Module 3: Multi-Agent Workflows and Parallelization in LangGraph

Introduction

This module builds on human-in-the-loop and memory concepts from Module 2. We will explore multi-agent workflows and develop a multi-agent research assistant that integrates concepts from all previous modules. To achieve this, we will first discuss LangGraph controllability topics, starting with parallelization.

Section 1: Fan Out and Fan In

- **Fan Out:** Distributing tasks across multiple parallel sub-graphs.
- **Fan In:** Combining results from parallel sub-graphs efficiently.
- When using fan out, ensure that a reducer is applied if multiple steps write to the same channel or key.
- Example: `operator.add` from Python's built-in `operator` module can concatenate lists when applied to them.

Section 2: Handling Parallel Execution with Reducers

- The `sorting_reducer` example sorts all values globally.
- Alternative approaches:
  - Write outputs to a separate field in the state during the parallel step.
  - Use a "sink" node after the parallel step to combine and order outputs.
  - Clear the temporary field after combining results.
- Each sub-graph runs in parallel, and since they return the same key, a reducer like `operator.add` is necessary.
- To avoid conflicts, create an **output state schema** for each sub-graph with different keys to store outputs.

Section 3: Map-Reduce for Efficient Task Processing

- **Map Phase:** Break a task into smaller sub-tasks, processing each in parallel.
- **Reduce Phase:** Aggregate results from all completed sub-tasks.

Section 4: Leveraging Send for Task Parallelization

- **Send** enables dynamic state passing to nodes without requiring alignment with the overall graph state.
- Example: Using `Send` to create a joke for each subject.
  - `generate_joke`: Name of the node in the graph.
  - `{"subject": s}`: State to send.
  - `Send` automatically parallelizes joke generation for multiple subjects.
- `generate_joke` utilizes its own internal state, which can be populated dynamically using `Send`.

Conclusion

This module provided insights into managing multi-agent workflows, parallel execution, and LangGraph controllability. By applying parallelization techniques like fan out/fan in, reducers, and map-reduce, we can efficiently distribute tasks and aggregate results dynamically. The `Send` function further enhances flexibility by enabling state-passing between nodes, supporting highly scalable workflow automation.