMs may hallucinate graph structures, miss critical edges, and suffer from severe performance degradation as graph size or reasoning depth increases (Liu et al., 2024b; Wang et al., 2024b; Heyman and Zylberberg, 2025). These findings raise concerns about using LLMs for graph computation, especially in correctness-critical scenarios.

Although several surveys have explored LLMs for graphs, none have systematically examined the suitability of LLMs for graph computation. Ren et al. (2024) review methods that integrate LLMs with graph learning techniques, whereas Li et al. (2024c) investigate text-attributed graphs and categorize methods according to the roles of LLMs in the model pipeline. Jin et al. (2024a) survey various graph types, but do not clearly distinguish graph computation from graph learning for graphs with limited textual semantics. Chen et al. (2024b) investigate the node classification task within the context of graph learning. Focusing on retrieval-augmented generation with graphs, Han et al. (2025) briefly mention graph computation when discussing LLM-

*Corresponding author. Email: w.kai@sjtu.edu.cn (Kai Wang).

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 23984–24001
October 24-29, 2026 ©2026 Association for Computational Linguistics

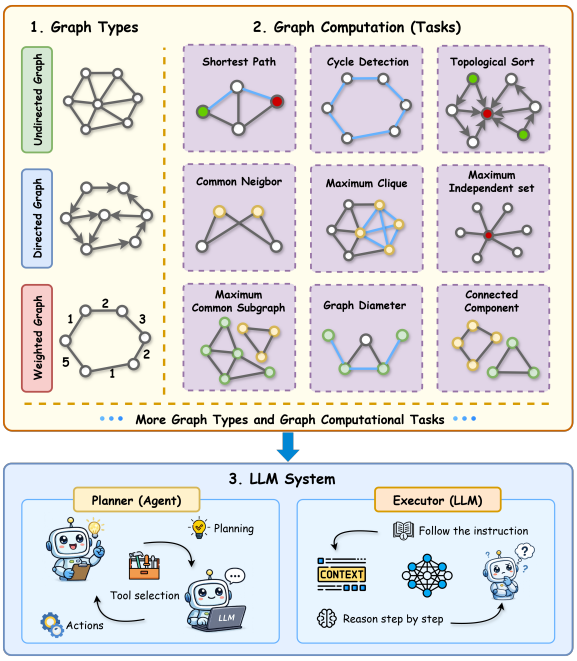

Figure 1: A framework for utilizing LLMs for graph computation tasks. The LLM system takes the graph structure and graph computation task as input, and acts as an executor or planner to deliver an answer.

based graph reasoning. Despite the rapid growth of this area, the literature on LLM-based graph computation has not yet been systematically organized.

This survey aims to bridge this gap by providing an in-depth and systematic review focusing on LLMs for graph computation and exploring potential avenues for future research. Our survey focuses on LLMs for graph computation and explicitly provides definitions of related tasks, a systematic taxonomy of existing methods, and a collection of benchmark datasets within the scope. The contributions can be summarized as follows:

- *Role-based taxonomy.* We introduce an operational role-based taxonomy specific to LLM-based graph computation. Our taxonomy organizes existing studies according to the roles that LLMs play in graph computational tasks, with a key focus on distinguishing whether graph operations are executed internally by LLMs or require support from external tools or algorithms. Building upon this foundation, we further sub-categorize studies based on the specific techniques employed within each paradigm. This taxonomy allows us to compare methods by their reliability, scalability, and applicability to different scenarios.
- *Comprehensive review and summary.* Based on the proposed taxonomy, we review the current literature, analyzing the strengths and limitations of each method. We also summarize the definitions and time complexity of representative graph computational tasks in Appendix B. In Appendix C, we assemble an extensive set of datasets for evaluating LLMs on graph computation.
- *Evidence-based assessment and future directions.* Based on existing evidence, we argue that LLMs are promising as flexible interfaces for simple tasks involving small graphs, but are limited for large-scale graphs and correctness-critical graph tasks. Based on this analysis, we further propose four future directions, including developing semantic graph benchmarks, optimizing multi-step execution for complex graph queries, preserving the privacy of graphs in prompts and training data, and adapting LLMs to domain-specific graph structures.

2 Background

In this section, we first provide essential background knowledge on graph computational tasks and LLMs. Then, we present our taxonomy of LLMs for graph computation.

2.1 Definitions

Graph-structured data. A graph $G = (V, E)$ is a non-linear data structure that consists of a set of vertices V and a set of edges E connecting these vertices. Each edge $e \in E$ is associated with a pair of vertices (u, v), where u and v are the endpoints of the edge. Depending on the specific information carried by their vertices and edges, graphs can be classified into various types. For example, directed graphs have directional edges $e = (u, v)$ pointing from u to v, and edge-weighted graphs have a weight $w(e) \in \mathbb{R}$ for each edge.

Graph description. As LLMs rely on a sequential text interface, describing the graph structure textually is the fundamental step for enabling LLM-based graph computation. List structures (Tabassum et al., 2018), such as edge list and adjacency list, are commonly considered as graph representation methods due to their space efficiency, which helps minimize token consumption when encoding graph data for LLMs.

- The adjacency list $\mathcal{L}$ of a graph G with n vertices is a collection of n lists, where the list l_u for each vertex u contains all vertices v such that there is an edge from u to v, i.e., $l_u = \{v \mid (u, v) \in E\}$.
- The edge list $\mathcal{E}$ of a graph G is a list of all edges in the graph. Each edge is represented as a pair (u, v), where u and v are the endpoints of the edge, i.e., $\mathcal{E} = \{ (u, v) \mid (u, v) \in E \}$.

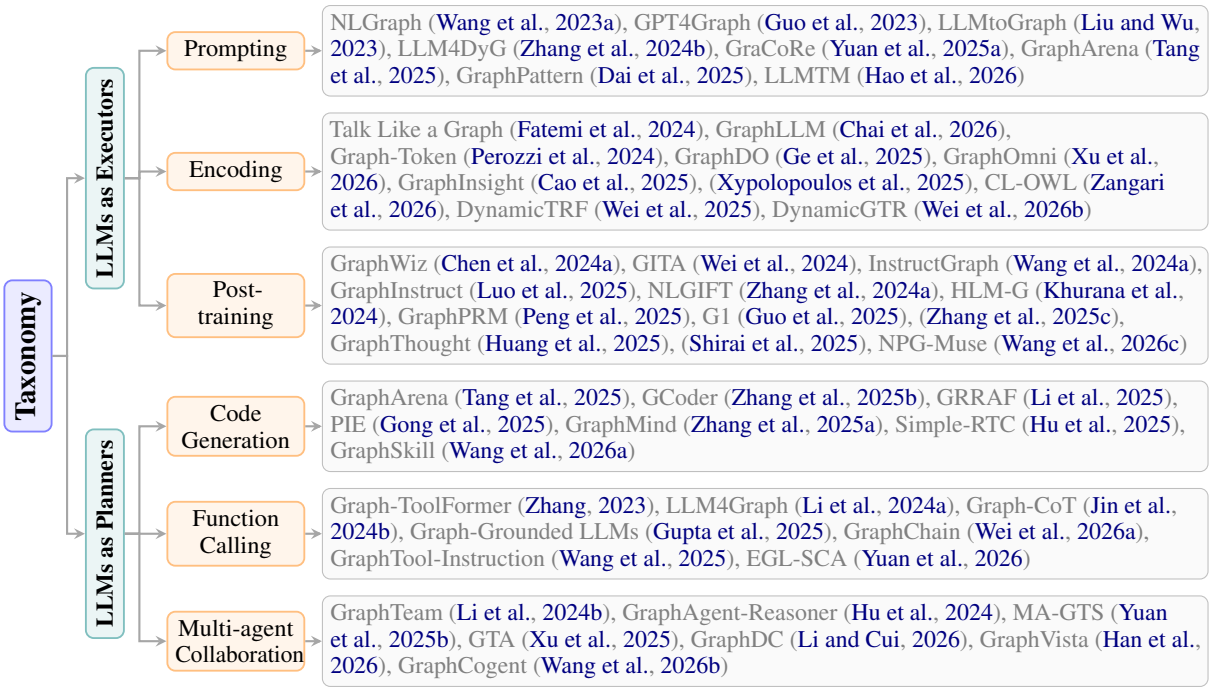

Figure 2: Taxonomy of LLMs for graph computation.

Besides list-based representations, matrix-based representations, such as adjacency matrix (Harary, 1962) and incidence matrix (Fulkerson and Gross, 1965), and graph description languages such as Graph Modeling Language (Himsolt, 1997), and Graph Markup Language (Brandes et al., 2013) are also widely considered to describe graphs.

Graph computational tasks. Graph computational tasks are problems defined on graphs with the objective of discovering relational patterns, extracting structural properties, or computing mathematical metrics of a given graph. Unlike graph learning tasks, e.g., node classification and link prediction, which produce learned predictions by learning predictive models or latent representations from graph data, graph computational tasks are solved through algorithmic operations on graph structures and produce deterministic results. Formally, given a graph $G = (V, E)$, a graph computation task $\mathcal{T}$ refers to a problem that requires operations on the vertices and edges of G to produce the answer $\mathcal{A}$. We have summarized the common tasks in Appendix B.

Large language models. LLMs are language models with billions of parameters that are pre-trained on massive text corpora and designed to understand and generate content from text distributions (Zhao et al., 2026; Li et al., 2026). LLMs can be easily and efficiently fine-tuned on task-specific data to achieve better results across diverse downstream tasks. Beyond standard statistical text generation, LLMs can act as the central reasoning engine and be augmented with dynamic memory and external computational tools to function as autonomous agents. By giving the model access to external tools and memory, LLMs have demonstrated impressive capabilities in planning multi-step strategies and solving complex questions (Zhao et al., 2024; Mohammadi et al., 2025).

LLMs for graph computation. LLMs for graph computation refer to methods that involve LLMs in the process of solving graph computational tasks. Given the textual description of graph $G = (V, E)$ and a graph computation task $\mathcal{T}$ as the input, LLMs leverage their pre-trained knowledge, reasoning capabilities, or the provided external tools to produce the corresponding answer $\mathcal{A}$.

2.2 Taxonomy

In this survey, we organize existing works by the role of LLMs in graph computation. As shown in Fig. 2, we summarize two main paradigms: LLMs as Executors and LLMs as Planners.

- **LLMs as executors.** LLMs directly solve graph problems by taking graph and task descriptions as input, relying on their inherent reasoning abilities to perform algorithmic computation internally and output answers without invoking external executable components. Existing executor-based methods mainly include three types: i) *Prompting*, which uses various prompt designs to describe

tasks; ii) *Encoding*, which focuses on representing graph structures to LLMs via natural language or specialized encoders; and iii) *Post-training*, which adapts LLMs to graph computation through fine-tuning on domain-specific graph datasets.

• **LLMs as planners.** LLMs serve as high-level orchestrators for graph computational tasks by interpreting the task, decomposing the problem, and coordinating external resources, such as external tools, generated programs, or specialized agents, to solve the problem. This includes three primary approaches: i) *Code generation*, where LLMs generate executable code to solve graph computational tasks; ii) *Function calling*, where LLMs invoke predefined graph APIs or specialized tools to obtain a result; and iii) *Multi-agent collaboration*, where multiple LLM-based agents cooperate to solve graph computational tasks.

In the following sections, we review representative studies under each category and discuss their strengths and limitations.

3 LLMs as Executors

In this section, we focus on the paradigm of LLMs as executors. In this setting, LLMs are given a graph structure and a task instruction, perform reasoning internally, and directly output the final answer without relying on external tools. We comprehensively review the subcategories and discuss their advantages and limitations.

3.1 Prompting

To equip LLMs for graph computation without modifying their internal parameters, a robust line of research investigates advanced prompting techniques. This approach leverages the models' inherent reasoning capabilities and pre-trained knowledge to interpret graph tasks strictly through natural language instructions.

Early studies investigate how to adapt LLMs to graph computation. GPT4Graph (Guo et al., 2023) is among the first works to introduce a pipeline to integrate LLMs into graph computation. Beyond general prompting strategies such as zero-shot prompting (Kojima et al., 2022), few-shot prompting (Brown et al., 2020), chain-of-thought (Wei et al., 2022), least-to-most prompting (Zhou et al., 2023), and self-consistency (Wang et al., 2023b), Build-a-Graph Prompting and Algorithmic Prompting were proposed by Wang et al. (2023a), which leverage instruction-based strategies to further enhance LLM capability.

Recent studies have improved prompting methods to enable LLMs to handle a broader range of graph types and tasks. LLM4DyG (Zhang et al., 2024b) and LLMTM (Hao et al., 2026) evaluate dynamic graphs where each edge is associated with a timestamp t to capture the temporal evolution of a network. GraCoRe (Yuan et al., 2025a) evaluates heterogeneous graphs that include rich, multi-typed semantic information. To push the boundaries of task diversity, recent studies have investigated whether LLMs can reliably enumerate all valid solutions to multi-answer queries, such as finding all shortest paths (Liu and Wu, 2023). GraphArena (Tang et al., 2025) examines LLMs on solving polynomial-time tasks alongside NP-complete tasks on real-world datasets. Furthermore, the work (Dai et al., 2025) focuses on tasks related to mining graph patterns, such as triangle, square, diamond, and house.

3.2 Encoding

Encoding is a crucial factor that can significantly affect LLMs' performance on graph computation, as it serves as an important preprocessing technique for representing graph structures in sequence form.

A major line of work improves LLM performance by designing more effective textual representations of graphs. Talk Like a Graph (Fatemi et al., 2024) experiments with different graph representations. GraphDO (Ge et al., 2025) suggests that the optimal ordering strategy is closely tied to the inherent structural requirements of the target task. To help LLMs better leverage the graph structures, GraphInsight (Cao et al., 2025) reorganizes the most critical subgraph descriptions to the head and tail of the text sequence, where LLMs exhibit stronger memory performance. Xypolopoulos et al. (2025) propose graph linearization methods that renumber vertices according to structural properties such as centrality and degeneracy, and CL-OWL (Zangari et al., 2026) introduces a human-interpretable structural encoding scheme that uses colors to represent graphs. In addition, GraphOmni (Xu et al., 2026) proposes an RL-based adaptive mechanism to select the optimal combination of graph description and prompt.

Beyond purely textual graph representations, another line of work injects graph structural knowledge into LLMs through specialized encoders. GraphLLM (Chai et al., 2026) employs a graph transformer to encode the graph structure into continuous vector embeddings, which are then used as

prefix embeddings for LLMs. Graph-Token (Perozzi et al., 2024) explores various GNN architectures to convert the graph into soft tokens that are fed into the LLM. Beyond embedding-based fusion, DynamicTRF (Wei et al., 2025) and DynamicGTR (Wei et al., 2026b) propose frameworks that dynamically select the most suitable textual or visual graph representation, optimizing both answer accuracy and response efficiency.

3.3 Post-training

Post-training methods adapt pre-trained LLMs for graph computation by optimizing them on graph-specific data. Compared with prompting and encoding, which keep model parameters fixed, post-training modifies the model to improve task-specific accuracy (Sanford et al., 2024).

To improve the generalization capabilities of the fine-tuned model, some studies adopt instruction tuning. GraphInstruct (Luo et al., 2025) introduces a comprehensive benchmark covering 21 diverse tasks and proposes a label mask training to help LLMs learn node ID information during fine-tuning. NLGIFT (Zhang et al., 2024a) shows that fine-tuned LLMs may still rely on memorized graph patterns rather than general graph reasoning rules. To improve reasoning supervision, Graph-Thought (Huang et al., 2025) generates training trajectories through heuristic-guided forward reasoning and solver-guided backward reasoning. Less is More (Shirai et al., 2025) further shows that LLMs instruction-tuned on textual input can outperform models with specialized graph encoders.

Several studies improve reliability and reduce hallucinations via Direct Preference Optimization (DPO), which aligns models with preference pairs, and Reinforcement Learning (RL), which optimizes via reward feedback. By using DPO, GraphWiz (Chen et al., 2024a) further enhances the performance of instruction-tuned LLMs to align the model with the correct reasoning path. To build high-quality preference pairs for DPO, InstructGraph (Wang et al., 2024a) constructs hallucination-based negative examples, and Graph-PRM (Peng et al., 2025) uses Process Reward Models to score intermediate reasoning steps. Zhang et al. (2025c) show that post-training alignment on synthetic graph data can partially transfer to real-world tasks with implicit graph structures. By adopting RL, G1 (Guo et al., 2025) enables LLMs to self-improve their graph reasoning abilities through rule-based verifiable rewards, and NPG-Muse (Wang et al., 2026c) introduces NP-hard graph problems as a training corpus.

To enable LLMs on large graphs, several studies have explored methods to reduce the computational cost of the prompts, which can become lengthy when incorporating both feature and structural information from graphs. HLM-G (Khurana et al., 2024) uses bespoke attention masks to force the token processing to remain localized within descriptions of single vertices, reducing the computational complexity from $\mathcal{O}\left((\sum n_i)^2\right)$ to $\mathcal{O}\left(\sum {n_i}^2\right)$, where n_i is the number of tokens describing vertex v_i. GITA (Wei et al., 2024) integrates fine-tuned vision-language models into LLM-based graph computation and adopts k-hop subgraph sampling, where the subgraph for vertex u is induced by all vertices within k steps from u.

3.4 Summary

Methods reviewed in this section improve the ability of LLMs to directly execute graph computation. Prompting-based methods improve model performance by encouraging models to think or follow intermediate computational steps or given examples. Encoding-based methods improve accuracy by presenting graph structures that are easier for LLMs to parse and reason over. Post-training-based methods improve accuracy by directly optimizing LLMs on graph computation data. Compared with prompting and encoding, post-training can generally produce better task-specific performance.

Although all three lines of work aim to make LLMs better graph executors, each category has its own limitations. For prompting, designing effective prompts, such as few-shot and CoT, requires human effort and graph algorithm knowledge. Different tasks also require task-specific prompts, limiting the flexibility of prompting methods across diverse graph tasks. For encoding, when the graph description becomes too long, models tend to forget the middle section (Liu et al., 2024b; Hsieh et al., 2024). Although placing the critical subgraph at the head and tail can improve the performance, it introduces additional cost to design the order. Post-training methods are restricted by their dependence on high-quality training datasets. Constructing such datasets is costly, as they typically necessitate broad coverage over diverse graph tasks and graph types. Moreover, LLMs often exhibit performance degradation on tasks or graphs that are not in the training data (Chen et al., 2024a).

4 LLMs as Planners

Although using LLMs as executors offers convenience and accessibility for general users, LLMs are probabilistic sequence models, which are unreliable when used as deterministic symbolic executors with guaranteed mathematical or algorithmic correctness (Hahn, 2020; Dziri et al., 2023). The planner paradigm shifts LLMs away from direct graph solving to external support such as programs, tools, or specialized agents, so that hallucination, omitted steps, and contextual overflow can be effectively avoided as the graph size and reasoning depth increase (Liu et al., 2024b; Heyman and Zylberberg, 2025; Wang et al., 2024b; Han et al., 2026). We review this paradigm through code generation, function calling, and multi-agent collaboration.

4.1 Code Generation

LLMs have demonstrated strong natural-language-to-code generation ability, which shifts the uncertainty of token-level reasoning into a more reliable computational process (Gao et al., 2023; Chen et al., 2023, 2021; Li et al., 2022).

GraphArena (Tang et al., 2025) first mentions code writing for graph computation and shows that executable solving can reduce errors caused by the repetitive iterative and backtracking operations required by many graph algorithms. Following this direction, GCoder (Zhang et al., 2025b) collects code-oriented instruction-tuning data for graph computational problems and improves the LLM's graph-solving ability through supervised fine-tuning on code generation. GraphMind (Zhang et al., 2025a) introduces a dataset featuring code-based solutions and execution traces, guiding LLMs to tackle graph problems programmatically. Simple-RTC (Hu et al., 2025) separates problem formatting, data extraction, algorithm reasoning, and code generation into a pipeline.

Beyond simply generating code, several studies explore how LLMs can detect and handle execution errors during graph computation. PIE (Gong et al., 2025) injects task-related pseudocode into prompts and uses repeated trial-and-error execution, improving accuracy while keeping computational cost under control. GRRAF (Li et al., 2025) provides a training-free variant by storing the graph in a graph database and prompting the LLM to generate executable NetworkX or Cypher queries, with error feedback and timeout control to keep graph access outside the model context. GraphSkill (Wang et al., 2026a) extends this code-based direction by combining hierarchical retrieval over technical documentation with self-generated small test cases, allowing the coding agent to reduce noisy retrieval and address logical errors.

4.2 Function Calling

Function calling provides a more stable and constrained planner interface for graph computation by allowing LLMs to select predefined graph APIs or tools from existing graph toolkits or user-defined libraries (Schick et al., 2023; Patil et al., 2024; Li et al., 2023; Patil et al., 2025).

Some studies explore how LLMs can invoke predefined graph tools. For instance, Graph-ToolFormer (Zhang, 2023) teaches the model to insert graph loading and graph analysis calls into generated statements. LLM4Graph (Li et al., 2024a) proposes a pipeline based on API documents of frequently used graph libraries, enabling LLMs to leverage both document and code to analyze larger-scale graphs. GraphTool-Instruction (Wang et al., 2025) provides a more effective tool-using pipeline by explicitly decomposing graph reasoning into graph extraction, tool-name identification, and tool-parameter extraction.

Recent work extends function calling to iterative settings. Graph-CoT (Jin et al., 2024b) introduces an iterative reasoning–interaction–execution framework for stepwise graph computation, allowing LLMs to focus on the information required at each step. Graph-Grounded LLMs (Gupta et al., 2025) connect LLMs with a graph library through closed-loop function calls to reduce hallucinations and mathematical inaccuracies. GraphChain (Wei et al., 2026a) enables large-scale graph computation by letting LLMs compose graph tool-chains and compresses task-relevant graph information step by step. EGL-SCA (Yuan et al., 2026) improves graph tool-using agents by employing structural credit assignment, which routes failures to either instruction updates or tool repairs.

4.3 Multi-agent Collaboration

Multi-agent collaboration provides a robust solution for complex graph problems by distributing graph-solving responsibilities across specialized agents. In this way, it alleviates the single-planner brittleness of the long reasoning step error accumulation as the graph scale and difficulty increase.

Some studies follow a conventional multi-agent workflow design by assigning each agent a role in

the graph-solving pipeline. GraphTeam (Li et al., 2024b) organizes graph solving into cooperating agents with different specialties, converting graph analysis from one-shot reasoning into a staged process. MA-GTS (Yuan et al., 2025b) follows a similar role-specialized logic and extends to real-world graph computation problems where graph data may be implicit, noisy, and irregular.

To further improve scalability, some recent works decompose the graphs across multiple agents. GraphAgent-Reasoner (Hu et al., 2024) allocates a dedicated agent to process the information of a single vertex, employing a master LLM to organize the distributed computation and merge the answers. However, this method introduces computation overhead in distributed computations. GraphDC (Li and Cui, 2026) optimizes the computational cost by decomposing the graph into subgraphs rather than individual vertices, and letting the master agent process inter-subgraph edges. To address memory and scalability constraints, GraphCogent (Wang et al., 2026b) employs a sensory agent to partition large complex graphs into manageable overlapping subgraphs via a sliding window, which are subsequently merged and indexed by a buffer agent.

In addition, some studies decompose complex graph tasks into sub-tasks. GTA (Xu et al., 2025) introduces a fine-tuned algorithm generator and algorithm decomposer to break down the graph tasks into manageable sub-steps. GraphVista (Han et al., 2026) introduces a planning agent that decomposes graph tasks and adaptively assigns them to textual retrieval or visual reasoning modules based on task requirements and structural dependencies.

4.4 Summary

Methods reviewed in this section improve graph computation by shifting LLMs from direct executors to planners that coordinate external computational resources. Code generation provides the most flexible planner interface, allowing LLMs to translate graph problems into executable programs and substantially improve accuracy and scalability by delegating graph-state tracking and algorithm execution to interpreters. Function calling enables LLMs to invoke predefined external tools, allowing complex graph operations to be performed more reliably and efficiently, especially when the task matches an existing tool schema. By assigning different agents to specialized roles, multi-agent collaboration has the flexibility of code generation with the reliability of function calling, enabling the planner to handle more complex graph problems.

However, each category also introduces distinct limitations. Code generation is prone to syntax errors, runtime bugs, API misuse, and input-format mismatches during the translation from a graph problem to executable code. Function calling is more constrained by tool coverage and compositionality. When it encounters an out-of-distribution operation, it may force the problem into the closest available function, causing invalid calls. Multi-agent collaboration suffers from computational overhead compared to the previous methods, as it requires intermediate message exchanges across multiple interaction turns (Tran and Kiela, 2026). Also, the intermediate errors may propagate across agents, leading to repeated and invalid communication, latency, and tool invocation.

Since we classify papers according to the main contributions, some planner-based methods also contain executor components, where the LLM or an agent performs intermediate computation after planning. GraphAgent-Reasoner (Hu et al., 2024) uses a master agent to arrange node agents to aggregate the answer, where each node agent serves as an executor to compute the local information. GraphDC (Li and Cui, 2026) splits the graph tasks to each specialized agent, where the specialized agent processes the assigned subgraph.

5 Discussions and Future Directions

In this section, we discuss whether LLMs are suitable for graph computation tasks based on existing literature and outline promising future directions.

5.1 Discussions

Table 1 shows the suitable scenarios, strengths, and limitations of each method. Overall, LLMs are more effective as planners that interpret, decompose, and delegate tasks to reliable tools or code, while as standalone executors, they are suitable only for small graphs and for scenarios where accuracy requirements are not stringent.

Despite these promising capabilities, several challenges remain that limit a more reliable adoption of LLMs for graph computation.

Scalability. Most executor-based methods achieve reasonable performance only on small graphs of up to 50 nodes (Cao et al., 2025). As the performance often drops when relevant information appears in the middle of a long context (Hu et al., 2024; Liu et al., 2024b), LLMs struggle to faithfully memorize the graph structures when textual descriptions

Table 1: Summary of LLM-based graph computation methods: suitable scenarios, strengths, and limitations.

Category	Suitable Scenarios	Strengths	Limitations
Prompting	Small graphs, simple tasks, or scenarios requiring minimal setup	No need for training/tools, some effective prompt techniques (e.g., Build-a-Graph, Algorithmic Prompting) are easy to use	Accuracy drops as the size of input graph grows, sensitive to prompt design, some techniques (e.g., CoT) require expertise
Encoding	Tasks where graph description affects performance	Straightforward way to improve structural understanding via serialization or embeddings	Accuracy drops as size of input graph grows, optimal encoding method is often task-specific
Post-training	Task type within the training dataset	Better task-specific performance than prompting and encoding	Requires high-quality training data, weaker generalization to unseen tasks
Code-generation	Large graph computation with an execution environment	Delegates computation to executable programs, scalable and flexible	May suffer from syntax errors and runtime bugs, requires an execution environment
Function-calling	Problems solvable via standard graph operations and supported graph types in the chosen library	Reliable, enabling the use of well-optimized and robust libraries	Constrained by the coverage of existing APIs/tools, API misuse
Multi-agent Collaboration	Complex, large-scale, multi-step, or distributed graph computation, enabling flexible integration of the techniques above	Can decompose and coordinately solve the tasks	Higher coordination and latency costs, possible error propagation between agents

become longer. Furthermore, the limited context windows of LLMs prevent them from processing graphs whose descriptions exceed the maximum context length. Planner-based systems suffer from contextual and inference overhead caused by operations like compiler feedback, API documents, compressed graph states, or inter-agent communication (Li et al., 2024a; Gong et al., 2025; Wei et al., 2026a; Hu et al., 2024). These operations consume considerable tokens and computation time, limiting the scalability to larger graphs.

Generalization. Executor-based methods tend to memorize graph patterns and training tasks rather than acquiring generalizable graph reasoning principles (Zhang et al., 2025c). Their performance may degrade significantly when evaluated on unseen graphs and tasks, indicating a reliance on superficial input patterns over true algorithmic understanding (Zhang et al., 2024a). Additionally, LLM performance is highly sensitive to text and structural representations. Different graph tasks and structures are better suited to specific prompting and encoding methods, necessitating careful optimization of these inputs (Fatemi et al., 2024; Xu et al., 2026). Some planner-based methods heavily depend on unified graph description methods, task formulations, tool interfaces, and execution environments. Such requirements limit their generalization ability in real-world scenarios, where tasks, available tools, and environment constraints may vary significantly across domains.

5.2 Future Directions

Apart from the common objectives of expanding graph scale, tackling harder tasks, improving accuracy, and minimizing token consumption, several other promising directions remain unexplored.

• *Benchmarks for queries with implicit graph structures.* Current benchmarks for LLM-based graph computation mostly provide explicit graph structures. In many scenarios, graphs are often implicit in queries or documents, such as transaction logs and medical records (Wang et al., 2024b; Hu et al., 2023), while LLMs struggle to construct graphs from such unstructured text and often include unrealistic biases in the graph structure (Chang et al., 2025). In such cases, LLMs need to recognize and identify the underlying graph structure (Hsieh et al., 2024; Bai et al., 2025). Moreover, selecting the correct computational procedure is also challenging for LLMs, as subtle differences in task descriptions or graph structures may require different algorithms. For example, numerical values associated with edges may represent either timestamps or weights, leading to entirely different graph computation tasks. To this end, future benchmarks should evaluate LLM-based graph computation with implicit graph structures, bringing LLM-based graph computation closer to real-world scenarios.

• *Optimizing execution pipelines for complex graph queries.* Complex graph queries often require multi-step operations, whose execution order strongly affects computation cost and accu-

racy (Wei et al., 2026a). Refining task-relevant intermediate subgraphs before complex steps improves efficiency and precision (Lyu et al., 2025). Without such optimization, executor-based methods suffer from inflated token consumption and diminished accuracy, while planner-based methods experience longer runtimes when large subgraphs are fed into high-complexity operations. Therefore, future work should enable LLMs to decompose queries into optimized sub-task sequences, thereby enabling more scalable, efficient, and robust multi-step graph computation.

• *Privacy vulnerabilities.* LLMs have been proven vulnerable to many attacks (Chen et al., 2025; Yao et al., 2024; Yu et al., 2025), while real-world graphs capture sensitive user information or relationships (Zhang et al., 2025d; Zhu et al., 2026). For example, in e-commerce networks, vertices may include private attributes like home address, and edges reveal purchase or transaction records. In healthcare systems, the graph not only stores patient data in its vertices, but also exposes patient–doctor associations through edges. Through jailbreaking attacks, the graphs in the input prompt can be exposed to unauthorized users (Priyanshu et al., 2023; Tang et al., 2024b). In addition, membership inference attacks may reveal whether specific vertices or edges appear in the training corpus (Das et al., 2025). It is essential to leverage LLMs for graph computation while ensuring data privacy.

• *LLMs for domain-specific graph computation.* Many real-world applications involve domain-specific graphs beyond conventional structures, with features such as rich attributes, heterogeneity, and temporal dependencies. For example, in a global metabolic network (Peregrín-Alvarez et al., 2009; Barabási et al., 2011), vertices encode enzymes, with colors denoting metabolic superclasses and sizes indicating associated genome counts. In an attack graph (Ou et al., 2006; Kaynar and Sivrikaya, 2016), attributes of vertices indicate types, security conditions, vulnerabilities, and exploitability scores, while the direction of edges captures logical dependencies and attacker movements. However, due to the limited generalizability of current LLMs (Lu et al., 2025), directly applying LLMs designed for general graph computation to these highly specialized graphs often results in low accuracy and severe hallucinations. Enabling LLMs to handle richer graph structures could improve their utility in complex real-world scenarios.

6 Conclusions

We systematically reviewed the status quo of LLMs for graph computation. Specifically, we introduced a new taxonomy based on the operational roles of models and provided a thorough review and comparison of methods within each paradigm. We summarized representative graph computation tasks and available benchmark datasets. Our review suggests that while LLMs perform well on small-scale graphs and simple tasks, they still exhibit limitations when faced with large-scale or high-precision demands, indicating opportunities for future research and improvement.

Limitations

As a survey paper, its limitations reflect the limitations of the LLM-based graph computation area. Although recent studies have made substantial progress in enabling LLMs to accurately solve graph computation tasks, LLMs still face substantial challenges when applied to graph-structured problems, including limited scalability, weak generalization, and unreliable reasoning over complex structures. In addition, many important graph problems have not been sufficiently covered by existing studies, such as community search and other advanced graph query tasks.

Due to space constraints in the main text, detailed summaries of representative graph computation tasks and benchmark datasets are provided in the appendices. The task summary provides definitions of common graph computation problems and their computational complexities. The benchmark overview reports dataset statistics, including graph size, graph type, and task type. We believe these materials are important and useful for readers.

References

Ariful Azad, Aydin Buluç, and John Gilbert. 2015. Parallel triangle counting and enumeration using matrix algebra. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 804–811. IEEE.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2025. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, Vienna, Austria. Association for Computational Linguistics.

Albert-László Barabási, Natali Gulbahce, and Joseph Loscalzo. 2011. Network medicine: a network-based approach to human disease. *Nature Reviews Genetics*, 12(1):56–68.

Ulrik Brandes, Markus Eiglsperger, Jürgen Lerner, and Christian Pich. 2013. Graph markup language (GraphML). In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, pages 517–541. Chapman and Hall/CRC.

Adam Breuer, Roee Eilat, and Udi Weinsberg. 2020. Friend or faux: Graph-based early detection of fake accounts on social networks. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 1287–1297. ACM / IW3C2.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Yukun Cao, Shuo Han, Zengyi Gao, Zezhong Ding, Xike Xie, and S Kevin Zhou. 2025. GraphInsight: Unlocking insights in large language models for graph structure understanding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12096–12134, Vienna, Austria. Association for Computational Linguistics.

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. 2026. Graphllm: Boosting graph reasoning ability of large language model. *IEEE Transactions on Big Data*, 12(2):475–483.

Serina Chang, Alicja Chaszczewicz, Emma Wang, Maya Josifovska, Emma Pierson, and Jure Leskovec. 2025. Llms generate structurally realistic social networks but overestimate political homophily. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 19, pages 341–371.

Kang Chen, Xiuze Zhou, Yuanguo Lin, Shibo Feng, Li Shen, and Pengcheng Wu. 2025. A survey on privacy risks and protection in large language models. *Journal of King Saud University Computer and Information Sciences*, 37(7):163. Article 163.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. 2024a. Graphwiz: An instruction-following language model for graph computational problems. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 353–364, New York, NY, USA. Association for Computing Machinery.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Preprint*, arXiv:2211.12588.

Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, and 1 others. 2024b. Exploring the potential of large language models (llms) in learning on graphs. *ACM SIGKDD Explorations Newsletter*, 25(2):42–61.

Xinnan Dai, Haohao Qu, Yifei Shen, Bohang Zhang, Qihao Wen, Wenqi Fan, Dongsheng Li, Jiliang Tang, and Caihua Shan. 2025. How do large language models understand graph patterns? a benchmark for graph pattern comprehension. In *International Conference on Learning Representations*, volume 2025, pages 58405–58435.

Badhan Chandra Das, M. Hadi Amini, and Yanzhao Wu. 2025. Security and privacy challenges of large language models: A survey. *ACM Comput. Surv.*, 57(6).

Sujatha Das Gollapalli and Cornelia Caragea. 2014. Extracting keyphrases from research papers using citation networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1).

E. W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang (Lorraine) Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, volume 36, pages 70293–70332. Curran Associates, Inc.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2024. Talk like a graph: Encoding graphs for large language models. In *International Conference on Learning Representations*, volume 2024, pages 43909–43934.

Delbert Fulkerson and Oliver Gross. 1965. Incidence matrices and interval graphs. *Pacific journal of mathematics*, 15(3):835–855.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *Proceedings of the 40th International*

Conference on Machine Learning, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

Yuyao Ge, Shenghua Liu, Baolong Bi, Yiwei Wang, Lingrui Mei, Wenjie Feng, Lizhe Chen, and Xueqi Cheng. 2025. Can graph descriptive order affect solving graph problems with LLMs? In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6404–6420, Vienna, Austria. Association for Computational Linguistics.

Chang Gong, Wanrui Bian, Zhijie Zhang, and Weiguo Zheng. 2025. Pseudocode-injection magic: Enabling llms to tackle graph computational tasks. *arXiv preprint arXiv:2501.13731*.

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

Xiaojun Guo, Ang Li, Yifei Wang, Stefanie Jegelka, and Yisen Wang. 2025. G1: Teaching llms to reason on graphs with reinforcement learning. *Preprint*, arXiv:2505.18499.

Piyush Gupta, Sangjae Bae, and David Isele. 2025. Graph-grounded llms: Leveraging graphical function calling to minimize llm hallucinations. *arXiv preprint arXiv:2503.10941*.

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A. Rossi, Subhabrata Mukherjee, Xianfeng Tang, Qi He, Zhigang Hua, Bo Long, Tong Zhao, Neil Shah, Amin Javari, Yinglong Xia, and Jiliang Tang. 2025. Retrieval-augmented generation with graphs (graphrag). *Preprint*, arXiv:2501.00309.

Shuo Han, Yukun Cao, Zezhong Ding, Zengyi Gao, S Kevin Zhou, and Xike Xie. 2026. See or say graphs: Agent-driven scalable graph structure understanding with vision-language models. *Preprint*, arXiv:2510.16769.

Bing Hao, Minglai Shao, Zengyi Wo, Yunlong Chu, Yuhang Liu, and Ruijie Wang. 2026. Llmtm: Benchmarking and optimizing llms for temporal motif analysis in dynamic graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(17):14802–14810.

Frank Harary. 1962. The determinant of the adjacency matrix of a graph. *Siam Review*, 4(3):202–210.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182.

Alex Heyman and Joel Zylberberg. 2025. Reasoning large language model errors arise from hallucinating critical problem features. *Preprint*, arXiv:2505.12151.

Michael Himsolt. 1997. Gml: Graph modelling language. *University of Passau*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *Preprint*, arXiv:2404.06654.

Yuntong Hu, Zheng Zhang, and Liang Zhao. 2023. Beyond text: A deep dive into large language models' ability on understanding graph data. *Preprint*, arXiv:2310.04944.

Yuwei Hu, Xinyi Huang, Zhewei Wei, Yongchao Liu, and Chuntao Hong. 2025. Rethinking and benchmarking large language models for graph reasoning. *Preprint*, arXiv:2509.24260.

Yuwei Hu, Runlin Lei, Xinyi Huang, Zhewei Wei, and Yongchao Liu. 2024. Scalable and accurate graph reasoning with llm-based multi-agents. *arXiv preprint arXiv:2410.05130*.

Zixiao Huang, Lifeng Guo, Wenhao Li, Junjie Sheng, Chuyun Shen, Haosheng Chen, Bo Jin, Changhong Lu, and Xiangfeng Wang. 2025. Graphthought: Graph combinatorial optimization with thought generation. *Preprint*, arXiv:2502.11607.

Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. 2024a. Large language models on graphs: A comprehensive survey. *IEEE Transactions on Knowledge and Data Engineering*, 36(12):8622–8642.

Bowen Jin, Chulin Xie, Jiawei Zhang, Kashob Kumar Roy, Yu Zhang, Zheng Li, Ruirui Li, Xianfeng Tang, Suhang Wang, Yu Meng, and Jiawei Han. 2024b. Graph chain-of-thought: Augmenting large language models by reasoning on graphs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 163–184, Bangkok, Thailand. Association for Computational Linguistics.

Kerem Kaynar and Fikret Sivrikaya. 2016. Distributed attack graph generation. *IEEE Transactions on Dependable and Secure Computing*, 13(5):519–532.

Sambhav Khurana, Xiner Li, Shurui Gui, and Shuiwang Ji. 2024. A hierarchical language model for interpretable graph reasoning. *arXiv preprint arXiv:2410.22372*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Hanqing Li, Sharika Mahadevan, Kiran Jyothi Sheena, Henry Liang, and Diego Klabjan. 2025. Zero-shot graph reasoning via retrieval augmented framework with LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 17041–17054, Suzhou, China. Association for Computational Linguistics.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Na Li, Yuchen Jiao, Changxiao Cai, and Gen Li. 2026. Convergeflow: Language flow with provable convergence to token embeddings. *Preprint*, arXiv:2608.23551.

Wenjin Li and Jiaming Cui. 2026. Graphdc: A divide-and-conquer multi-agent system for scalable graph algorithm reasoning. *Preprint*, arXiv:2605.06671.

Xin Li, Weize Chen, Qizhi Chu, Haopeng Li, Zhaojun Sun, Ran Li, Chen Qian, Yiwei Wei, Zhiyuan Liu, Chuan Shi, and 1 others. 2024a. Can large language models analyze graphs like professionals? a benchmark, datasets and models. *Advances in Neural Information Processing Systems*, 37:141045–141070.

Xin Li, Qizhi Chu, Yubin Chen, Yang Liu, Yaoqi Liu, Zekai Yu, Weize Chen, Chen Qian, Chuan Shi, and Cheng Yang. 2024b. Graphteam: Facilitating large language model-based graph analysis via multi-agent collaboration. *arXiv preprint arXiv:2410.18032*.

Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. 2024c. A survey of graph meets large language model: Progress and future directions. *Preprint*, arXiv:2311.12399.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Chang Liu and Bo Wu. 2023. Evaluating large language models on graphs: Performance insights and comparative analysis. *arXiv preprint arXiv:2308.11224*.

Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2024a. One for all: Towards training one graph model for all classification tasks. In *International Conference on Learning Representations*, volume 2024, pages 20188–20210.

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

Wei Lu, Rachel K. Luu, and Markus J. Buehler. 2025. Fine-tuning large language models for domain adaptation: exploration of training strategies, scaling, model merging and synergistic capabilities. *npj Computational Materials*, 11(1):84.

Zihan Luo, Xiran Song, Hong Huang, Jianxun Lian, Chenhao Zhang, Jinqi Jiang, Xing Xie, and Hai Jin. 2025. Graphinstruct: Empowering large language models with graph understanding and reasoning capability. *Preprint*, arXiv:2403.04483.

B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou. 2020. Maximum biclique search at billion scale. *Proceedings of the VLDB Endowment*, 13:1359–1372.

Bingqing Lyu, Xiaoli Zhou, Longbin Lai, Yufan Yang, Yunkai Lou, Wenyuan Yu, Ying Zhang, and Jingren Zhou. 2025. A modular graph-native query optimization framework. In *Companion of the 2025 International Conference on Management of Data*, SIGMOD/PODS '25, page 566–579, New York, NY, USA. Association for Computing Machinery.

Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. 2025. Evaluation and benchmarking of llm agents: A survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 6129–6139, New York, NY, USA. Association for Computing Machinery.

Mark EJ Newman. 2003. The structure and function of complex networks. *SIAM review*, 45(2):167–256.

Behnam Neyshabur, Ahmadreza Khadem, Somaye Hashemifar, and Seyed Shahriar Arab. 2013. Netal: A new graph-based method for global alignment of protein-protein interaction networks. *Bioinformatics (Oxford, England)*, 29.

Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. 2006. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 336–345, New York, NY, USA. Association for Computing Machinery.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565. Curran Associates, Inc.

Miao Peng, Nuo Chen, Zongrui Suo, and Jia Li. 2025. Rewarding graph reasoning process makes llms more generalized reasoners. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 2257–2268, New York, NY, USA. Association for Computing Machinery.

José M. Peregrín-Alvarez, Chris Sanford, and John Parkinson. 2009. The conservation and evolutionary modularity of metabolism. *Genome Biology*, 10(6):R63.

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. *Preprint*, arXiv:2402.05862.

Aman Priyanshu, Supriti Vijay, Ayush Kumar, Rakshit Naidu, and Fatemehsadat Mireshghallah. 2023. Are chatbots ready for privacy-sensitive applications? an investigation into input regurgitation and prompt-induced sanitization. *Preprint*, arXiv:2305.15008.

Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. 2018. Real-time constrained cycle detection in large dynamic graphs. *Proc. VLDB Endow.*, 11(12):1876–1888.

Xubin Ren, Jiabin Tang, Dawei Yin, Nitesh Chawla, and Chao Huang. 2024. A survey of large language models for graphs. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 6616–6626, New York, NY, USA. Association for Computing Machinery.

Clayton Sanford, Bahare Fatemi, Ethan Hall, Anton Tsitsulin, Mehran Kazemi, Jonathan Halcrow, Bryan Perozzi, and Vahab Mirrokni. 2024. Understanding transformer reasoning capabilities via graph algorithms. In *Advances in Neural Information Processing Systems*, volume 37, pages 78320–78370. Curran Associates, Inc.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc.

Sola Shirai, Kavitha Srinivas, Julian Dolby, Michael Katz, Horst Samulowitz, and Shirin Sohrabi. 2025. Less is more: Learning graph tasks with just llms. *arXiv preprint arXiv:2508.10115*.

Shazia Tabassum, Fabiola Pereira, Sofia Fernandes, and João Gama. 2018. Social network analysis: An overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8:e1256.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2024a. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 491–500.

Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. 2025. Grapharena: Evaluating and exploring large language models on graph computation. In *International Conference on Learning Representations*, volume 2025, pages 48118–48145.

Xinyu Tang, Richard Shin, Huseyin A. Inan, Andre Manoel, Fatemehsadat Mireshghallah, Zinan Lin, Sivakanth Gopi, Janardhan Kulkarni, and Robert Sim. 2024b. Privacy-preserving in-context learning with differentially private few-shot generation. *Preprint*, arXiv:2309.11765.

Dat Tran and Douwe Kiela. 2026. Single-agent llms outperform multi-agent systems on multi-hop reasoning under equal thinking token budgets. *Preprint*, arXiv:2604.02460.

Fali Wang, Chenglin Weng, Xianren Zhang, Siyuan Hong, Hui Liu, and Suhang Wang. 2026a. Graphskill: Documentation-guided hierarchical retrieval-augmented coding for complex graph reasoning. *Preprint*, arXiv:2603.06620.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2023a. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36:30840–30861.

Jianing Wang, Junda Wu, Yupeng Hou, Yao Liu, Ming Gao, and Julian McAuley. 2024a. InstructGraph: Boosting large language models via graph-centric instruction tuning and preference alignment. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13492–13510, Bangkok, Thailand. Association for Computational Linguistics.

Rongzheng Wang, Shuang Liang, Qizhi Chen, Yihong Huang, Muquan Li, Yizhuo Ma, Dongyang Zhang, Ke Qin, and Man-Fai Leung. 2026b. Graphcogent: Mitigating llms' working memory constraints via multi-agent collaboration in complex graph understanding. In *Proceedings of the ACM Web Conference 2026*, WWW '26, page 3811–3822, New York, NY, USA. Association for Computing Machinery.

Rongzheng Wang, Shuang Liang, Qizhi Chen, Jiasheng Zhang, and Ke Qin. 2025. Graphtool-instruction: Revolutionizing graph reasoning in llms through decomposed subtask instruction. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, KDD '25, page 1492–1503, New York, NY, USA. Association for Computing Machinery.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. *Preprint*, arXiv:2203.11171.

Yanbang Wang, Hejie Cui, and Jon Kleinberg. 2024b. Microstructures and accuracy of graph recall by large language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 65154–65182. Curran Associates, Inc.

Yuyao Wang, Bowen Liu, Jianheng Tang, Nuo Chen, Yuhan Li, Qifan Zhang, Chenyi Zi, Chen Zhang, and Jia Li. 2026c. Npg-muse: Scaling long chain-of-thought reasoning with np-hard graph problems. *Preprint*, arXiv:2508.20373.

Chunyu Wei, Wenji Hu, Xingjia Hao, Xin Wang, Yifan Yang, Yunhai Wang, Yang Tian, and Yueguo Chen. 2026a. Graphchain: Large language models for large-scale graph analysis via tool chaining. *Advances in Neural Information Processing Systems*, 38:7402–7436.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models.

Yanbin Wei, Shuai Fu, Weisen Jiang, Zejian Zhang, Zhixiong Zeng, Qi Wu, James Kwok, and Yu Zhang. 2024. Gita: Graph to visual and textual integration for vision-language graph reasoning. In *Advances in Neural Information Processing Systems*, volume 37, pages 44–72. Curran Associates, Inc.

Yanbin Wei, Jiangyue Yan, Chun Kang, Yang Chen, Hua Liu, James Kwok, and Yu Zhang. 2026b. Dynamicgtr: Leveraging graph topology representation preferences to boost vlm capabilities on graph qas. *Preprint*, arXiv:2602.21864.

Yanbin Wei, Jiangyue Yan, Chun Kang, Yang Chen, Hua Liu, James T. Kwok, and Yu Zhang. 2025. Harnessing adaptive topology representations for zero-shot graph question answering. *Preprint*, arXiv:2508.06345.

Hao Xu, Xiangru Jian, Xinjian Zhao, Wei Pang, Chao Zhang, Suyuchen Wang, Qixin Zhang, Zhengyuan Dong, Joao Monteiro, Bang Liu, Qiuzhuang Sun, and Tianshu Yu. 2026. Graphomni: A comprehensive and extensible benchmark framework for large language models on graph-theoretic tasks. *Preprint*, arXiv:2504.12764.

Zixiang Xu, Yanbo Wang, Yue Huang, Chenxi Wang, Lang Gao, Zirui Song, Zhaorun Chen, Xiangliang Zhang, and Xiuying Chen. 2025. Gta: Graph theory agent and benchmark for algorithmic graph reasoning with llms.

Christos Xypolopoulos, Guokan Shang, Xiao Fei, Giannis Nikolentzos, Hadi Abdine, Iakovos Evdaimon, Michail Chatzianastasis, Giorgos Stamou, and Michalis Vazirgiannis. 2025. Graph linearization methods for reasoning on graphs with large language models. *Preprint*, arXiv:2410.19494.

Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4(2):100211.

Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pan, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, Bo An, and Qingsong Wen. 2025. A survey on trustworthy llm agents: Threats and countermeasures. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 6216–6226, New York, NY, USA. Association for Computing Machinery.

Zike Yuan, Yukun Cao, Han Zhang, Jianzhi Yan, Le Liu, Cai ke, Yue Yu, Hui Wang, Ming Liu, and Bing Qin. 2026. Egl-sca: Structural credit assignment for co-evolving instructions and tools in graph reasoning agents. *Preprint*, arXiv:2605.10366.

Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. 2025a. GraCoRe: Benchmarking graph comprehension and complex reasoning in large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7925–7948, Abu Dhabi, UAE. Association for Computational Linguistics.

Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. 2025b. MA-GTS: A multi-agent framework for solving complex graph problems in real-world applications. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 19286–19304, Suzhou, China. Association for Computational Linguistics.

Angelo Zangari, Peyman Baghershahi, and Sourav Medya. 2026. Colorful talks with graphs: Human-interpretable graph encodings for large language models. *Preprint*, arXiv:2602.10386.

Jiawei Zhang. 2023. Graph-toolformer: To empower llms with graph reasoning ability via prompt augmented by chatgpt. *arXiv preprint arXiv:2304.11116*.

Qifan Zhang, Nuo Chen, Zehua Li, Miao Peng, Jing Tang, and Jia Li. 2025a. Improving llms' generalized reasoning abilities by graph problems. *arXiv preprint arXiv:2507.17168*.

Qifan Zhang, Xiaobin Hong, Jianheng Tang, Nuo Chen, Yuhan Li, Wenzhong Li, Jing Tang, and Jia Li. 2025b. Gcoder: Improving large language model for generalized graph reasoning. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, CIKM '25, page 4149–4159, New York, NY, USA. Association for Computing Machinery.

Qifan Zhang, Jianhao Ruan, Aochuan Chen, Kang Zeng, Nuo Chen, Jing Tang, and Jia Li. 2026. Exposing weaknesses of large reasoning models through graph algorithm problems. *Preprint*, arXiv:2602.06319.

Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xiaochuang Han, Tianxing He, and Yulia Tsvetkov. 2024a. Can llm graph reasoning generalize beyond pattern memorization? In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2289–2305.

Yizhuo Zhang, Heng Wang, Shangbin Feng, Zhaoxuan Tan, Xinyun Liu, and Yulia Tsvetkov. 2025c. Generalizable llm learning of graph synthetic data with post-training alignment. *Preprint*, arXiv:2506.00845.

Yuting Zhang, Wei Ni, Kai Wang, Yizhang He, and Conggai Li. 2025d. Truss decomposition under edge local differential privacy. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pages 2670–2683.

Zeyang Zhang, Xin Wang, Ziwei Zhang, Haoyang Li, Yijian Qin, and Wenwu Zhu. 2024b. LLM4DyG: Can large language models solve spatial-temporal problems on dynamic graphs? In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4350–4361. Association for Computing Machinery.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2026. A survey of large language models. *Preprint*, arXiv:2303.18223.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. *Preprint*, arXiv:2205.10625.

Suirui Zhu, Xin Yuan, Baihe Ma, Wei Ni, and Wenjie Zhang. 2026. Beyond spatial privacy: Protecting trajectories with spatio-temporal differential privacy. *IEEE Transactions on Knowledge and Data Engineering*.

Yingjie Zhu, Xuefeng Bai, Kehai Chen, Yang Xiang, Jun Yu, and Min Zhang. 2025. Benchmarking and improving large vision-language models for fundamental visual graph understanding and reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30678–30701, Vienna, Austria. Association for Computational Linguistics.

A Scope Definition and Paper Selection

We collected candidate papers from major academic databases and preprint archives, including ACL Anthology, arXiv, Google Scholar, Semantic Scholar, DBLP, and OpenReview. Our search covers papers published or publicly released from the emergence of LLM-based graph computation to May 2026. To ensure broad coverage, we used combinations of LLM-related and graph-computation-related keywords. Representative search queries include `large language models for graph computation`, `LLM graph reasoning`, `LLM graph algorithm`, `LLM graph traversal`, `LLM shortest path`, `LLM graph benchmark`, `multi-agent graph reasoning`, `LLM graph tool use`, and `LLM graph code generation`. We also considered variants of these queries by replacing `LLM` with `large language model`, `graph computation` with `graph algorithm` or `graph reasoning`, and adding task-specific terms such as connectivity, triangle counting, graph coloring, Hamiltonian path, and maximum clique.

This survey explicitly focuses on LLMs participating in algorithmic graph computational tasks. To maintain this focused scope, we clarify that a paper is included if it satisfies the following conditions: 1) It explicitly involves LLMs in solving graph computational tasks. 2) The target task requires algorithmic or structural computation over graphs, such as shortest path, connectivity, traversal, triangle counting, graph coloring, or Hamiltonian path. 3) The paper provides benchmark construction, experimental evaluation, or systematic analysis related to LLM-based graph computation. We exclude the paper if it 1) focuses only on graph learning tasks such as node classification, link prediction, and graph classification unless they involve explicit algorithmic graph computation. For example, Tang et al. (2024a) is excluded because its primary focus falls outside our definition of graph computation. 2) General LLM-on-graph surveys that do not involve graph computation. 3) Methods without LLM involvement or purely multimodal methods without LLM participation.

B Summary of Graph Computation Tasks

A broad range of graph computation tasks has been explored in LLM-based methods, ranging from simple queries such as checking whether an edge exists to complex problems such as finding a Hamiltonian path. To provide a systematic overview of the tasks

Table 2: Summary of representative graph computation tasks, their definitions, and computational complexity.

Task	Definition	Time Complexity
Linear-time graph computation tasks		
Vertex Degree	Given an undirected graph G and a vertex v, return $\deg(v)$.	$O(n)$
Edge Existence	Given a graph G and vertices u, v, determine whether edge (u, v) exists in G.	$O(n)$
Common Neighbors	Given a graph G and vertices u, v, return their common neighbors.	$O(n)$
Graph Traversal	Given a graph G and a source vertex s, visit every vertex from s.	$O(n + m)$
Connectivity	Given an undirected graph G and vertices u, v, determine whether a path connects them.	$O(n + m)$
Connected Components	Given a graph G, partition its vertices into maximal connected components.	$O(n + m)$
Cycle Detection	Given a directed graph G, determine whether it contains a directed cycle.	$O(n + m)$
Bipartite Graph Check	Given a graph G, determine whether its vertices can be divided into two sets such that no edge connects vertices within the same set.	$O(n + m)$
Eulerian Path / Circuit	Given a graph G, determine whether it has a trail that visits every edge exactly once.	$O(n + m)$
Topological Sort	Given a directed acyclic graph G, output a linear ordering of all vertices in V such that for every directed edge $(u, v) \in E$, vertex u appears before vertex v in the ordering.	$O(n + m)$
Polynomial-time graph computation tasks		
Shortest Path	Given an edge-weighted graph G, source vertex u, and target vertex v, find a minimum-weight path from u to v.	$O(m+n \log n)$
Graph Diameter	Given a graph G, compute the largest shortest-path distance between two vertices.	$O(n(n + m))$
Minimum Spanning Tree	Given an edge-weighted graph G, find a spanning tree that connects all vertices and has the minimum possible total edge weight.	$O(m)$
Maximum Triangle Sum	Given a vertex-weighted graph G, find a triangle with maximum total vertex weight.	$O(m^{3/2})$
Maximum Flow	Given a directed graph G with edge capacities, a source vertex s, and a sink vertex t, compute the maximum s–t flow.	$O(nm^2)$
Bipartite Matching	Given a bipartite graph G, output a set of edges such that no two chosen edges share the same endpoint.	$O(m\sqrt{n})$
Triangle Counting	Given a graph G, count its triangles. A triangle is a cycle of length 3.	$O(m^{3/2})$
Clustering Coefficient	Given a graph G, compute exact local or global clustering coefficients that measure how likely the neighbors of a vertex are to be connected to each other.	$O(m^{3/2})$
PageRank	Given a directed graph G, compute PageRank scores for T power iterations.	$O(T(n + m))$
NP-complete graph computation tasks		
Hamiltonian Path / Cycle	Given a graph G, determine whether there exists a simple path or cycle that visits every vertex in G exactly once.	NP-complete
Traveling Salesman Problem	Given a complete weighted graph G and a bound B, determine whether G has a Hamiltonian cycle of weight at most B.	NP-complete
Maximum Clique	Given a graph G and an integer r, determine whether G has a clique of size at least r.	NP-complete
Maximum Independent Set	Given a graph G, determine whether there exists a subset of vertices such that no two vertices in the subset are adjacent.	NP-complete
Minimum Vertex Cover	Given a graph G and an integer r, determine whether there exists a subset of vertices of size at most r such that every edge in G has at least one endpoint in the subset.	NP-complete
Graph Coloring	Given a graph G and an integer $c \geq 3$, determine whether G has a proper c-coloring.	NP-complete
Dominating Set	Given a graph G and an integer r, determine whether there exists a set of at most r vertices such that every vertex in G is either in the set or adjacent to a vertex in the set.	NP-complete
Subgraph Matching	Given a host graph G and a pattern graph H, determine whether H is isomorphic to a subgraph of G.	NP-complete
Graph Edit Distance	Given graphs G_1, G_2, and a bound B, determine whether G_1 can be transformed into G_2 using at most B edit operations.	NP-complete
Maximum Common Subgraph	Given graphs G_1, G_2, and an integer r, determine whether they have a common subgraph with at least r vertices.	NP-complete

T: the number of PageRank iterations.

used in existing studies, we summarize representative graph tasks in Table 2, including the definition and computational complexity.

Following prior work on graph computation benchmarks (Chen et al., 2024a), we categorize these tasks by computational complexity into linear-time, polynomial-time, and NP-complete tasks. Because computational complexity captures the intrinsic algorithmic difficulty of a task, it directly reflects the different reasoning requirements imposed on LLMs. Consequently, organizing tasks by complexity provides a natural framework for comparing the reasoning abilities of LLMs in handling graph computation. In complexity analysis, we refer to standard graph algorithms and denote the number of vertices and edges by $n = |V|$ and $m = |E|$, respectively, and the degree of vertex v is denoted as $\deg(v)$.

– Linear-time graph tasks are tasks that can be solved in time proportional to the size of the graph, typically $O(n + m)$, or proportional to the size of a local neighborhood, such as $O(\deg(v))$. Solving these tasks requires LLMs to inspect vertices and edges or perform standard graph traversal.

– Polynomial-time graph tasks are tasks that can be solved by algorithms whose running time is bounded by a polynomial function of the graph size, such as $O(n^2)$, $O(n^3)$, or $O(m \log n)$. These tasks usually require LLMs to execute multi-step logical deduction and track intermediate states.

– NP-complete tasks represent a much higher level of computational difficulty, as no polynomial-time algorithms are known to exist for these problems. These tasks require LLMs to perform complex reasoning and often demand abilities such as dynamic backtracking and systematic exploration of possible solutions.

C Introduction of Benchmark Datasets

In this section, we list benchmark datasets for LLMs for graph computation. Table 4 summarizes representative benchmark datasets that have been used to evaluate large language models for graph computation. This size column indicates the largest graph scale in each dataset. The scale is calculated as $\mathrm{round}(\log_{10} n_{\max})$, where $n_{\max}$ is the largest reported vertex count in graph computation tasks. The graph type column describes the structural or semantic properties of the graphs, such as whether they are directed, weighted, dynamic, heterogeneous, or attributed. The graph source column reports whether the graphs are synthetic or derived from real-world data. The meaning of the abbreviations used in the graph type and graph source columns is given in Table 3. The task coverage columns indicate whether the dataset includes linear-time, polynomial-time, or NP-complete graph computation tasks. Finally, the number of tasks column reports the total number of graph computation tasks included in each dataset when available.

Table 3: Signs used in Table 4.

Sign	Meaning
Network Format	
U	Unipartite, undirected, and unweighted graph
D	Directed graph
D_{acyc}	Directed acyclic graph
B	Bipartite graph
+	Weighted or capacitated graph
T	Dynamic or temporal graph
Het	Heterogeneous graph
Attr	Attributed or text-attributed graph
Graph Source	
RW	Real-world graph
Syn	Synthetic graph

Table 4: Summary of benchmark datasets for evaluating large language models for graph computation.

Name	Graphs			Tasks			# of Tasks
	Size	Type	Source	Linear	Poly	NP-Comp	
GPT4Graph (Guo et al., 2023)	$\sim 10^1$	U, D, Attr	RW	✓	✓	–	10
NLGraph (Wang et al., 2023a)	$\sim 10^2$	U, D, +, B, D_{acyc}	Syn	✓	✓	✓	8
LLMtoGraph Eval. (Liu and Wu, 2023)	$\sim 10^2$	U	Syn	✓	✓	–	5
GraphQA (Fatemi et al., 2024)	$\sim 10^1$	U	Syn	✓	–	–	7
GraphLLM Eval. (Chai et al., 2026)	$\sim 10^2$	U, B, Attr	RW	–	✓	–	4
LLM4DyG (Zhang et al., 2024b)	$\sim 10^1$	T, U	RW, Syn	–	✓	–	9
GVLQA / GITA (Wei et al., 2024)	$\sim 10^1$	U, D, +, B	RW, Syn	✓	✓	✓	7
GraphWiz (Chen et al., 2024a)	$\sim 10^2$	U, D, +, B	Syn	✓	✓	✓	9
GraphInstruct (Luo et al., 2025)	$\sim 10^2$	U, D, +, B	Syn	✓	✓	✓	21
InstructGraph (Wang et al., 2024a)	$\sim 10^2$	U, D, +, Het, Attr	RW, Syn	✓	✓	✓	29
NLGIFT (Zhang et al., 2024a)	$\sim 10^1$	U, D, +	Syn	✓	✓	–	4
ProGraph (Li et al., 2024a)	$\sim 10^6$	U, D, +, Attr	RW	✓	✓	–	3
GraphArena (Tang et al., 2025)	$\sim 10^2$	U, +	RW	–	✓	✓	10
GraCoRe (Yuan et al., 2025a)	$\sim 10^1$	U, D, +, Het, Attr	RW, Syn	✓	✓	✓	19
Graph Pattern Comp. (Dai et al., 2025)	$\sim 10^2$	U, D, Attr	RW, Syn	–	✓	–	11
GCoder / GraphWild (Zhang et al., 2025b)	$\sim 10^2$	U, D, +, B	RW, Syn	✓	✓	✓	13
GraphSilo / GraphPRM (Peng et al., 2025)	$\sim 10^2$	U, D, +	Syn	✓	✓	✓	13
GTools (Wang et al., 2025)	$\sim 10^2$	U, D, +	Syn	✓	✓	–	20
G-REAL / MA-GTS (Yuan et al., 2025b)	$\sim 10^1$	U, +	RW	✓	✓	✓	4
GraphPile (Zhang et al., 2025a)	$\sim 10^2$	U, D, +, B	RW, Syn	✓	✓	–	23
GraphAlgorithm (Hu et al., 2025)	$\sim 10^3$	U, D, +	RW	✓	✓	✓	239
GT Bench / GTA (Xu et al., 2025)	$\sim 10^2$	U, D, +	Syn	✓	✓	✓	44
GraphOmni (Xu et al., 2026)	$\sim 10^1$	U, B	Syn	✓	✓	–	6
G1 / Erdős (Guo et al., 2025)	$\sim 10^2$	U, D, +	RW	✓	✓	✓	50
VGCURE (Zhu et al., 2025)	$\sim 10^1$	U, D, Attr	RW, Syn	✓	✓	–	22
GraphSQA / GraphInsight (Cao et al., 2025)	$\sim 10^2$	U, D, +	Syn	✓	–	–	20
LLMTM (Hao et al., 2026)	$\sim 10^2$	T, U	RW, Syn	–	✓	–	6
GRAlgoBench (Zhang et al., 2026)	$\sim 10^2$	U, +	RW	✓	✓	✓	9
ComplexGraph / GraphSkill (Wang et al., 2026a)	$\sim 10^4$	U, D, D_{acyc}, +	Syn	✓	✓	✓	23
Grena / GraphVista (Han et al., 2026)	$\sim 10^3$	U, +	Syn	✓	✓	✓	20
Graph4real / GraphCogent (Wang et al., 2026b)	$\sim 10^3$	U, D, +, Attr	RW	✓	✓	–	21
Less-is-More Graph Tasks (Shirai et al., 2025)	$\sim 10^2$	D, Attr	Syn	✓	–	–	8
GraphThought Dataset (Huang et al., 2025)	$\sim 10^2$	U, +, Attr	RW	✓	✓	✓	10
NPG-Muse Corpus (Wang et al., 2026c)	$\sim 10^1$	U, +, Attr	Syn	–	–	✓	3
TRF Preference Dataset (Wei et al., 2025)	$\sim 10^1$	U, D, +, B, D_{acyc}	Syn	✓	✓	✓	7
GTR Preference Dataset (Wei et al., 2026b)	$\sim 10^1$	U, D, +, B, D_{acyc}	Syn	✓	✓	✓	7