Can LLMs and LLM-based agents effectively solve classical graph problems?

Michael Surazhsky
Ben-Gurion University of the Negev
Be'er-Sheva, Israel
surazhsk@post.bgu.ac.il

Guy Amzaleg
Ben-Gurion University of the Negev
Be'er-Sheva, Israel
amzalegg@post.bgu.ac.il

Itay Zloczower
Ben-Gurion University of the Negev
Be'er-Sheva, Israel
itayzloc@post.bgu.ac.il

Tomer Ovadya
Ben-Gurion University of the Negev
Be'er-Sheva, Israel
ovadyat@post.bgu.ac.il

ABSTRACT

Todo.

KEYWORDS

Graph reasoning, Large Language Models, Tool use, Graph algorithms, LLM Agents

ACM Reference Format:
Michael Surazhsky, Guy Amzaleg, Itay Zloczower, and Tomer Ovadya. 2025. Can LLMs and LLM-based agents effectively solve classical graph problems?. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 4 pages.

1 INTRODUCTION

1.1 Background

Classical graph algorithms provide exact ground truth, making them ideal for evaluating LLM reasoning. However, LLMs process graphs as linear text rather than structured objects, leading to performance degradation as graph size increases [1].

Most failures arise from misreading graph structure rather than faulty reasoning. The core challenge lies in the mismatch between non-linear graphs and the sequential nature of transformers [3].

LLM-based graph reasoning typically follows three paradigms:

Zero/few-shot prompting: The model receives a textual description of the graph and is asked to reason and compute the answer directly, without training on the task; accuracy depends strongly on how the graph is serialized [3].

Fine-tuning: The model is trained on graph–question examples to learn algorithmic patterns and output formats, improving reliability on seen tasks but often limiting generalization to new graph structures [5].

Tool-based execution: The LLM parses the graph, selects the appropriate algorithm, and delegates computation to an external solver; performance relies on correct extraction and invocation rather than internal reasoning [7].

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), A. El Fallah Seghrouchni, Y. Vorobeychik, S. Das, A. Nowe (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). This work is licenced under the Creative Commons Attribution 4.0 International (CC-BY 4.0) licence.

This work compares internal reasoning and tool-based execution under different graph encodings using fixed algorithms and ground truth.

2 RELATED WORK

Research on LLM-based graph reasoning falls into three main areas: benchmarking core capabilities, studying the effects of graph serialization, and enhancing performance through instruction tuning, tools, or agent-based architectures.

2.1 Benchmarks for Graph Reasoning with LLMs

Initial efforts to systematically evaluate LLM capabilities in graph reasoning have led to the creation of several benchmarks that test their ability to process natural language descriptions into actionable graph structures.

The **NLGraph** benchmark [1] is an early effort to explicitly test LLMs on graph problems presented in natural language. Comprising 29,370 problems across eight algorithmic tasks (e.g., connectivity, Hamiltonian path), it revealed that LLM performance on simple tasks does not translate to complex ones, where they often fail to track structure and exploit spurious correlations.

The **GraphArena** benchmark [2] targets practical graph computation by using large, real-world graphs to test ten tasks, including both polynomial and NP-complete problems (e.g., Traveling Salesperson). Evaluations across multiple LLMs revealed that even top models struggle with larger instances, exhibiting high rates of hallucination and only limited improvement from Chain-of-Thought prompting.

2.2 Effects of Prompting – Graph Serialization, Structural Priors and Instruction Tuning

A second line of research shows that graph serialization strongly affects LLM performance. **Talk Like a Graph** [3] demonstrates that accuracy varies widely across encodings, with incident-based formats consistently outperforming adjacency-style representations and performance depending on graph topology.

Conversely, the **One For All (OFA)** framework [4] argues that text serialization inevitably loses structural fidelity. Instead of serializing the topology, OFA uses LLMs solely to encode node attributes into a unified space, preserving the native graph structure for GNN processing.

2.3 Fine Tuning, Tool Use, and Agent Architectures

Finally, a fourth research direction focuses on improving graph reasoning through fine-tuning, tool integration, and agent-based designs.

GraphWiz builds an instruction-following LLM specialized for graph computation by fine-tuning on the GraphInstruct dataset [5]. This suggests that task-specific supervision and preference tuning can provide significant gains, at least within the training distribution.

Language is All a Graph Needs fine-tunes LLMs to perform classification and prediction tasks on graph-structured data, achieving competitive performance with GNNs on benchmarks including ogbn-arxiv, Cora, and PubMed [6].

GraphTool Instruction combines instruction tuning with tool-based execution by transitioning the LLM from an algorithm simulator to an orchestrator that performs graph extraction, algorithm selection, and parameter filling via external solvers [7]. This approach improves scalability and execution reliability.

GraphAgent Reasoner extends this approach using agent-based architectures, in which an LLM orchestrator controls multiple node agents to explicitly simulate distributed graph algorithms [8].

Although these works demonstrate progress through better representations, supervision, and tooling, they make it difficult to isolate which components drive performance gains. This project systematically disentangles these factors through controlled comparisons to assess when and how out-of-the-box LLMs succeed on graph problems.

2.4 Key Concepts and Representative Works

To summarize the literature discussed above, prior work on LLM-based graph reasoning can be organized around several recurring conceptual themes rather than isolated systems. *Language is All a Graph Needs* explores generative graph learning by framing node and link prediction tasks through natural language supervision and instruction tuning [6]. *GraphTool-Instruction* formalizes tool-augmented graph reasoning, where LLMs act as orchestrators that extract graph structure, select algorithms, and invoke external solvers [7]. *Scalable and Accurate Graph Reasoning with LLM-Based Multi-Agents* extends this paradigm by introducing multi-agent coordination to explicitly simulate distributed graph algorithms [8]. *Talk Like a Graph* demonstrates that the choice of graph serialization critically affects reasoning accuracy, with incident-based encodings consistently outperforming adjacency-style formats [3]. *NLGraph* provides a large-scale benchmark for evaluating natural-language graph reasoning and highlights persistent failures due to structural misinterpretation and spurious correlations [1]. *GraphArena* evaluates LLMs on large, real-world graph computation tasks, revealing severe scalability limitations and hallucination effects on complex instances [2]. *GraphWiz* shows that instruction tuning and graph-specific chain-of-thought supervision can significantly improve performance on graph reasoning tasks within the training distribution [5]. Finally, *One For All* argues that pure text-based graph serialization is inherently lossy and instead proposes preserving native graph structure while leveraging LLMs for semantic encoding [4].

3 METHODOLOGY

3.1 Framework Overview

We adopt **GraphNL** [1] as the base problem formulation. GraphNL generates synthetic graphs with controllable structural properties, renders them into fixed natural-language templates, appends a task query, and computes exact ground-truth solutions using classical graph algorithms. This design enables fully automated, reproducible evaluation across a wide range of graph reasoning tasks.

Our contribution extends GraphNL from a prompt-only evaluation benchmark into an executable reasoning framework. Instead of relying on implicit internal reasoning within a single LLM call, we decompose reasoning into explicit, verifiable stages that operate over structured representations derived from natural language.

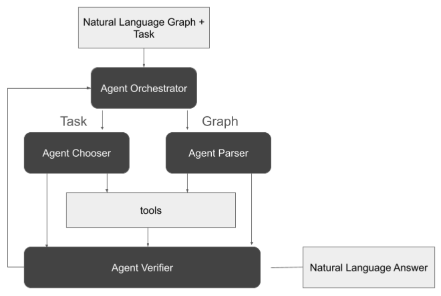
3.2 Agent-Based Reasoning Pipeline

Given a natural-language input containing both a graph description and a task query, reasoning is executed by four specialized agents coordinated by a central *Agent Orchestrator*. The orchestrator separates the input into graph and task components and controls the execution flow.

The *Agent Parser* converts the natural-language graph description into a structured representation by extracting nodes, edges, directions, and task-specific constraints, following the GraphNL templates [1]. In parallel, the *Agent Chooser* classifies the task type (e.g., connectivity, shortest path, or maximum flow) and selects the appropriate classical graph algorithm.

Once the graph structure and algorithm are determined, classical algorithmic tools operate on the structured graph to compute an exact solution. Finally, the *Agent Verifier* validates both the parsed graph structure and the computed output against consistency checks and task-specific correctness conditions. If validation fails, execution is rejected and repeated using a reject-and-repair loop.

This design replaces implicit internal reasoning with explicit structure and tool execution, enabling inspection, correction, and reproducibility, similar in spirit to recent tool-based and agent-based graph reasoning systems [7, 8].



3.3 Supported Tasks

The framework supports all GraphNL tasks [1], including connectivity, cycle detection, topological sorting, shortest path, maximum flow, bipartite matching, Hamiltonian path, and graph neural network message passing. Importantly, all inputs remain purely natural language, and no symbolic graph representation is provided directly to the model.

3.4 Novel Contributions

Our methodology introduces three key contributions:

(1) An agent-based execution framework grounded in the GraphNL natural-language graph formulation, which has not previously been used in tool-based or agent-based graph reasoning systems;
(2) A graph-specific LLM actor–critic design, in which parsing and strategy selection act as the actor, while structural and solution validation act as the critic;
(3) An iterative reject-and-repair loop that improves robustness as graph size and complexity increase.

3.5 Minimal Example

Input: "Nodes A, B, C, D with edges A–B, B–C, C–D. Is A connected to D?"

The Agent Parser extracts the graph structure, the Agent Chooser selects a connectivity algorithm, the execution tool performs breadth-first search, and the Agent Verifier confirms correctness. The system outputs a valid path:

$$A \rightarrow B \rightarrow C \rightarrow D.$$

4 DESIGN OF EXPERIMENTS

This section describes the experimental protocol used to evaluate the effectiveness of our proposed agent-based reasoning framework compared to standard LLM prompting strategies for classical graph problem solving. The goal of these experiments is to isolate the impact of explicit structure, tool execution, and verification on graph reasoning performance.

4.1 Research Questions and Hypotheses

The core objective of this work is to evaluate whether agentic reasoning frameworks can systematically improve the reliability of Large Language Models (LLMs) on classical graph problems. While recent LLMs demonstrate emerging graph reasoning abilities, prior benchmark studies such as **NLGraph** [1] and **GraphArena** [2] have shown that these capabilities are often brittle, sensitive to prompt formulation, and prone to structural misinterpretation as graph complexity increases.

We hypothesize that an agent-based architecture—combining explicit graph parsing, algorithm selection, external tool execution, and iterative verification—can mitigate these limitations and yield more robust and interpretable reasoning behavior than prompt-only approaches.

4.2 Data Selection and Preparation

We evaluate our framework primarily on the **NLGraph** benchmark [1], which contains 29,370 natural-language graph reasoning problems spanning eight algorithmic tasks. To focus on the most challenging and realistic setting, we restrict our evaluation to the natural-language subsets, requiring the system to recover graph structure solely from textual descriptions.

To increase structural diversity, we additionally incorporate selected tasks from the **GraphInstruct** dataset [5], which was originally introduced to support instruction tuning for graph reasoning and includes diverse edge cases such as disconnected components and self-loops. All inputs are preserved in their original natural-language form to ensure a fair comparison with prompt-based baselines. Structured representations (e.g., adjacency lists) are generated internally by the Agent Parser and are not provided explicitly to the model.

4.3 Computational Resources

Experiments are conducted on local machines equipped with multi-core CPUs and consumer-grade GPUs. The agentic framework is implemented in Python 3.10 using LangChain and LangGraph for orchestration. Local models are served via Ollama, while API-based models are accessed through standard interfaces. Since our evaluation focuses on reasoning correctness rather than throughput, runtime performance is not treated as a primary metric.

4.4 Open-Source Components

Our experimental setup relies exclusively on open-source or publicly available resources:

- **Benchmarks:** NLGraph [1]
- **Libraries:** NetworkX for classical graph algorithms; LangGraph for stateful multi-agent coordination
- **Models:** Llama-3.1-8B-Instruct, DeepSeek-R1-8B, and GPT-4o (API-based baseline)

4.5 Performance Metrics

We evaluate system performance using three complementary metrics:

- **Exact Match Accuracy (ACC):** The fraction of queries for which the final output exactly matches the ground-truth solution.
- **Tool-Calling Success Rate (TSR):** The proportion of cases in which the Agent Chooser selects the correct algorithm and the Agent Parser produces valid inputs for tool execution.
- **Repair Efficiency:** The percentage of initially incorrect intermediate states that are successfully corrected through the reject-and-repair verification loop.

4.6 Benchmarking Methods

We compare our agent-based framework against three baselines:

- **Zero-Shot Prompting:** Direct prompting using the natural-language input, following the evaluation protocol of NLGraph [1].
- **Chain-of-Thought (CoT):** Prompting the model to produce step-by-step reasoning without access to external tools, as commonly used in prior graph reasoning studies [3].
- **Agentic Pipeline (Ours):** The full orchestrator-driven pipeline with explicit parsing, tool execution, and verification.

All methods are evaluated on identical inputs using the same underlying models to ensure controlled comparisons.

4.7 Parameter Settings

To minimize stochastic effects and ensure reproducibility, we adopt a deterministic configuration across all experiments:

- **Temperature:** 0.0
- **Maximum Iterations:** 3 reject-and-repair cycles
- **Context Window:** 8,192 tokens

These settings are fixed across all models and baselines.

4.8 Statistical Analysis

To assess the statistical significance of observed performance differences, we apply:

- **McNemar's Test** to compare paired success/failure outcomes between the agentic framework and Chain-of-Thought prompting.
- **ANOVA** to analyze the effect of model choice (local vs. API-based) on accuracy within the agentic framework.

4.9 Ablation Study Design

To isolate the contribution of each component of the proposed framework, we conduct a set of ablation experiments in which individual modules are selectively disabled.

Specifically, we evaluate:

- **No-Verification Ablation:** The Agent Verifier is removed, and the system returns the first tool-executed result without reject-and-repair.
- **No-Tool Ablation:** The Agent Chooser is disabled, and the LLM is forced to reason without invoking external graph algorithms.
- **No-Parsing Ablation:** The Agent Parser is bypassed, and the model receives the graph only in its raw natural-language form.

Each ablated variant is evaluated using the same benchmarks, models, and metrics as the full system. This allows us to quantify the individual impact of structured parsing, tool execution, and verification on overall reasoning performance.

REFERENCES

[1] Z. Wang et al., "NLGraph: A Benchmark for Large Language Models on Graph Tasks," arXiv, 2023.
[2] Y. Yang et al., "GraphArena: Evaluating and Comparing Large Language Models on Graph Computation," arXiv, 2023.
[3] H. Liu et al., "Talk Like a Graph: Encoding Graph Structure for Large Language Models," arXiv, 2023.
[4] X. Liu et al., "One For All: Training a Graph Foundation Model," arXiv, 2022.
[5] J. Zhang et al., "GraphWiz: Instruction Tuning for Graph Reasoning," arXiv, 2023.
[6] J. Nie et al., "Language Is All a Graph Needs," arXiv, 2023.
[7] T. Wu et al., "GraphTool Instruction: Orchestrating LLMs with Graph Algorithms," arXiv, 2024.
[8] S. Chen et al., "Scalable and Accurate Graph Reasoning with LLM-Based Multi-Agents," arXiv, 2024.