
- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

ConversationKGMemory

- 대화 내용을 **지식 그래프(ConversationKnowledgeGraphMemory)** 라는 복잡한 네트워크 형태로 정리하여 저장하는 메모리

- **역할**:
  - 대화 속 여러 정보들의 관계를 파악하고 연결하여, 더 깊이 있는 대화를 가능하게 함
  - **모델이 서로 다른 개체 간 관계를 이해하는 데 도움을 줌**
  - **↳ 복잡한 인물 관계도를 그려놓은 메모**

- **LLM 사용 → 엔티티에 대한 정보 추출 → 시간이 지남에 따라 해당 엔티티에 대한 지식 축적**
  - **복잡한 연결망과 역사적 맥락을 기반으로 대응하는 능력을 향상시킴**

```
# 환경변수 처리 및 클라이언트 생성
from langsmith import Client
from langchain.prompts import PromptTemplate
from langchain.prompts import ChatPromptTemplate
from dotenv import load_dotenv

import os
import json

# 클라이언트 생성
api_key = os.getenv("LANGSMITH_API_KEY")
client = Client(api_key=api_key)


# LangSmith 추적 설정하기 (https:smith.langchin.com)
# LangSmith 추적을 위한 라이브러리 임포트
from langsmith import traceable

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if os.getenv('LANGCHAIN_A
org = "설정됨" if os.getenv('LANGCHAIN_ORGANIZATION') else "설

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: {langchain_api_key_status
    print("  -> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않았습니다. 다음을 확
    if langchain_tracing_v2 != "true":
        print(f"  - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

"@traceable" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

- 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
```

```
✅ LangSmith 프로젝트: 'LangChain-prantice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

- 대화 지식그래프 메모리 모듈 추출을 위한 임포트

```
from langchain.memory import ConversationKGMemory
```

- **LLM 생성하기**

```
import os
from dotenv import load_dotenv
import openai

# .env 파일에서 환경변수 불러오기
load_dotenv()

# 환경변수에서 API 키 가져오기
api_key = os.getenv("OPENAI_API_KEY")

# OpenAI API 키 설정
openai.api_key = api_key

# OpenAI를 불러오기
# ✅ 디버깅 함수: API 키가 잘 불러와졌는지 확인
def debug_api_key():
    if api_key is None:
        print("❌ API 키를 불러오지 못했습니다. .env 파일과 변수명을 확인하세요.")
    elif api_key.startswith("sk-") and len(api_key) > 20:
        print("✅ API 키를 성공적으로 불러왔습니다.")
    else:
        print("⚠️ API 키 형식이 올바르지 않은 것 같습니다. 값을 확인하세요.")

# 디버깅 함수 실행
debug_api_key()
```

- 셀 출력

```
✅ API 키를 성공적으로 불러왔습니다.
```

Chain에 메모리 활용하기

- **ConversationChain에 ConversationKGMemory를 메모리로 지정하여 대화를 나눈 후 memory 확인해보기**

```
from langchain.prompts.prompt import PromptTemplate
from langchain.chains import ConversationChain

llm = ChatOpenAI(temperature=0)

template = """The following is a friendly conversation between a human and an AI.
The AI is talkative and provides lots of specific details from its context.
If the AI does not know the answer to a question, it truthfully says it does not know.
The AI ONLY uses information contained in the "Relevant Information" section and does not hallucinate.

Relevant Information:

{history}

Conversation:
Human: {input}
AI:"""
```

```
prompt = PromptTemplate(
    input_variables=["history", "input"], template=template)

conversation_with_kg = ConversationChain(
    llm=llm, prompt=prompt, memory=ConversationKGMemory(llm=llm)
)

conversation_with_kg.predict(
    input="My name is Teddy. Shirley is a coworker of mine, and she's a new designer at our company."
)
```

- 셀 출력

```
/var/folders/h3/l7wnkv352kqftv0t8ctl2ld40000gn/T/ipykernel_39258/3115637315.py:28: LangChainDeprecationWarning: T
conversation_with_kg = ConversationChain(
```

- 셀 출력 내용
  - **ConversationChain 클래스 = LangChain 0.2.7 버전** 에서 **deprecated(사용되지 않음)** 경고가 나오는 클래스
  - 이후 **1.0 에서 제거될 예정 → 대신 RunnableWithMessageHistory 를 사용하는 방식으로 업데이트 필요**

업데이트된 방식

1) ConversationKGMemory

- 지식 추출: 대화에서 엔티티와 관계를 자동 추출
- 구조화: 지식을 그래프 형태로 저장
- 질의: 특정 엔티티에 대한 정보를 빠르게 검색 가능
- 한계: 단순한 체인 구조, 복잡한 워크플로우 처리 어려움

```
# 수동으로 메모리 관리하기
import os
from dotenv import load_dotenv
from openai import OpenAI
from langchain.memory import ConversationKGMemory
from langchain_openai import ChatOpenAI

# LLM과 메모리 초기화
# .env에서 API 키 로드
load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")

# LLM 생성하기
llm = ChatOpenAI(
    temperature=0,
    openai_api_key=api_key,
    model="gpt-4o-mini",
    )

memory = ConversationKGMemory(llm=llm)

# 첫 번째 대화 저장
memory.save_context(
    {"input": "My name is Teddy. Shirley is a coworker of mine, and she's a new designer at our company."},
    {"output": "Nice to meet you, Teddy! It's great to hear that Shirley is joining as a new designer. How are you finding wo
)

# 메모리 내용 확인
history = memory.load_memory_variables({"input": "Who is Shirley?"})
print(history)

# 지식 트리플 확인
triplets = memory.get_knowledge_triplets("Shirley is a designer")
print(triplets)
```

- 셀 출력 (4.2s)

```
{'history': 'On Shirley: Shirley is a coworker. Shirley is a new designer. Shirley works at the company.'}
[KnowledgeTriple(subject='Shirley', predicate='is a', object_='designer')]
```

---

2) LangGraph

- 상태 관리: 대화 전체를 상태로 관리하여 더 유연
- 워크플로우: 복잡한 대화 흐름 제어 가능
- 확장성: 여러 노드와 조건부 분기 지원
- 현대적: LangChain의 **최신 아키텍처** 반영

```
# 최신 LangGraph 방식
import os
from dotenv import load_dotenv
import uuid
from langchain_core.messages import HumanMessage, AIMessage
from langgraph.checkpoint.memory import MemorySaver
from langgraph.graph import START, MessagesState, StateGraph
from langchain_openai import ChatOpenAI

# LLM과 메모리 초기화
# .env에서 API 키 로드
load_dotenv()
api_key = os.getenv("OPENAI_API_KEY")

# LangGraph를 사용한 현대적 접근
def create_conversation_graph():
    # LLM 생성하기
    llm = ChatOpenAI(
        temperature=0,
        openai_api_key=api_key,
        model="gpt-4o-mini",
        )


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

    # 그래프 생성
    workflow = StateGraph(state_schema=MessagesState)
    workflow.add_edge(START, "model")
    workflow.add_node("model", call_model)

    # 메모리 설정
    memory = MemorySaver()
    app = workflow.compile(checkpointer=memory)

    return app

# 사용 예시
app = create_conversation_graph()
thread_id = str(uuid.uuid4())
config = {"configurable": {"thread_id": thread_id}}

# 대화 실행
input_message = HumanMessage(
    content="My name is Teddy. Shirley is a coworker of mine, and she's a new designer at our company."
)

for event in app.stream({"messages": [input_message]}, config, stream_mode="values"):
    event["messages"][-1].pretty_print()
```

- 셀 출력 (1.5s)

```
================================ **Human Message** ================================
```

```
My name is Teddy. Shirley is a coworker of mine, and she's a new designer at our company.
================================ **Ai Message** ================================

Hi Teddy! It's great to hear about your coworker Shirley joining the team as a new designer. How's she settling i
```

- *next: 대화 요약 메모리(ConversationSummaryMemory)*
