Analytics Vidhya

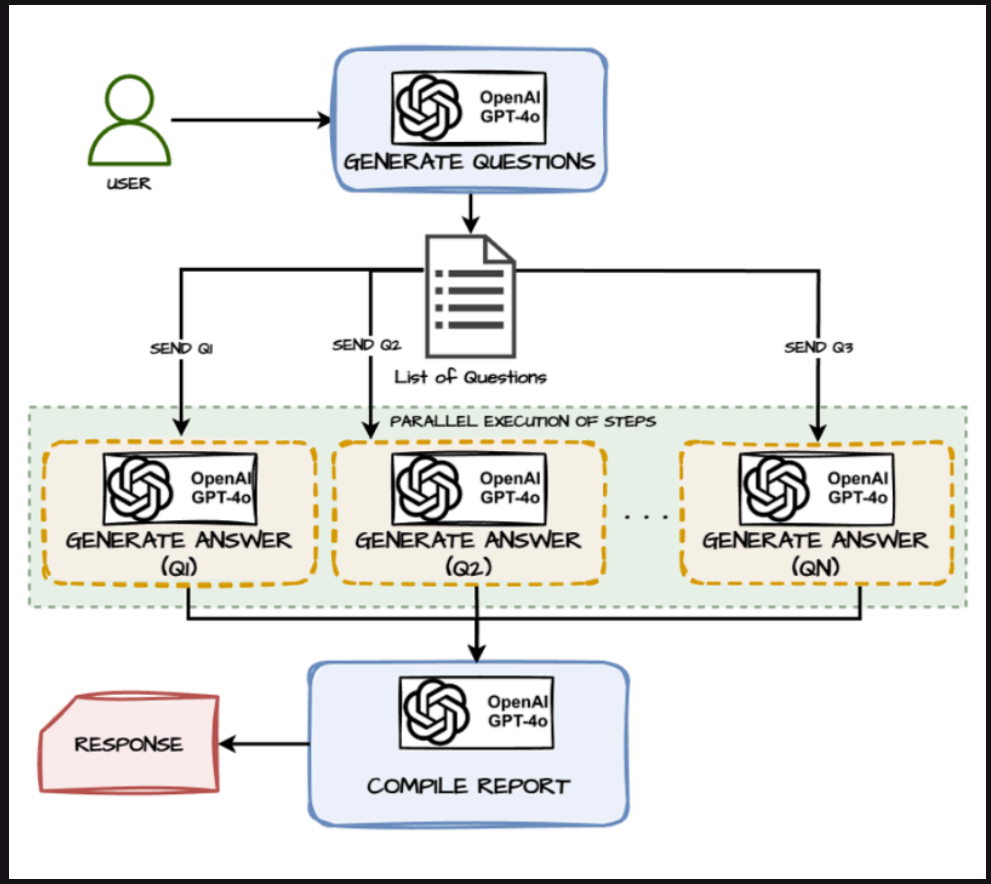
The Send construct in LangGraph - Dynamic Parallel Execution with Map-Reduce

Dipanjan Sarkar
Head of Community & Principal AI Scientist at Analytics Vidhya
Google Developer Expert - ML & Cloud Champion Innovator
Published Author

Parallel Steps Execution in LangGraph

LangGraph provides robust support for parallel execution of nodes, enhancing the efficiency and performance of graph-based agentic workflows.

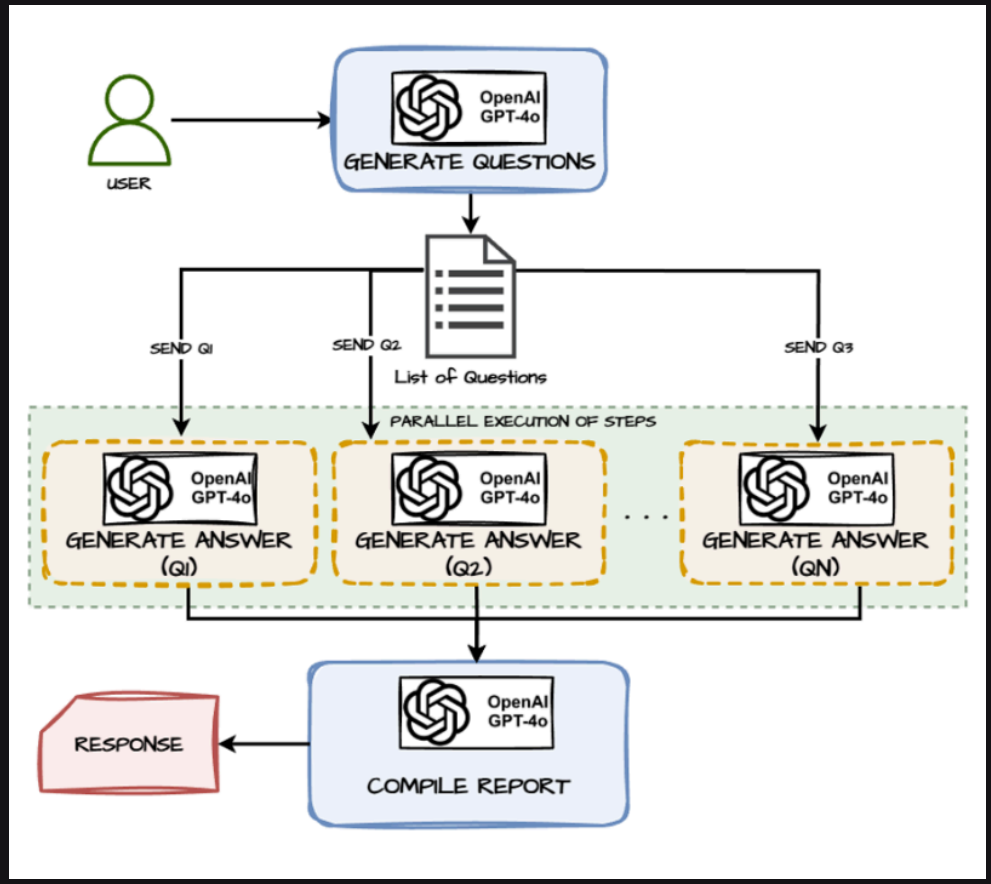
This parallelization is achieved through fan-out and fan-in mechanisms and can utilize standard edges or conditional edges.



Parallel Steps Execution in LangGraph

Map-Reduce in LangGraph

- Task Decomposition
  - Breaks down a large task into smaller, manageable sub-tasks (planning or complex question decomposition)
- Parallel Processing
  - Executes sub-tasks concurrently, significantly reducing overall processing time.
- Result Aggregation
  - Combines outcomes from all sub-tasks to form a comprehensive response.

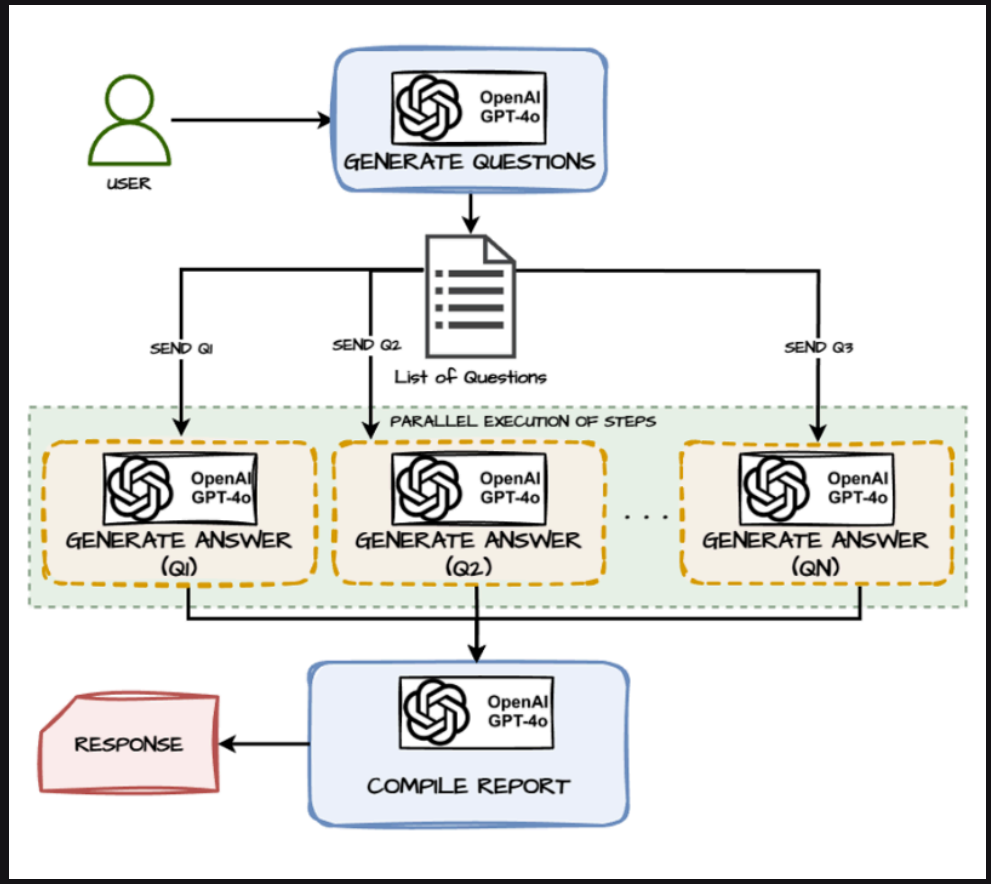


Analytics Vidhya

Parallel Steps Execution in LangGraph

Role of the Send Function

- Dynamic Task Distribution
  - Utilizes the Send function to dispatch different states to multiple instances of a node, facilitating parallel execution.
- Flexible Workflow Management
  - Very useful when you do not have a fixed number of static edges to parallelize like in router agent. A simple example would be generating a random number of questions or steps to solve a problem and parallelizing the generation process for each of those questions or steps.

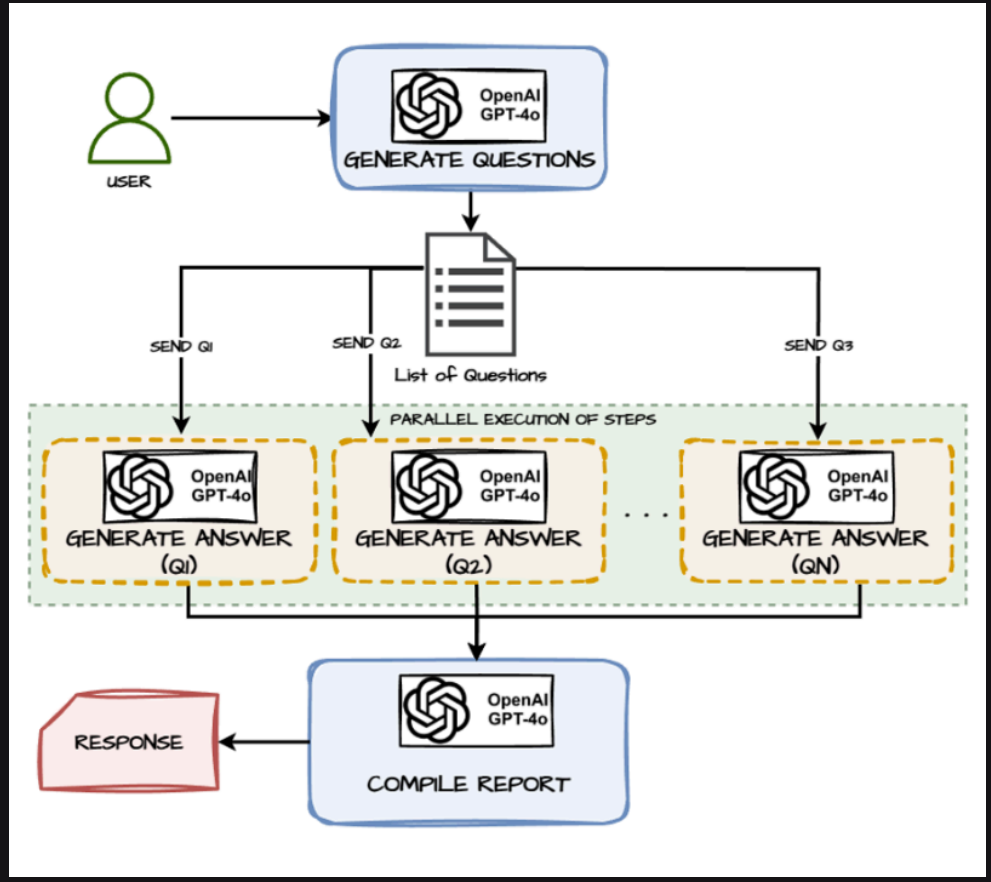


Analytics Vidhya

Parallel Steps Execution in LangGraph

Example Usage of Send Function

- Given a topic or complex question, use the generate_questions node to break it down into a list of sub-questions which can vary in number
- Send each question to an generate_answer node to search the web and generate answers in parallel
- The same generate_answer node is executed in parallel for all questions
- Compile all the questions and answers in the compile_report node and generate a comprehensive report



Analytics Vidhya

Example Usage of Send Function

- Here we use LangGraph to process a topic using the following steps:
  - Generate sub-questions from the main topic.
  - Generate answer for each question (in-parallel)
  - Combine them later (not shown here)

```python
questions_prompt = """Generate a list of concise sub-questions related
                      to this overall topic: {topic} which would help
                      build a good report.
                      Number of questions should be 3 for simple topics
                      and 5 for more complex topics
                    """
answer_prompt = """Generate the answer about {question}."""

# assume state variable is our graph state
# this variable stores all the questions and answers
def generate_questions(state: OverallState):
    prompt = questions_prompt.format(topic=state["topic"])
    response = llm.with_structured_output(Questions).invoke(prompt)
    return {"questions": response.questions}

# Node to generate answer to one question
def generate_answer(state: Answer):
    prompt = answer_prompt.format(question=state["question"])
    response = llm.with_structured_output(Answer).invoke(prompt)
    return {"answers": [{"question": state["question"],
                         "answer": response.answer}]}

# Node to call generate_answer multiple times (once for each question)
def continue_to_answers(state: OverallState):
    return [Send("generate_answer", {"question": q})
                for q in state["questions"]]
```

Example Usage of Send Function

- The generate_questions function creates a list of sub-questions based on the given topic.
- The generate_answer function generates an answer for a single question using an LLM.
- The continue_to_answers function dynamically connects the "generate_questions" node to the "generate_answer" node, once for each question (in parallel) using Send
- Instead of predefining edges, Send creates connections dynamically at runtime which is useful especially when the number of inputs to a node could vary (e.g multiple questions to answer that are not fixed in count)

```python
questions_prompt = """Generate a list of concise sub-questions related
                      to this overall topic: {topic} which would help
                      build a good report.
                      Number of questions should be 3 for simple topics
                      and 5 for more complex topics
                    """
answer_prompt = """Generate the answer about {question}."""

# assume state variable is our graph state
# this variable stores all the questions and answers
def generate_questions(state: OverallState):
    prompt = questions_prompt.format(topic=state["topic"])
    response = llm.with_structured_output(Questions).invoke(prompt)
    return {"questions": response.questions}

# Node to generate answer to one question
def generate_answer(state: Answer):
    prompt = answer_prompt.format(question=state["question"])
    response = llm.with_structured_output(Answer).invoke(prompt)
    return {"answers": [{"question": state["question"],
                         "answer": response.answer}]}

# Node to call generate_answer multiple times (once for each question)
def continue_to_answers(state: OverallState):
    return [Send("generate_answer", {"question": q})
                for q in state["questions"]]
```

Thanks