Beyond Trees: A DAG Memory Framework for Multi-Task Structured LLM Agents

Ye Ye

April 2025

Abstract

Recent advances in Large Language Models (LLMs) have enabled agents to perform complex multi-step tasks through autonomous reasoning and memory. While tree-based memory structures have been used to organize task flows hierarchically, they fall short in modeling real-world agent behaviors, where concurrent, interleaved, or dependent tasks often arise. Such scenarios require memory architectures that go beyond single-rooted, single-parent trees.

In this work, we introduce a DAG-based memory framework for structured LLM agents that supports multi-task execution, multi-parent dependencies, and flexible state inheritance. Our design allows task nodes to share substeps, converge on common goals, and support rollback or re-planning without disrupting coherent memory state. The architecture maintains acyclicity to preserve interpretability and consistency, while enabling richer inter-task interactions than traditional tree memory.

We illustrate our approach through case studies in travel planning, cooking, and interactive multi-turn tasks. Compared to tree-based and flat memory baselines, our DAG memory structure demonstrates improved contextual fidelity, reduced repetition, and better support for dynamic task management. We also present a prototype implementation and outline directions for further development and empirical evaluation.

1 Introduction

Large Language Models (LLMs) have emerged as powerful tools for autonomous agents, capable of performing multi-step tasks such as trip planning, web navigation, and software debugging. These tasks require the model to track progress, manage dependencies, and adapt flexibly to user changes. To support such capabilities, memory architectures play a central role.

Recent work has proposed structured memory systems [3, 1, 2], notably tree-based memory (e.g., Task Memory Tree in TME [4]), which enable LLMs to organize task steps hierarchically. These structures offer benefits in interpretability and local rollback. However, tree architectures inherently assume that each task node has only one parent, which restricts their ability to model shared dependencies, parallel paths, and convergent actions.

In real-world multi-step tasks, the assumption of strict tree hierarchy often breaks down. For example, a user may book a hotel and a flight separately, but both depend on the chosen destination. A cooking plan may require chopping vegetables and boiling water concurrently, both leading to a shared step of assembling the dish. These scenarios naturally call for a more general structure: a Directed Acyclic Graph (DAG).

In this paper, we propose a DAG-based memory architecture that extends tree-based structures with multi-parent support, enabling richer and more realistic task modeling for LLM agents. Our architecture preserves acyclicity to maintain clear temporal flow, while allowing nodes to inherit state from multiple predecessors. We further implement dynamic memory updates and node activation rules that maintain consistency across converging paths.

Through case studies in travel planning, cooking, and undo-retry tasks, we show that DAG memory improves coherence, reduces token repetition, and better captures the causal semantics of user commands.

[0]This is a preprint submitted to arXiv. Substantial updates, including experiments and refinements, are planned. Please do not cite this version as final.

We compare our system with baselines including ReAct, flat memory, and tree memory, and show consistent gains in both interpretability and execution efficiency.

Our contributions are:

- We identify the limitations of tree-based task memory and motivate the need for DAG structures in multi-step LLM agent tasks.
- We propose a novel DAG-based memory framework with support for multi-parent dependencies and dynamic state inheritance.
- We evaluate our approach through multi-domain task scenarios and demonstrate improved contextual stability and task coherence.

2 Method: DAG Memory Framework

Our system extends beyond ReAct's linear memory and Tree-of-Thoughts' branching search, offering DAG-style task consolidation and reusability [4].

2.1 Node Structure and Memory Update

Each task step is stored as a node in a directed acyclic graph (DAG), associated with contextual information (e.g., user prompt, tool invocation, or system response). Nodes can inherit memory states from one or more parent nodes, allowing shared dependencies and cross-task consistency.

Memory updates occur when a new user input arrives. The system determines the relationship between the input and existing nodes (e.g., continuation, parallel task, or rollback), and inserts a new node accordingly, updating references to affected subtrees or paths.

2.2 Multi-Parent Dependencies

Unlike trees, our DAG memory allows a node to have multiple parents, supporting converging tasks or shared subtasks. For example, a "confirm booking" node can be the endpoint of both a hotel and flight booking branch, while inheriting the destination context.

To maintain clarity, we prevent cycles and resolve state conflicts by prioritizing recently active branches and applying a topological traversal when rendering memory into prompt format.

2.3 Multi-Task and Interleaved Execution

Our framework enables agents to manage multiple concurrent tasks by allowing multiple roots and dynamically linking them when dependencies arise. This design supports task switching, background tasks, and cross-task subgoal reuse (e.g., querying nutrition while cooking).

We define task context scopes and allow the agent to reason about which nodes to activate, copy, or skip, maintaining consistency across parallel paths.

2.4 Cycle Detection and DAG Integrity

To ensure the memory structure remains a valid Directed Acyclic Graph (DAG), we introduce cycle detection during memory updates. Although our system supports multi-parent nodes and interleaved task flows, it explicitly prohibits the formation of cycles, which could lead to infinite reasoning loops, contradictory dependencies, or invalid execution orders.

Whenever a new edge is introduced—such as when a new task node references a prior node—we perform cycle detection via standard topological analysis. If a cycle is detected, we adopt one of the following strategies:

- **Reference Downgrade:** Convert the cyclic edge into a soft link that supports memory recall (read-only) but does not influence execution flow.

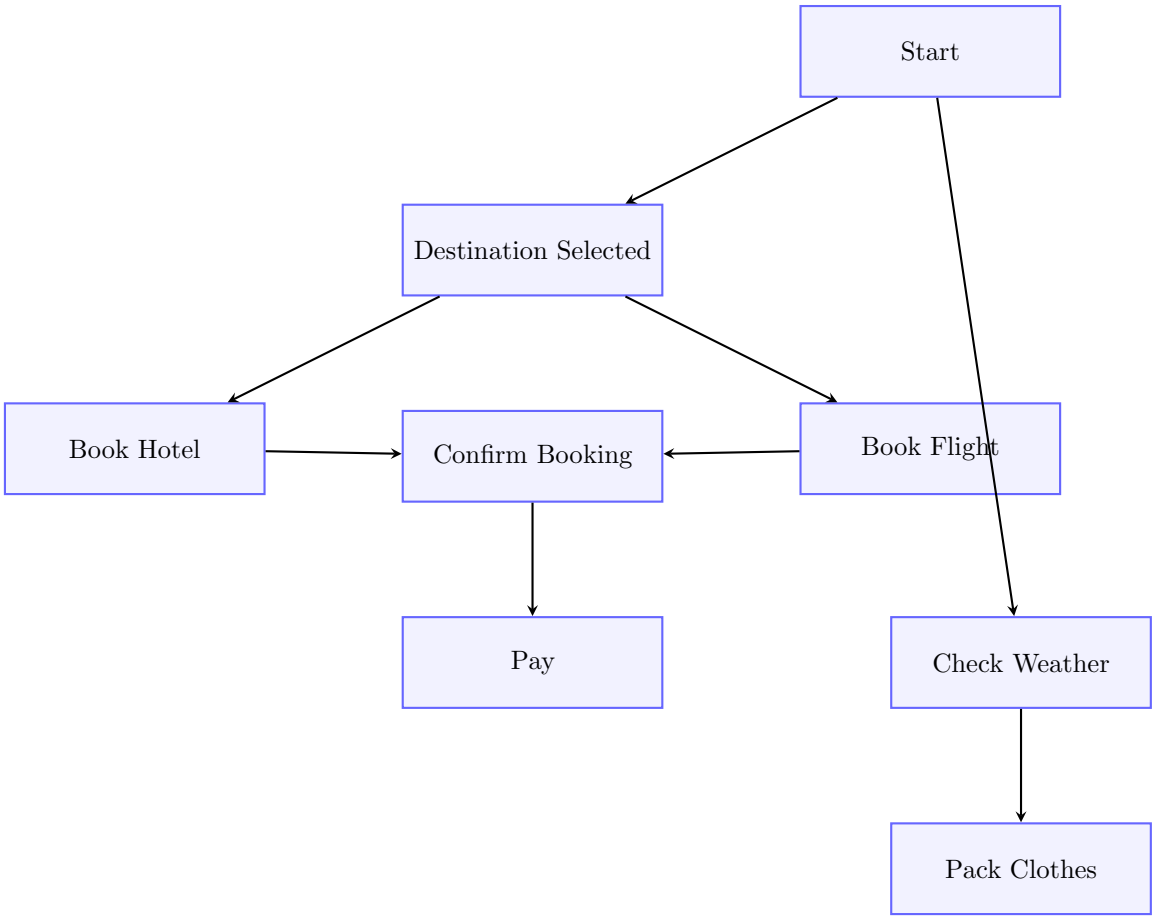

Figure 1: DAG memory structure with explicit directionality, showing task branches and convergence.

- **User Intervention:** Flag the cycle and prompt user validation for task disambiguation or manual edge removal.

This ensures that the memory graph remains acyclic and topologically traversable, preserving the correctness of agent-level task planning and rollback.

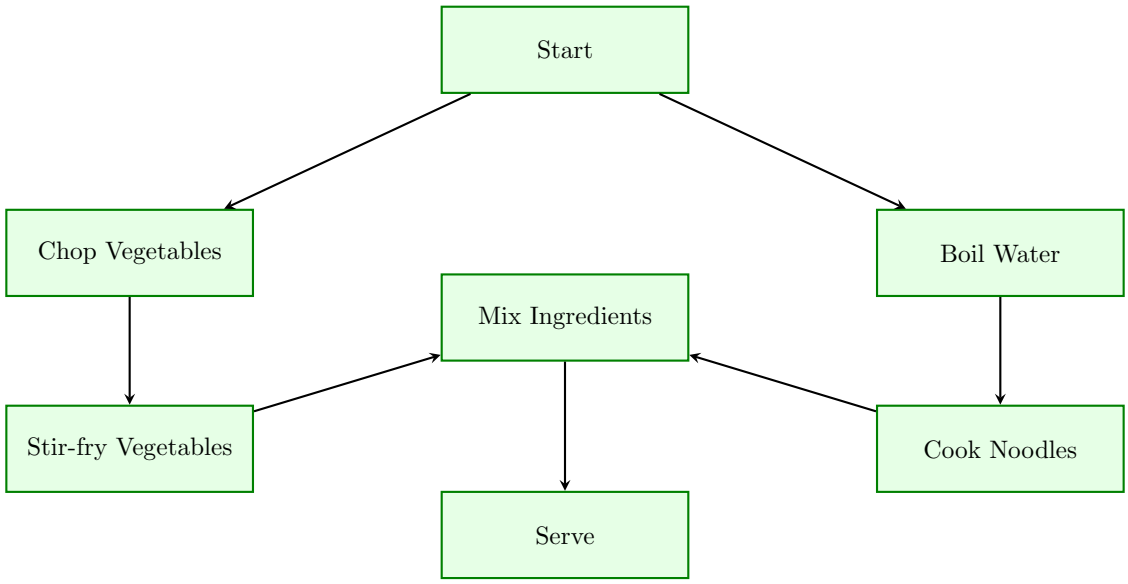

Figure 2: Cooking task modeled with DAG memory: two independent preparation paths (vegetables and noodles) converge for final serving.

3 Future Work

We are currently building a full-stack prototype of the DAG memory system to support interactive LLM agents. Future work will include integrating our architecture with real-world prompting agents, running benchmark comparisons across baseline systems (e.g., ReAct, flat memory), and developing a generalized memory planner to guide multi-task execution over graph structures.

References

[1] Fei Long et al. Memorybank: Enhancing llms with long-term memory. *arXiv preprint arXiv:2305.07519*, 2023.

[2] Jason Xu et al. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

[3] Shinn Yao, Jeffrey Zhao, Dian Yu, and et al. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[4] Ye Ye. Task memory engine (tme): A structured memory framework with graph-aware extensions for multi-step llm agent tasks, 2025.