让Agents更聪明，3招搞定短期记忆管理~

原创 猕猴桃 探索AGI 2025年02月12日 11:22 湖北

嘿，大家好！这里是一个专注于AI智能体的频道~

今天简单聊聊Agent的短期记忆管理，短期记忆大多数时候是指对话过程中产生的上下文信息。

一共3种短期记忆管理策略，内容可能会比较简单，主要是这2天在试玩一些开源的open deepreserach方案，发现langgraph新增的一些方法，用起来比较方便，分享给家人们~

目前的大多数chatbot系统都在做一件事，它们会存储对话历史中的每一句话。现在的大模型，大多数都会使用ChatML的prompt模板，可以天然的完成这个存储过程，自动的拼接到prompt中。

special-token syntax, similar to SQL injections:

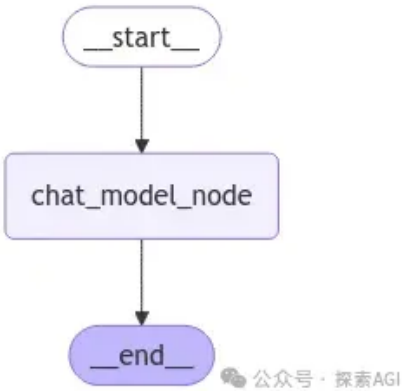
```
<|im_start|>system
You are ChatGPT, a large language model trained by OpenAI. Answer as concisely as possible.
Knowledge cutoff: 2021-09-01
Current date: 2023-03-01<|im_end|>
<|im_start|>user
How are you<|im_end|>
<|im_start|>assistant
I am doing well!<|im_end|>
<|im_start|>user
How are you now?<|im_end|>
```


但是，想象一下，当我们和其他人聊天时，似乎并不会记住对话中的每一个字、每一个标点符号。可能只会记住谈话的要点和关键信息。

所以目前这种常见的短期记忆的拼接，会带来三个明显的问题:

1. Token用量暴增 - 每次对话都要消耗大量token
2. 上下文窗口溢出 - 超出模型的处理上限
3. 响应延迟 - 需要处理过多历史信息

我们可以用langgraph轻松模拟出这样一个chatbot系统。




```
from langgraph.graph import MessagesState, START, END, StateGraph
```

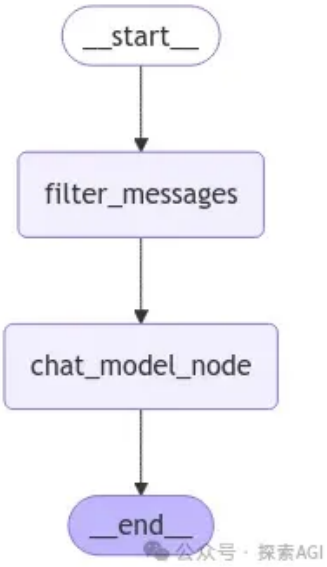
```
def chat_model_node(state: MessagesState):
    response = llm.invoke(state["messages"])
    return {"messages": response}

builder = StateGraph(MessagesState)

builder.add_node("chat_model_node", chat_model_node)

builder.add_edge(START, "chat_model_node")
builder.add_edge("chat_model_node", END)

graph = builder.compile()


graph_response = graph.invoke({"messages": messages})

for m in graph_response["messages"]:
    m.pretty_print()
```

RemoveMessage是langgraph中的一个最简单管理短期记忆的方法。简单来说，它就是只保留最近的N条对话记录。比如:

为了精简信息，这里可以只保留消息列表中的最后三条消息：

```
from langchain_core.messages import RemoveMessage

def filter_messages(state):
    messages = state["messages"]
    # 只保留最后3条消息
    messages = [RemoveMessage(m.id) for m in messages[:-3]]
    return {"messages": messages}

builder = StateGraph(MessagesState)

builder.add_node("filter_messages", filter_messages)
builder.add_node("chat_model_node", chat_model_node)

builder.add_edge(START, "filter_messages")
builder.add_edge("filter_messages", "chat_model_node")
builder.add_edge("chat_model_node", END)

graph = builder.compile()
```

插入一个filter_messages节点



这种方法简单有效,但可能会丢失一些重要的历史信息。并且每次输入prompt的轮次固定的，可能会超长，也可能会分成短。

因此，trim_messages 方法来了，仍然是只保留一定量的记忆历史，但是会计算出保留的历史信息符合，自行设定的特定tokens大小。

```
from langchain_core.messages import trim_messages

def chat_model_node(state: MessagesState):
    messages = trim_messages(
        allow_partial=True,   # 允许消息在中间部分部分拆分；这种方法可能会丢失上下文。
        strategy="last",   # 从哪里开始算，last最后开始
        max_tokens=100,  # 最大tokens数量
        token_counter=ChatOpenAI(model="gpt-3.5-turbo"),
        state=state["messages"]
    )
    response = llm.invoke(messages)
    return {"messages": response}

builder = StateGraph(MessagesState)

builder.add_node("chat_model_node", chat_model_node)

builder.add_edge(START, "chat_model_node")
builder.add_edge("chat_model_node", END)

graph = builder.compile()
```

最后是，动态摘要的方法。通过定期对对话历史进行总结,只保留关键信息。

代码也很简单，这里有个技巧是，当每一次积累到K轮才开始总结，避免过多的消耗，增加一个conditional边即可。

```
def summarize_conversation(state: State):
    summary = state.get("summary", "")
    if summary:
        summary_message = (
            f"This is summary of the conversation to date: {summary}\n\n"
            "Extend the summary by taking into account the new messages above:"
        )
    else:
        summary_message = "Create a summary of the conversation above:"

    messages = state["messages"] + [HumanMessage(content=summary_message)]
    response = model.invoke(messages)
    delete_messages = [RemoveMessage(id=m.id) for m in state["messages"][:-2]]
    return {"summary": response.content, "messages": delete_messages}

def should_continue(state: State) -> Literal["summarize_conversation", END]:
    messages = state["messages"]
    if len(messages) > 6:
        return "summarize_conversation"
    return END

workflow = StateGraph(State)

workflow.add_node("conversation", call_model)
workflow.add_node(summarize_conversation)

workflow.add_edge(START, "conversation")

workflow.add_conditional_edges(
    "conversation",
    should_continue,
)

workflow.add_edge("summarize_conversation", END)
```

好了，这就是我今天想分享的内容。如果你对构建AI智能体感兴趣，别忘了点赞、关注噢~ 😉

探索AGI

目前专注于大模型agent的产品落地方向，未来不确定~

119篇原创内容

公众号