2025

Lesson 13

LangGraph: Memory, Parallelization, Filtering & Trimming

Memory

Introduction to Memory

- Memory plays an important part in maintaining context and enhancing the functionality of agentic systems.
- Persistent memory is the ability to retain information across multiple interactions.
- It allows the system to recall past states and decision which is essential for creating coherent and context aware conversation.
- LangGraph has checkpoint that is a specific memory implementation which saves the state of the graph at various points of execution.
- The state in a graph is dynamically changed when executed.
- Persistent memory and checkpoints provides a way to store and retrieval historical states, allowing more complex workflows and human-in-the loop interactions.

State vs Checkpoint

State

- Represent current context or working memory of the LLM application at any given time.
- Evolve as the node in the graph is executed. The state reflects the real-time changes and updates.
- State is used by the node to perform computation and decide the next step to perform in the workflow.

- Checkpoint
  - Checkpoint is a saved version of the state at a specific point in time during graph execution. It marks specific points in a workflow where the progress or outputs are saved.
  - Allows system to pause and resume execution, preserving the context for future interaction.
- Thread ID
  - A unique identifier associating specific thread of execution within the system.
  - Each thread represents a unique flow of interaction or conversation.

State vs Checkpoint

- Thread ID
  - The unique thread ID allows the system to manage multiple dialogues simultaneously.
- State is like your draft-in-progress.
- Checkpoint is like saving a backup copy at key milestone.

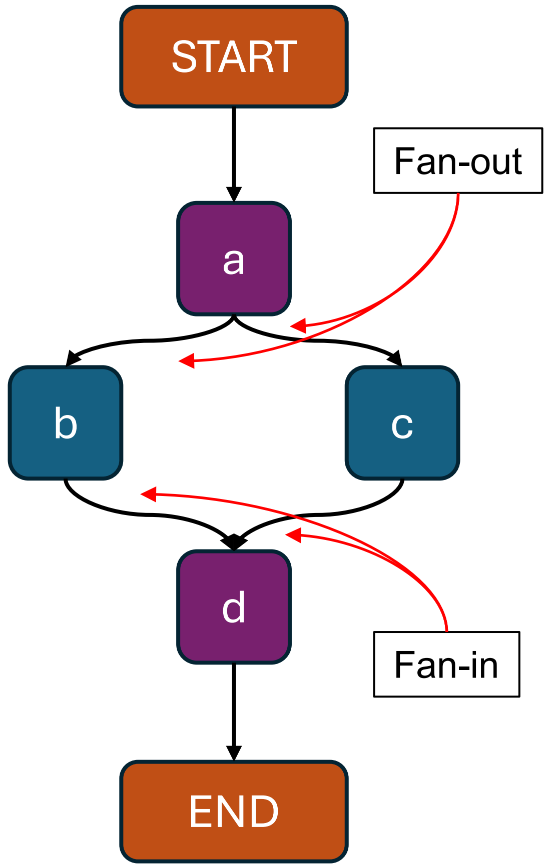
Parallelism

Parallel Node Execution

- Sequential execution is slower than parallel execution because tasks are executed and completed one after the other.
- In parallel execution, tasks are divided and independently executed simultaneously on multiple resources.
- For sequential execution in LangGraph, if one task is delayed, all subsequent task must wait.
- For parallel execution in LangGraph, other independent tasks can proceed even if one task is delayed, reducing overall waiting time.

Parallel node Fan-out and Fan-in

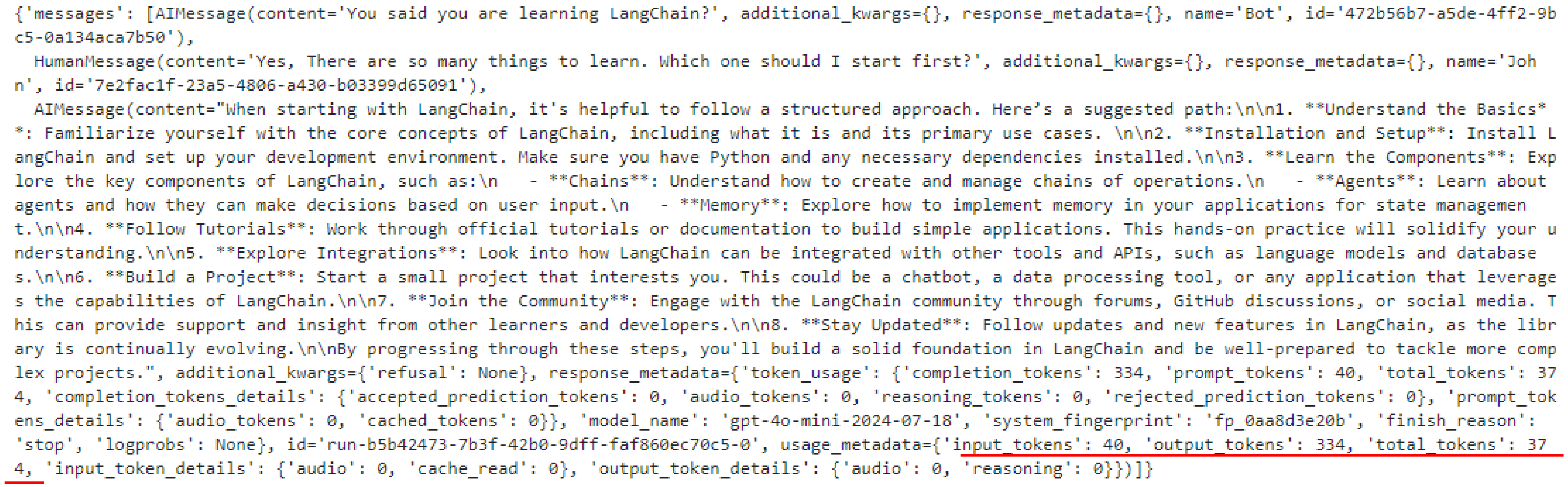
- The diagram shows fan-out and fan-in in a graph.
- Fan-out
  - Refers to the number of edges that originate from a given node and connect to other nodes (outgoing edges).
  - Indicates the extents to which a node influences or directs other nodes.
  - Fan-out of node ‘a’ is 2.
- Fan-in
  - Refer to the number of edges that terminate at a given node and come from other nodes (incoming edges).
  - Indicates the extent to which a node is influenced or dependent on other nodes.
  - Fan-in for node ‘d’ is 2.



Filtering and Trimming Messages

- Filtering and trimming messages are techniques used to optimize communication between components, like when handling conversations or interacting with large language models (LLMs).
- These techniques ensure messages are concise, relevant, and efficient, improving performance and maintaining meaningful context.
- Filtering
  - Purpose: Removes irrelevant or redundant information.
  - Why Important: Helps focus the LLM or other components on the most critical parts of the interaction.
- Trimming Messages
  - Purpose: Shortens the length of messages to fit within constraints, like token limits for LLMs.
  - Why Important: Many LLMs have token limits (e.g. 4096 or 8000 tokens for certain OpenAI models). Exceeding this limit can result in incomplete processing or errors.

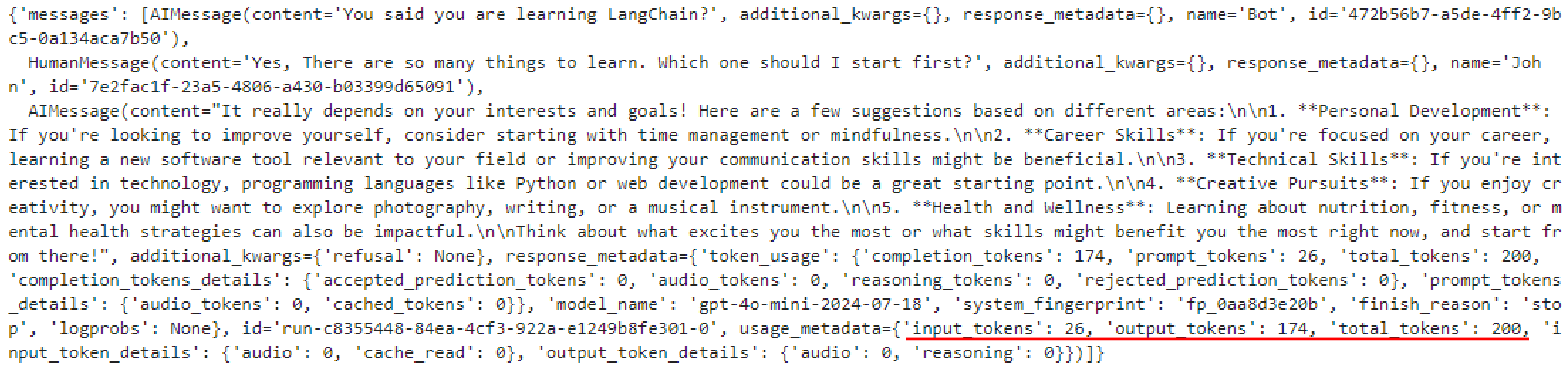
Trimming Observation



```
{'messages': [AIMessage(content='You said you are learning LangChain?', additional_kwargs={}, response_metadata={}, name='Bot', id='472b56b7-a5de-4ff2-9b
c5-0a134aca7b50'),
  HumanMessage(content='Yes, There are so many things to learn. Which one should I start first?', additional_kwargs={}, response_metadata={}, name='Joh
n', id='7e2fac1f-23a5-4806-a430-b03399d65091'),
  AIMessage(content="When starting with LangChain, it's helpful to follow a structured approach. Here’s a suggested path:\n\n1. **Understand the Basics*
*: Familiarize yourself with the core concepts of LangChain, including what it is and its primary use cases. \n\n2. **Installation and Setup**: Install L
angChain and set up your development environment. Make sure you have Python and any necessary dependencies installed.\n\n3. **Learn the Components**: Exp
lore the key components of LangChain, such as:\n   - **Chains**: Understand how to create and manage chains of operations.\n   - **Agents**: Learn about
agents and how they can make decisions based on user input.\n   - **Memory**: Explore how to implement memory in your applications for state managemen
t.\n\n4. **Follow Tutorials**: Work through official tutorials or documentation to build simple applications. This hands-on practice will solidify your u
nderstanding.\n\n5. **Explore Integrations**: Look into how LangChain can be integrated with other tools and APIs, such as language models and database
s.\n\n6. **Build a Project**: Start a small project that interests you. This could be a chatbot, a data processing tool, or any application that leverage
s the capabilities of LangChain.\n\n7. **Join the Community**: Engage with the LangChain community through forums, GitHub discussions, or social media. T
his can provide support and insight from other learners and developers.\n\n8. **Stay Updated**: Follow updates and new features in LangChain, as the libr
ary is continually evolving.\n\nBy progressing through these steps, you'll build a solid foundation in LangChain and be well-prepared to tackle more comp
lex projects.", additional_kwargs={'refusal': None}, response_metadata={'token_usage': {'completion_tokens': 334, 'prompt_tokens': 40, 'total_tokens': 37
4, 'completion_tokens_details': {'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_prediction_tokens': 0}, 'prompt_tok
ens_details': {'audio_tokens': 0, 'cached_tokens': 0}}, 'model_name': 'gpt-4o-mini-2024-07-18', 'system_fingerprint': 'fp_0aa8d3e20b', 'finish_reason':
'stop', 'logprobs': None}, id='run-b5b42473-7b3f-42b0-9dff-faf860ec70c5-0', usage_metadata={'input_tokens': 40, 'output_tokens': 334, 'total_tokens': 37
4, 'input_token_details': {'audio': 0, 'cache_read': 0}, 'output_token_details': {'audio': 0, 'reasoning': 0}})]}
```

Before trimming

'input_tokens': 40,
'output_tokens': 334,
'total_tokens': 374,
.
.
.}



```
{'messages': [AIMessage(content='You said you are learning LangChain?', additional_kwargs={}, response_metadata={}, name='Bot', id='472b56b7-a5de-4ff2-9b
c5-0a134aca7b50'),
  HumanMessage(content='Yes, There are so many things to learn. Which one should I start first?', additional_kwargs={}, response_metadata={}, name='Joh
n', id='7e2fac1f-23a5-4806-a430-b03399d65091'),
  AIMessage(content="It really depends on your interests and goals! Here are a few suggestions based on different areas:\n\n1. **Personal Development**:
If you're looking to improve yourself, consider starting with time management or mindfulness.\n\n2. **Career Skills**: If you're focused on your career,
learning a new software tool relevant to your field or improving your communication skills might be beneficial.\n\n3. **Technical Skills**: If you're int
erested in technology, programming languages like Python or web development could be a great starting point.\n\n4. **Creative Pursuits**: If you enjoy cr
eativity, you might want to explore photography, writing, or a musical instrument.\n\n5. **Health and Wellness**: Learning about nutrition, fitness, or m
ental health strategies can also be impactful.\n\nThink about what excites you the most or what skills might benefit you the most right now, and start fr
om there!", additional_kwargs={'refusal': None}, response_metadata={'token_usage': {'completion_tokens': 174, 'prompt_tokens': 26, 'total_tokens': 200,
'completion_tokens_details': {'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens': 0, 'rejected_prediction_tokens': 0}, 'prompt_tokens
_details': {'audio_tokens': 0, 'cached_tokens': 0}}, 'model_name': 'gpt-4o-mini-2024-07-18', 'system_fingerprint': 'fp_0aa8d3e20b', 'finish_reason': 'sto
p', 'logprobs': None}, id='run-c8355448-84ea-4cf3-922a-e1249b8fe301-0', usage_metadata={'input_tokens': 26, 'output_tokens': 174, 'total_tokens': 200, 'i
nput_token_details': {'audio': 0, 'cache_read': 0}, 'output_token_details': {'audio': 0, 'reasoning': 0}})]}
```

After trimming

{'input_tokens': 26,
'output_tokens': 174,
'total_tokens': 200,
.
.
.}

RP

Reducers

Reducer

- LangGraph uses state schemas and reducers to manage how the state is updated as it moves through the nodes.
- `operator.add` encountered earlier is a reducer. This annotation ensures that each time a node returns a message, it is appended to the existing list in the state. Without this operator, each new message would replace the previous value rather than being added to the list.

```
class State(TypedDict):
    state: Annotated[list, operator.add]
```

=

(similar)

```
class State(TypedDict):
    # Messages have the type "list". The `add_messages` function
    # in the annotation defines how this state key should be updated
    # (in this case, it appends messages to the list, rather than overwriting them)
    #
    # pre-built 'add_message' is also known as 'reducer'
    # reducer allows us to specify how stat updates are performed
    messages: Annotated[list[AnyMessage], add_messages]
```

Reducer: add_messages

`add_message (left: Messages, right: Messages, ...)`

- It merges two list of messages, updating existing message by ID.
- The left side is the base list.
- The right side is the list of messages to merge into the base list.
- There is a useful trick when working with add_message reducer.
  - If you pass a message with the same ID as an existing one in our message list, it will get over written.
  - This is to rewrite the “message” in the message list.

`RemoveMessage`

- To remove a message, we simply use RemoveMessage.

Activity

Activity

Memory

- LangGraph has checkpoint that is a specific memory implementation which saves the state of the graph at various points of execution. The state in a graph is dynamically changed when executed.
- Persistent memory and checkpoints provides a way to store and retrieval historical states, allowing more complex workflows and human-in-the loop interactions.

Parallelization

- Parallel node execution involves executing multiple nodes or tasks simultaneously rather than sequentially. This will be useful when tasks are independent of each other and can be processed in parallel to improve efficiency.

Filtering and Trimming

- Filtering: Removes irrelevant or redundant information from messages. This helps the LLM to focus on the most critical parts of the interaction.
- Trimming Messages: Many LLMs have token limits. Exceeding this limit can result in incomplete processing or errors. Shortens the length of messages to fit within constraints, like token limits for LLMs.

Challenge

30 mins

Activity 1

- We are building a Retrieval-Augmented Generation (RAG) application using an LLM to answer questions about Harry Potter the book "The Sorcerer’s Stone".
- Complete the code, construct the graph, and run tests to ensure functionality. Feel free to enhance the implementation.

Activity 2

- We will be using Wikipedia and a Google search services to gather contexts to answer a question directed to the LLM.
- You are to complete the code and build the graph. Run test on it to ensure that it is working. Feel free to improve it.

Reference

- LangGraph Glossary
  https://langchain-ai.github.io/langgraphjs/concepts/low_level/

Quiz 2

- https://forms.office.com/r/pSN2cj48qm

GenAI & LLM App Dev (2)

Thank you!