Graph Reasoning via Self-Correcting Multi-Agent Debate

Implementation and Experimental Analysis

Group: WW

December 15, 2025

Abstract

Large Language Models (LLMs) traditionally struggle with graph-based reasoning tasks due to context window limitations, hallucinations, and a lack of backtracking capabilities. This project implements a Multi-Agent System using the `LangGraph` framework to address these issues. We designed two architectures: a linear **Backtracking Solver** and a parallel **Consensus Solver**, testing them across two models (Ministral-8b and Llama-3.1-8b). Our experiments on the GraphWiz dataset reveal that while the Consensus architecture aids in complex combinatorial tasks like Hamiltonian Paths, it introduces significant computational overhead. This overhead often causes the system to exceed recursion limits on simpler structural tasks, leading to higher error rates compared to the leaner Backtracking architecture. We conclude that "more agents" does not strictly equal better performance; the architecture must be tuned to the complexity of the reasoning task.

Contents

1 Introduction

Graph reasoning requires a rigorous adherence to topological constraints (nodes and edges) and the ability to perform look-ahead planning. Standard LLMs, which generate text linearly, often fail in this domain because they cannot natively "undo" a wrong step or maintain a consistent internal state of visited nodes. This leads to three primary risks:

1. **Loss of Context:** Describing entire graphs in a prompt saturates the window, causing the model to forget connections.
2. **Structural Hallucinations:** Models invent edges to force a solution in a single Chain-of-Thought (CoT) generation.
3. **Inability to Backtrack:** If an LLM mistakes step 3 of 10, the entire subsequent generation is compromised.

The objective of this project is to move beyond monolithic prompting by implementing an agentic workflow. We aim to answer the question: *Can a multi-agent debate system correct the hallucinations of a single model?* We propose a solution based on **LangGraph**, employing specialized agents (Proposer(s), Verifier, Manager) to decompose the problem into discrete, verifiable steps.

2 Related Work

Our work builds upon recent advancements in LLM reasoning and agent orchestration.

GraphWiz [1] establishes the baseline for this domain. Chen et al. demonstrated that while instruction-tuned models can solve some graph problems, they struggle with complexity. We utilize their dataset and evaluation metrics as our ground truth.

To address the linear limitation of LLMs, we draw inspiration from **Tree of Thoughts (ToT)** [2]. Yao et al. proposed that problem-solving should be a tree search rather than a line. Our *Backtracking Solver* implements this by explicitly maintaining a state stack that allows the agent to return to previous nodes upon hitting a dead end.

Finally, the multi-agent interaction model is influenced by **ChatDev** [3], which assigns specific roles (e.g., CEO, Programmer) to different model instances. Similarly, we separate our architecture into *Proposers* (generating moves) and *Verifiers* (checking validity), creating a "debate" that filters out hallucinations before they become part of the solution.

3 Dataset and Exploratory Data Analysis (EDA)

We utilized the **GraphWiz/GraphInstruct** dataset. Before designing the agents, we performed a Exploratory Data Analysis (EDA) using `graphwiz.py` and `nlgraph.py` to understand the data distribution and response formats.

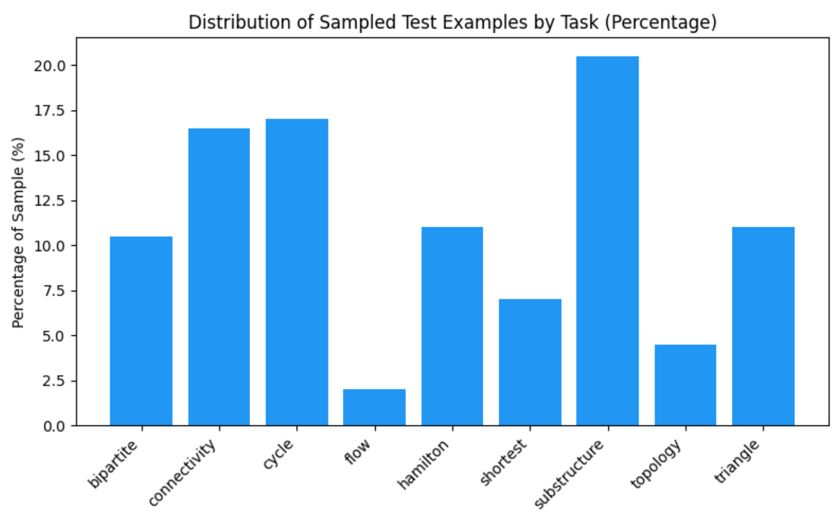


Figure 1: Distribution of sampled test examples by task. The dataset was stratified to ensure representation across binary, path-finding, and numeric tasks.

Our analysis revealed significant diversity in how questions must be answered (see Figure 2). We categorized the tasks into three distinct output types:

- **Binary (Blue):** Simple "Yes/No" questions (e.g., Connectivity, Cycle detection).
- **Numeric (Green):** Tasks requiring calculation (e.g., Shortest Path weight, Max Flow).
- **Other (Orange):** Tasks requiring path descriptions or node lists (e.g., Topology, Hamiltonian Path).

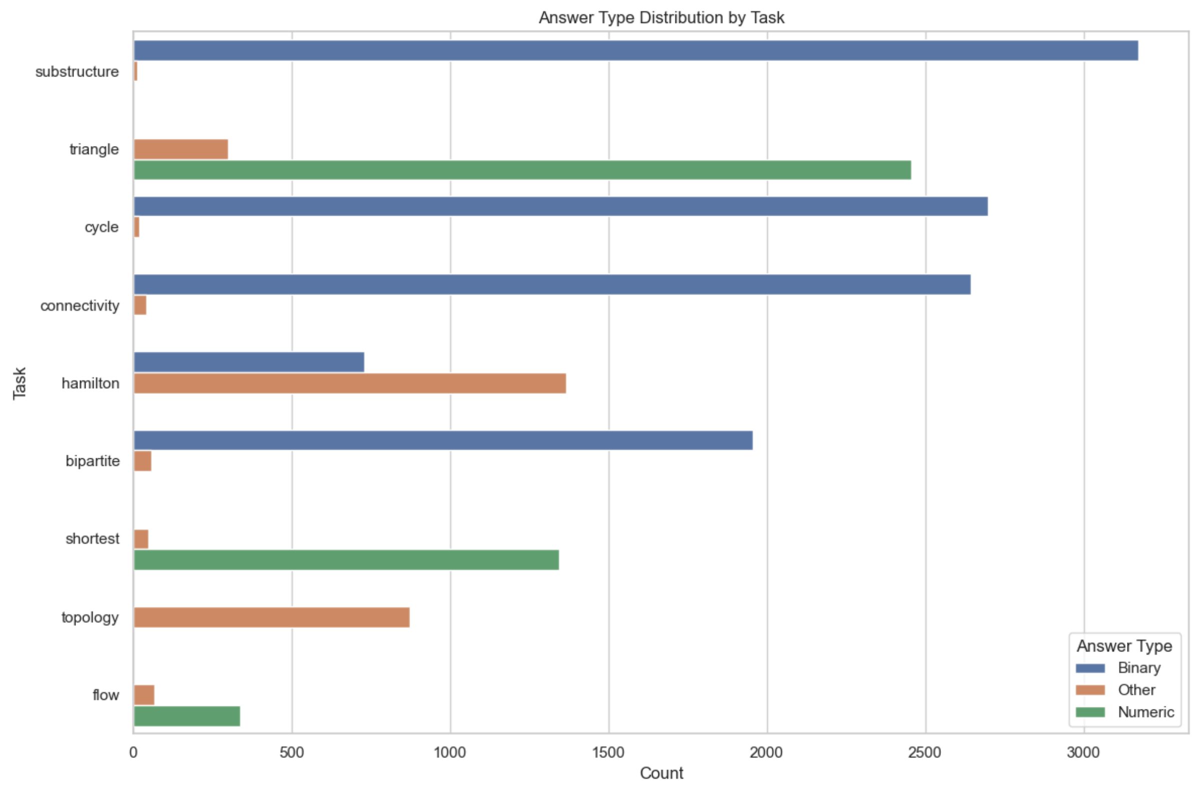


Figure 2: EDA on Answer Types. We analyzed the full dataset to categorize responses into Binary, Numeric, and Other. This distribution guided the design of our regex-based Parsers.

Based on this, we constructed a stratified test set of **200 samples** to ensure balanced evaluation across these logical categories.

4 Methodology

4.1 State management

The core of our system is the `GraphState`, a persistent memory structure passed between agents. It ensures that the context is preserved throughout the reasoning process.

```python
class GraphState(TypedDict):
    query: str
    task_type: str            # e.g., hamilton, triangle, flow
    partial_solution: str     # The solution constructed so far (e.g.,
   path 0->5->2)
    forbidden_moves: str      # Memory of past errors to avoid loops
    current_proposal: str     # The step being debated
    status: str               # SEARCHING, SOLVED, FAILED
    attempts: int             # Counter to manage retries
```

Listing 1: The GraphState definition in Python

4.2 Agent Architectures

We developed two distinct solver configurations to test different reasoning strategies.

4.2.1 1. Backtracking Solver (Baseline)

This architecture simulates a standard Depth-First Search (DFS) focused on efficiency.

- **Proposer Agent:** Analyzes the current state and suggests a single next move.
- **Verifier Agent:** Acts as a logic compiler. It checks the validity of the move against the edge list. It returns `VALID_STEP`, `INVALID`, or `VALID_BACKTRACK`.
- **Manager Agent (Logic Core):** Updates the `partial_solution` if valid. If invalid or a dead end is reached, it updates `forbidden_moves` and forces the Proposer to backtrack.

4.2.2 2. Consensus Solver (Debate)

To reduce hallucinations in complex tasks, we introduced parallelism and debate.

- **Three Proposers (A, B, C):** Instead of a single agent, three parallel instances generate proposals with different system prompts:
  - *Proposer A:* BFS Strategy (broad exploration).
  - *Proposer B:* DFS Strategy (deep exploration).
  - *Proposer C:* Heuristics (avoid past errors).
- **Arbiter Node:** A new agent that receives the three proposals. It evaluates them based on progress towards the goal and novelty, selecting the single best move to pass to the Verifier.

4.3 The Parser

A specialized Parser agent was implemented to normalize the final output. It strips verbose reasoning (e.g., "I have found the path...") to return a clean format matching the dataset (e.g., "Yes"), enabling automated accuracy metrics.

5 Experimental setup

5.1 Models

We tested both architectures on two different models to check for generalizability:

- **Ministral-8b:** A highly efficient, low-latency model.
- **Llama-3.1-8b-instruct:** A model optimized for complex instruction following and reasoning.

5.2 Constraints and Limits

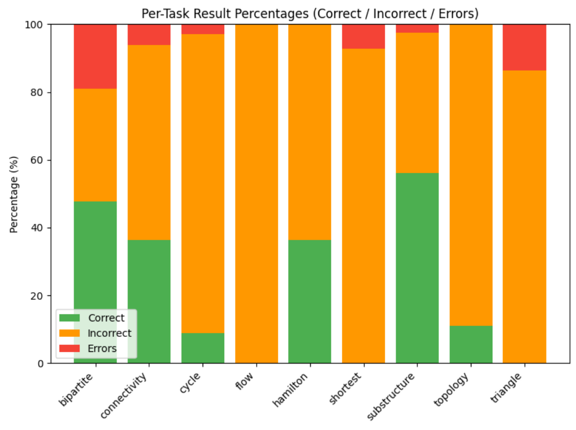
A critical discovery during experimentation was that fixed recursion limits do not work for variable graph complexities. We implemented dynamic limits:

- **Hamilton/Substructure:** 300 steps.
- **Connectivity:** 120 steps.
- **Triangle/Shortest Path:** 90 steps.

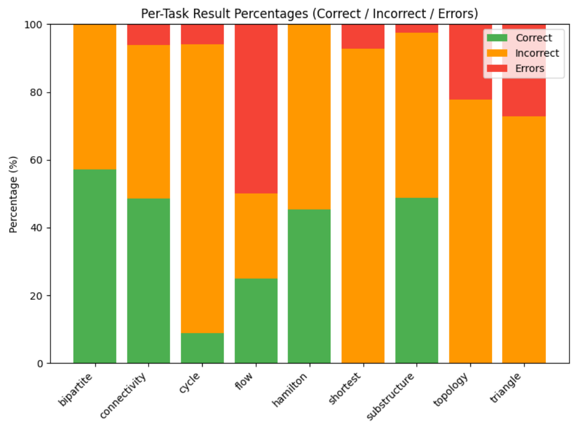
Any trajectory exceeding these limits is marked as an **Error** (Red in results).

6 Results

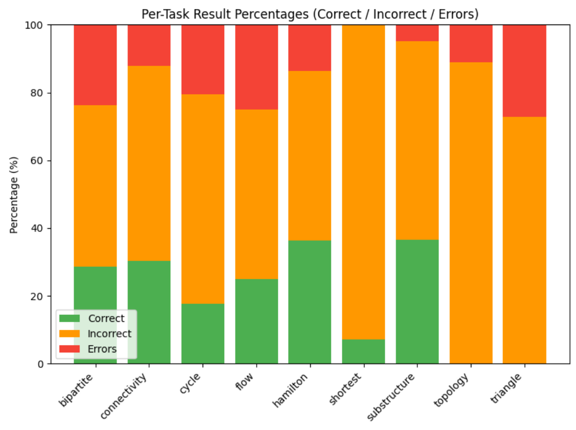
We present a comparative analysis of the Backtracking (Base) architecture versus the Consensus architecture across both models.



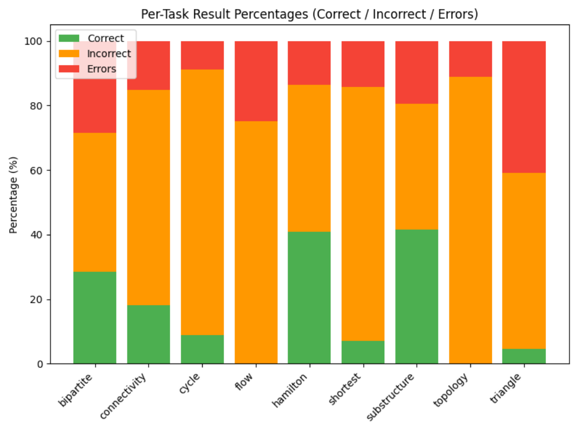
(a) Mistral - Base Architecture



(b) Mistral - Consensus Architecture



(c) Llama - Base Architecture



(d) Llama - Consensus Architecture

Figure 3: Per-Task results. Green represents Correct answers, Orange represents Incorrect answers (wrong path/value), and Red represents Errors (step limit exceeded or crash).

6.1 The cost of the consensus architecture

Comparing the left column (Base) to the right column (Consensus) in Figure 3, a clear trend emerges: **increased complexity leads to increased error rates** (red bars).

In the Base architecture, a single step requires 2-3 LLM calls (Proposer → Verifier → Manager). In the Consensus architecture, a single step requires many calls (3 Proposers → Arbiter → Verifier → Manager). Consequently, for structural tasks like *Connectivity* or *Substructure*, the Consensus architecture frequently hits the step limit before traversing the graph, resulting in a significantly higher failure rate than the simpler Backtracking solver.

6.2 Where consensus architecture shines

Despite the overhead, the Consensus architecture demonstrates utility in high-complexity combinatorial problems. In the Hamiltonian Path task (which requires visiting every node exactly once), the Backtracking solver can get stuck in dead ends. The Consensus mechanism, via the Arbiter and multiple search strategies, allows the system to make "smarter" moves. This is reflected in the retention of performance in the Hamilton category compared to the sharp drop in simpler tasks.

7 Conclusion

This project demonstrates that transforming an LLM into a state-aware agent improves reliability over zero-shot prompting, but adding more agents is not a "silver bullet."

Our key technical conclusions are:

1. **Architecture over Model:** For structural tasks (Substructure, Bipartite), a lean architecture (Backtracking) significantly outperforms a complex one, regardless of whether the underlying model is Ministral or Llama.
2. **The Latency Trade-off:** The Consensus architecture introduces a 4x computational overhead. This is only justifiable for tasks with deep local minima (like Hamiltonian paths). For standard traversal, the added latency leads to timeouts.
3. **Semantic vs. Numeric:** No amount of agentic reasoning fixed the inability to perform arithmetic on weighted edges.

7.1 Future Work

To bridge the gap with state-of-the-art systems like GraphWiz-DPO:

- **Hybrid routing:** Use Backtracking by default and dynamically switch to Consensus only when the Manager detects stagnation.
- **Context pruning:** Implement a mechanism to active "clean" the conversation history by removing obsolete intermediate reasoning steps. This will prevent saturation of the context window during long traversals of dense graphs, ensuring the model retains focus on the current state.

References

[1] Chen, Z., et al. (2024). *GraphWiz: An Instruction-Following Language Model for Graph Computational Problems*. arXiv preprint arXiv:2402.16029.

[2] Yao, S., et al. (2023). *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. arXiv preprint arXiv:2305.10601.

[3] Qian, C., et al. (2023). *ChatDev: Communicative Agents for Large-Scale Software Development*. arXiv preprint arXiv:2307.07924.