Measuring LLM performance in Graph Traversal tasks

by sannysanoff and his aider.
21 Dec 2024.

Abstract

This study provides a systematic framework for evaluating the graph traversal capabilities of Large Language Models (LLMs). It introduces a parametric methodology for generating and analyzing directed acyclic graphs (DAGs) with controlled complexity metrics. By applying this approach, the research enables structured assessment of path-finding tasks across graphs of varying structural complexities, offering insights into LLMs' reasoning capabilities.

Introduction

Large Language Models have become increasingly proficient in various structured reasoning tasks. One fundamental area requiring deeper exploration is graph traversal, which integrates pattern recognition, sequential logic, and spatial reasoning. To evaluate this capability, this research presents a benchmark system employing methodologically designed DAGs, enabling a consistent and rigorous analysis of LLMs' abilities to navigate and solve graph-related tasks.

The ability of large language models (LLMs) to traverse DAGs and find a correct path offers intriguing parallels to their reasoning and logical manipulation capabilities. This graph traversal task tests not only memory recall but also dynamic pathfinding, a skill that intersects with reasoning about dependencies, constraints, and implications. Success in this task requires aligning stored conceptual knowledge (memory) with real-time computation to identify valid sequences, akin to solving logical puzzles or forming chains of reasoning in more abstract domains. Variations in performance among LLMs on increasingly complex DAGs suggest differences in how these models internally represent and process structured relationships. For example, more advanced reasoning models may better synthesize stored information with constraints imposed by graph structure to build consistent conclusions, potentially reflecting superior algorithmic reasoning, token interdependencies, and emergent logical capabilities. These insights underline the importance of benchmark design in evaluating LLM strengths and identifying bottlenecks in their pathfinding and reasoning faculties.

Purpose

Originally this test was intended to detect best of small open-weight models, to find hidden gems, but they, apart from few poor performers, were mostly in the same league, if you compare to proprietary ones, and with a bit of pity we can conclude (see conclusion) that proprietary models have quite a few cards in the sleeves, not seen in open models yet.

Of course, we were also embarrassed by people doing strawberry tests.

2. Methodology

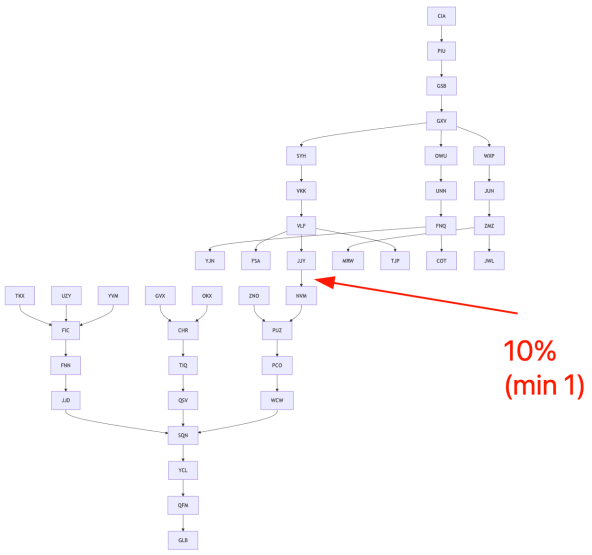
2.1 Graph Generation Framework

The benchmark employs a graph generation system defined by three primary parameters to control structural properties:

- Node Split Probability (N): The likelihood of path divergence at each node, ranging from 0.2 to 0.8
- Maximum Splits (S): The upper limit on the number of child nodes per parent node, set between 2 and 6.
- Depth (D): The depth of the graph, controlled between 2 and 8, determining the hierarchical structure.

The generated DAGs use a dual-tree framework with controlled cross-connections, maintaining a constant cross-connection density (k=0.1) to ensure uniform characteristics across all test cases.

Example dual tree is represented below. On this example, there's only one path from start to end node (top to bottom). Tree was constructed, then mirrored, then joined utilizing 10% (but not less than one) edge.



Mermaid representation was given to LLM, and node traversal sequence request was requested. This was a zero-shot prompt.

Example requested graph:

```
graph TD
   TKX --> FIC
   FNQ --> YJN
   JJD --> SQN
   VLF --> FSA
   CHR --> TIQ
   GVX --> CHR
   OWU --> UNN
    .......
```

2.2 Complexity Metrics

Graph complexity is quantified using two key metrics:

1. Line Count Complexity (LL):

This is defined as the cardinality of the graph's edge set:

$$L=|E|$$

where EE represents the set of edges.

2. Structural Complexity (CC):

Approximated using:

$$C\approx N\times S^{D}$$

Here, N, S, and D correspond to the node split probability, maximum probable splits, and fixed depth, respectively.

2.3 Test Set Generation and Selection

Test graphs are systematically generated and filtered through a multi-phase process:

- Initial Generation: For each parameter set (N, S, D), 100 iterations produce a pool of graphs. Invalid graphs are excluded, and the median line count is calculated.
- Filtering: Graphs exceeding 1000 nodes are eliminated. Remaining graphs are grouped by complexity, with 10 variants selected per level.

- Selection Criteria: Strict, relaxed, and extended selection thresholds ensure diversity and representativeness, based on how closely a graph's line count aligns with the median (exact, 5%, or 10%).

2.4 Benchmark Execution Protocol

The protocol evaluates LLMs through the following tasks:

1. Path Finding Task:
   Models identify valid paths from start nodes (in-degree= 0) to end nodes (out-degree=0). Multiple correct paths are acceptable if validity is confirmed.

2. Testing Parameters:
   - Each model is allowed up to three attempts per graph.
   - A graph is considered solved if any valid path is discovered.
   - Testing terminates after 24 consecutive failures.
   - Within each complexity category, stable shuffling is used.

2.5 Performance Metrics

The following metrics assess LLM performance:

1. Success Rate (SR):
   For each complexity level (cc) and model (mm):

   $$SR(c,m)=(successful\_tests / total\_tests)\times 100$$

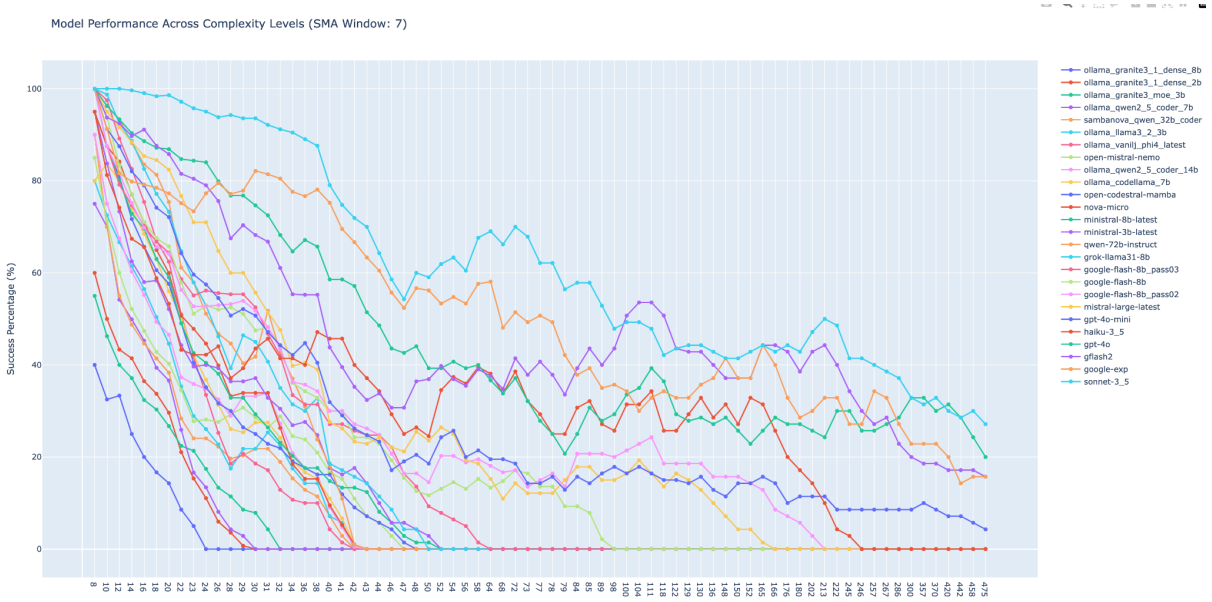
2. Complexity Correlation Analysis:

   Success rates are plotted against complexity levels to evaluate performance trends. Optional smoothing using a Simple Moving Average can be applied for clearer insights. Performance rankings of models are derived based on success rates across varying complexities.

This structured methodology ensures a consistent, measurable evaluation of LLM capabilities in graph traversal tasks.

NB As it turned out after tests, the complexity metric in few areas poorly correlates with actual complexity as evidenced by lower models scores in this areat, but it does not greatly affect final insights.

3. Benchmark Results



See various interactive charts results at:
https://sannysanoff.github.io/simple-llm-benchmark/index.html.

3.1 Benchmark Analysis / Conclusions

- of all tested free/open weights small models (sizes up to 14B), LLama-3.1-8B was performing best.
- IBM Granite models were performing worst, regardless of size.
- until to certain point, 3 passes of google-1.5-flash-8b behave within margin of statistical error, but for unknown reasons sometimes just stop working on higher complexities. It is not statistical error. Maybe model deployment differences, where old models coexist with new. Maybe time of the day.
- Google 1.5 flash (non 8B) was NOT measured, but google 2.0 flash is very cool in this benchmark. It is compared to the top tier models in different comparison.
- It can be seen that for weaker complexities, google flash 2.0 outperforms gpt-4o-mini and haiku 3.5 with large margin, and on big complexities outperforms gpt-4o.
- In general, Google Flash 8B model is very interesting. It is very deeply thinking, compared to its size. Also it's very cheap. Wonder why no other model is like this.

- Qwen models for this task they all are similarly weak. Interestingly, Coder-14B model performed better than Coder-32B model. Coder-14B is K5_M model ran on local Nvidia card, while Coder-32B was run via sambanova.

- all Mistral models are probably built with very same architecture and constant depth. They are very close, except mistral-large, which gained more depth due to size. Interestingly, ministral-3b perform best of all non-large models.

- - o1-mini is not included. o1-mini and above models are of completely different league achieving **close to 100% at complexity 200**, and it was not benchmarked at this complexity scale. We admit OpenAI has big lead in this area.

- - **all charted models often make errors on very simple tasks**. You can see by task complexity 016 and higher, no model on this chart produced 100% correct results. There are around 30-40 simpler tests per simple complexity level.

- Qwen QwQ stopped producing results around complexity 72 (determined by random sampling), and was not included in the chat. We cannot tell if it achieves 100% in the simpler tests. We ran out of free glhf quota to benchmark it fully.

- Sonnet 3.5 is best of class of non-reasoning models.

- Google Learn-Exp is below Sonnet 3.5 in this task.

- Google Flash2 is close to Sonnet on more complex tasks, but constantly is not as performant/precise in simple tasks. It makes it interesting player.

- 4o-mini outpaces Haiku only on higher complexity levels.

- Maybe, 4o outpaces Sonnet beyond the edge of tested complexities range.

4 Pitfalls and future work

- While we performed several runs for one of the models (google flash 1.5 8b), and we achieved illustration of statistical error when running same model several times, we stumbled upon its varying (non-reproducible) performance on complex levels. This is yet to investigate.

- We need to perform more repeated testing on different models as well, however some tests/restarts during this testing proved that results are quite consistent (on same set of tasks, of course).

- We need to investigate the influence of node names on task performance, outside of scope of this work. Of course current naming of the nodes (random) can influence performance because of additional semantic meaning names can accidentally have.

- We do not pretend on complete scientific correctness and made our results published in paper-like form for lulz.

- This benchmark can be used to measure model quality fluctuation over time. Observers notice that there's a feeling OpenAI models are getting dumber with time. This way, or by providing similar benchmarks, we can now be sure that is not only feeling, but measurable figure.

- Most expensive was openai testing with circa $15 for o1-mini, 4o and 4o-mini models single pass, all together. Anthropic costs (smaller) were covered by other people. Costs of Google and AWS Nova models were negligible. Rest of tests were within free quota / throttled / local nvidia card.