LangChain: Memory Component

Tracking State and Context in LLM Workflows

Overview

In LangChain, **Memory** enables LLM-based applications to **retain context** across multiple interactions. Since LLM API calls are inherently **stateless**, memory provides:

- **Conversational Continuity:** Keep track of previous messages for natural dialogue.
- **Context Summarization:** Condense older interactions to save tokens while preserving important information.
- **Custom State Management:** Store user-specific preferences, facts, or session data for personalized responses.

Memory ensures that your applications can behave intelligently and context-aware over time.

1. Core Memory Types

ConversationBufferMemory

Stores a transcript of recent messages.
- Suitable for short chats.
- Memory size grows quickly if conversation is long.

ConversationBufferWindowMemory

Keeps only the last N interactions.
- Avoids excessive token usage.
- Ideal for medium-length conversations where only recent context matters.

Summarizer-Based Memory

Periodically summarizes older chat segments.
- Condenses memory footprint.
- Maintains relevant context while discarding detailed old messages.

Custom Memory

Allows storing specialized state.
- Examples: user preferences, facts about users, session-specific data.
- Can be fully tailored to advanced application needs.

2. Memory Flow in a Conversation

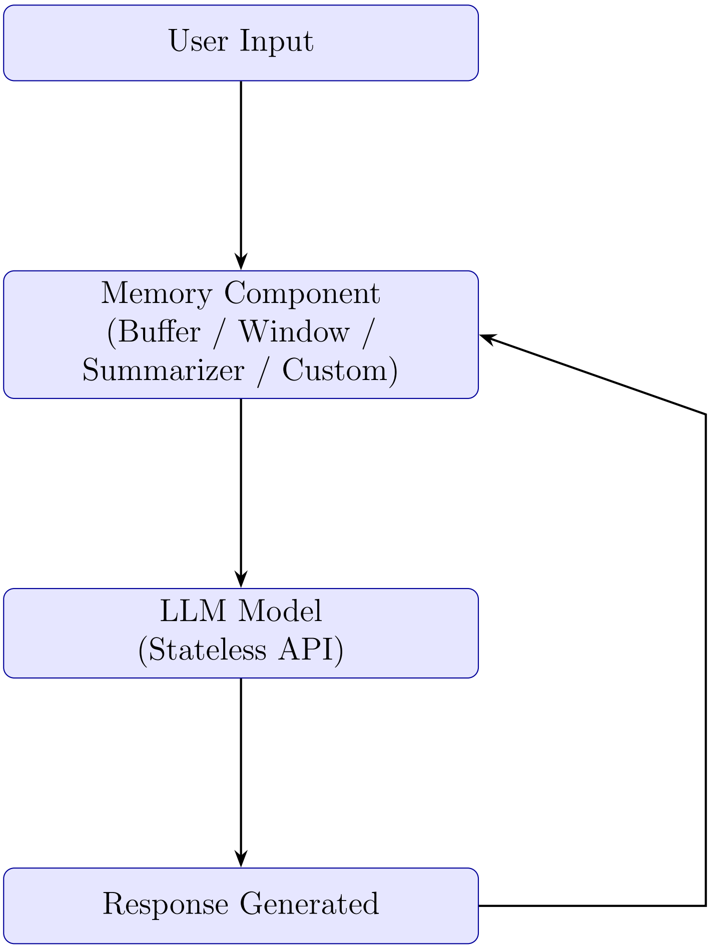
User Input
Memory Component (Buffer / Window / Summarizer / Custom)
LLM Model (Stateless API)
Response Generated

Figure 1: LangChain Memory Flow: Maintaining Context Across Interactions

Key Takeaways

- Memory enables LLMs to behave conversationally despite stateless API calls.
- Choose memory type based on context length, token limitations, and application requirements.
- Summarization and custom memory strategies help scale conversations efficiently.