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

---

ConversationSummaryMemory

- **시간의 경과에 따른 대화의 요약 생성 = 길었던 대화를 요약 해서 저장 하는 메모리**
- 시간 경과 에 따른 대화의 **정보 를 압축하는 데 유용**
  - 대화 요약 메모리는 대화가 진행되는 동안 대화 를 요약 하고 현재 요약 을 메모리에 저장
  - → 지금까지의 **대화 요약 을 프롬프트 / 체인 에 삽입 가능**
  - 따라서, 과거 메시지 기록을 프롬프트에 그대로 보관하면 토큰을 너무 많이 차지할 수 있는 **긴 대화 에 가장 유용**

- **역할**:
  - 모든 대화 내용을 기억하는 대신, **핵심 내용만 간결하게 요약하여 저장**
  - **늑 길었던 회의 내용을 간단한 요약본으로 정리한 것**

```
# 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 요약 메모리 모듈 추출을 위한 임포트

```
from langchain.memory import ConversationSummaryMemory
```

- **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.
```

```
from langchain.memory import ConversationSummaryMemory
from langchain_openai import ChatOpenAI

memory = ConversationSummaryMemory(
    llm = ChatOpenAI(
        temperature=0,
        openai_api_key=api_key,
        model="gpt-4o-mini",
        ),
    return_messages=True)
```

- 셀 출력 (0.3s)

```
/var/folders/h3/l7wnkv352kqftv0t8ctl2ld40000gn/T/ipykernel_12653/1215817412.py:4: LangChainDeprecationWarning: Pl
memory = ConversationSummaryMemory(
```

- 셀 출력 의미

- `ConversationSummaryMemory`는 LangChain `v0.3.1`부터 **사용 중단 예정(deprecated) → `langchain==1.0.0` 버전에서 완전히 제거**
  - 즉, 이 모듈은 더이상 추천되지 않음 → **migration 가이드 참고할 것**
  - 이유: **LangChain은 현재 메모리 시스템이 더 유연한 LangGraph, LCEL 방식으로 업그레이드 중이기 때문**
- 공식 가이드
  - 마이그레이션 안내 가이드
  - 해당 메모리 마이그레이션 안내 가이드

- **최신 코드 migration 방법**
  - 기존의 `ConversationSummaryMemory` 대신 **LangGraph 사용 → 대화 요약 메모리 구현 가능**
  - 단계
    1. LangGraph 설치: `pip install langgraph`
    2. 요약 로직 = LangGraph에 통합
    3. **`trim_messages`로 토큰 제한 관리 = 기존의 `ConversationSummaryMemory` 대체**
  - `test`에 최신 코드 시도 예정

- 여러 대화 저장하기

```
memory.save_context(
    inputs={"human": "유럽 여행 패키지의 가격은 얼마인가요?"},
    outputs={
        "ai": "유럽 14박 15일 패키지의 기본 가격은 3,500유로입니다. 이 가격에는 항공료, 호텔 숙박비, 지정된 관광지 입장료가 포함되어 있습니다. 추가 비용은 
    },
)
memory.save_context(
    inputs={"human": "여행 중에 방문할 주요 관광지는 어디인가요?"},
    outputs={
        "ai": "이 여행에서는 파리의 에펠탑, 로마의 콜로세움, 베를린의 브란덴부르크 문, 취리히의 라이네폴 등 유럽의 유명한 관광지들을 방문합니다. 각 도시의 대표
    },
)
memory.save_context(
    inputs={"human": "여행자 보험은 포함되어 있나요?"},
    outputs={
        "ai": "네, 모든 여행자에게 기본 여행자 보험을 제공합니다. 이 보험은 의료비 지원, 긴급 상황 발생 시 지원 등을 포함합니다. 추가적인 보험 보장을 원하시면
    },
)
memory.save_context(
    inputs={
        "human": "항공편 좌석을 비즈니스 클래스로 업그레이드할 수 있나요? 비용은 어떻게 되나요?"
    },
    outputs={
        "ai": "항공편 좌석을 비즈니스 클래스로 업그레이드하는 것이 가능합니다. 업그레이드 비용은 왕복 기준으로 약 1,200유로 추가됩니다. 비즈니스 클래스에서는 
    },
)
memory.save_context(
    inputs={"human": "패키지에 포함된 호텔의 등급은 어떻게 되나요?"},
    outputs={
        "ai": "이 패키지에는 4성급 호텔 숙박이 포함되어 있습니다. 각 호텔은 편안함과 편의성을 제공하며, 중심지에 위치해 관광지와의 접근성이 좋습니다. 모든 호텔
    },
)
memory.save_context(
    inputs={"human": "식사 옵션에 대해 더 자세히 알려주실 수 있나요?"},
    outputs={
        "ai": "이 여행 패키지는 매일 아침 호텔에서 제공되는 조식을 포함하고 있습니다. 점심과 저녁 식사는 포함되어 있지 않아, 여행자가 자유롭게 현지의 다양한 음
    },
)
memory.save_context(
    inputs={"human": "패키지 예약 시 예약금은 얼마인가요? 취소 정책은 어떻게 되나요?"},
    outputs={
        "ai": "패키지 예약 시 500유로의 예약금이 필요합니다. 취소 정책은 예약일로부터 30일 전까지는 전액 환불이 가능하며, 이후 취소 시에는 예약금이 환불되지 
    },
```

)

- 처리 시간 (1m 34.5s)

```
print(type(memory.load_memory_variables({})["history"]))            # <class 'list'>

# 저장된 메모리 확인

print(memory.load_memory_variables({})["history"])
```

- 셀 출력_(첫번째)

```
[SystemMessage(content='The human asks about the price of a European travel package. The AI responds that the bas
```

- 셀 출력_(ver_ko) = content 내용을 한국어로 해석

```
[SystemMessage(content='고객이 유럽 여행 패키지의 가격을 묻자, AI는 14박 15일 일정의 기본 가격이 3,500유로이며 항공료, 호텔 숙박비, 지정
```

- 셀 출력_2 - 메모리가 꼬여서 두번째로 출력된 내용

```
[HumanMessage(content='유럽 여행 패키지의 가격은 얼마인가요?', additional_kwargs={}, response_metadata={}), AIMessage(cont
```

- memory.load_memory_variables({})["history"] 의 출력값은 SystemMessage 객체를 포함한 리스트이며, SystemMessage.content 에 실제 텍스트가 들어있음

```
# 가독성을 높인 출력

from langchain.schema import SystemMessage
from langchain.memory import ConversationBufferMemory

history = memory.load_memory_variables({})["history"]

for i, msg in enumerate(history, start=1):
    print("="*50)
    print(f"메시지 {i} - 타입: {msg.__class__.__name__}")
    print(msg.content)
    print()
```

- 셀 출력_1

```
==================================================
메시지 1 - 타입: SystemMessage
The human asks about the price of a European travel package. The AI responds that the basic price for a 14-night,
```

- 셀 출력_2 - 메모리가 꼬여서 두번째로 출력된 내용

```
==================================================
메시지 1 - 타입: HumanMessage
유럽 여행 패키지의 가격은 얼마인가요?

==================================================
메시지 2 - 타입: AIMessage
```

```
유럽 14박 15일 패키지의 기본 가격은 3,500유로입니다. 이 가격에는 항공료, 호텔 숙박비, 지정된 관광지 입장료가 포함되어 있습니다. 추가 비용은 선택

==================================================
메시지 3 - 타입: HumanMessage
여행 중에 방문할 주요 관광지는 어디인가요?

==================================================
메시지 4 - 타입: AIMessage
이 여행에서는 파리의 에펠탑, 로마의 콜로세움, 베를린의 브란덴부르크 문, 취리히의 라이네폴 등 유럽의 유명한 관광지들을 방문합니다. 각 도시의 대표적인

==================================================
메시지 5 - 타입: HumanMessage
여행자 보험은 포함되어 있나요?

==================================================
메시지 6 - 타입: AIMessage
네, 모든 여행자에게 기본 여행자 보험을 제공합니다. 이 보험은 의료비 지원, 긴급 상황 발생 시 지원 등을 포함합니다. 추가적인 보험 보장을 원하시면 상

==================================================
메시지 7 - 타입: HumanMessage
항공편 좌석을 비즈니스 클래스로 업그레이드할 수 있나요? 비용은 어떻게 되나요?

==================================================
메시지 8 - 타입: AIMessage
항공편 좌석을 비즈니스 클래스로 업그레이드하는 것이 가능합니다. 업그레이드 비용은 왕복 기준으로 약 1,200유로 추가됩니다. 비즈니스 클래스에서는 더 넓

==================================================
메시지 9 - 타입: HumanMessage
패키지에 포함된 호텔의 등급은 어떻게 되나요?

==================================================
메시지 10 - 타입: AIMessage
이 패키지에는 4성급 호텔 숙박이 포함되어 있습니다. 각 호텔은 편안함과 편의성을 제공하며, 중심지에 위치해 관광지와의 접근성이 좋습니다. 모든 호텔은 수

==================================================
메시지 11 - 타입: HumanMessage
식사 옵션에 대해 더 자세히 알려주실 수 있나요?

==================================================
메시지 12 - 타입: AIMessage
이 여행 패키지는 매일 아침 호텔에서 제공되는 조식을 포함하고 있습니다. 점심과 저녁 식사는 포함되어 있지 않아, 여행자가 자유롭게 현지의 다양한 음식을

==================================================
메시지 13 - 타입: HumanMessage
패키지 예약 시 예약금은 얼마인가요? 취소 정책은 어떻게 되나요?

==================================================
메시지 14 - 타입: AIMessage
패키지 예약 시 500유로의 예약금이 필요합니다. 취소 정책은 예약일로부터 30일 전까지는 전액 환불이 가능하며, 이후 취소 시에는 예약금이 환불되지 않습니
```

ConversationSummeryBufferMemory

- **두 가지 아이디어를 결합한 것**
  - **최근 대화내용의 버퍼를 메모리에 유지**
  - **이전 대화 내용을 완전히 플러시(flush)하지 않고 요약으로 컴파일** → 두 가지를 모두 사용

- **대화 내용** 을 플러시할 시기를 결정하기 위해 상호작용의 개수가 아닌 **토큰 길이** 사용

```
from langchain_openai import ChatOpenAI
from langchain.memory import ConversationSummaryBufferMemory
```

```
llm = ChatOpenAI(
    temperature=0,
    model="gpt-4o-mini",
)

memory = ConversationSummaryBufferMemory(
    llm = llm,
    max_token_limit=200,                                    # 요약의 기준이 되는 토큰 길이 설정
    return_messages=True,
)
```

- 셀 출력

```
/var/folders/h3/l7wnkv352kqftv0t8ctl2ld40000gn/T/ipykernel_12653/3116438127.py:9: LangChainDeprecationWarning: Pl
memory = ConversationSummaryBufferMemory(
```

- 먼저 **1개의 대화만 저장** → 메모리 확인해보기

```
memory.save_context(
    inputs={"human": "유럽 여행 패키지의 가격은 얼마인가요?"},
    outputs={
        "ai": "유럽 14박 15일 패키지의 기본 가격은 3,500유로입니다. 이 가격에는 항공료, 호텔 숙박비, 지정된 관광지 입장료가 포함되어 있습니다. 추가 비용은
    },
)
```

- 처리 시간 (2.6s)

- 메모리에 저장된 대화 확인해보기
  - **아직은 대화 내용을 요약하지 않음 → 기준이 되는 200 토큰에 도달하지 않았기 때문**

```
# 메모리에 저장된 대화내용 확인

memory.load_memory_variables({})["history"]
```

- 셀 출력

```
[HumanMessage(content='유럽 여행 패키지의 가격은 얼마인가요?', additional_kwargs={}, response_metadata={}),
AIMessage(content='유럽 14박 15일 패키지의 기본 가격은 3,500유로입니다. 이 가격에는 항공료, 호텔 숙박비, 지정된 관광지 입장료가 포함되어 있
```

- **대화 추가 저장 = 200 토큰 제한 넘김**

```
# 대화 추가하기
memory.save_context(
    inputs={"human": "여행 중에 방문할 주요 관광지는 어디인가요?"},
    outputs={
        "ai": "이 여행에서는 파리의 에펠탑, 로마의 콜로세움, 베를린의 브란덴부르크 문, 취리히의 라이네폴 등 유럽의 유명한 관광지들을 방문합니다. 각 도시의 대표
    },
)
memory.save_context(
    inputs={"human": "여행자 보험은 포함되어 있나요?"},
    outputs={
        "ai": "네, 모든 여행자에게 기본 여행자 보험을 제공합니다. 이 보험은 의료비 지원, 긴급 상황 발생 시 지원 등을 포함합니다. 추가적인 보험 보장을 원하시면
    },
)
memory.save_context(
    inputs={
```

```
        "human": "항공편 좌석을 비즈니스 클래스로 업그레이드할 수 있나요? 비용은 어떻게 되나요?"
    },
    outputs={
        "ai": "항공편 좌석을 비즈니스 클래스로 업그레이드하는 것이 가능합니다. 업그레이드 비용은 왕복 기준으로 약 1,200유로 추가됩니다. 비즈니스 클래스에서는 더
    },
)
memory.save_context(
    inputs={"human": "패키지에 포함된 호텔의 등급은 어떻게 되나요?"},
    outputs={
        "ai": "이 패키지에는 4성급 호텔 숙박이 포함되어 있습니다. 각 호텔은 편안함과 편의성을 제공하며, 중심지에 위치해 관광지와의 접근성이 좋습니다. 모든 호텔
    },
)
```

- 처리 시간 (8.7s)

- 저장된 내용 확인하기
  - **가장 최근 1개의 대화 ≠ 요약**
  - **이전 의 대화 내용 = 요약본 으로 저장**

```
# 메모리에 저장된 대화내용 확인

memory.load_memory_variables({})["history"]
```

- 셀 출력

```
[SystemMessage(content='The human asks about the price of a European travel package. The AI responds that the bas
HumanMessage(content='항공편 좌석을 비즈니스 클래스로 업그레이드할 수 있나요? 비용은 어떻게 되나요?', additional_kwargs={}, response
AIMessage(content='항공편 좌석을 비즈니스 클래스로 업그레이드하는 것이 가능합니다. 업그레이드 비용은 왕복 기준으로 약 1,200유로 추가됩니다. 비
HumanMessage(content='패키지에 포함된 호텔의 등급은 어떻게 되나요?', additional_kwargs={}, response_metadata={}),
AIMessage(content='이 패키지에는 4성급 호텔 숙박이 포함되어 있습니다. 각 호텔은 편안함과 편의성을 제공하며, 중심지에 위치해 관광지와의 접근성이
```

```
from langchain.schema import SystemMessage
from langchain.memory import ConversationBufferMemory

history = memory.load_memory_variables({})["history"]

for i, msg in enumerate(history, start=1):
    print("="*50)
    print(f"메시지 {i} - 타입: {msg.__class__.__name__}")
    print(msg.content)
    print()
```

- 셀 출력

```
==================================================
메시지 1 - 타입: SystemMessage
The human asks about the price of a European travel package. The AI responds that the basic price for a 14-night,

==================================================
메시지 2 - 타입: HumanMessage
항공편 좌석을 비즈니스 클래스로 업그레이드할 수 있나요? 비용은 어떻게 되나요?

==================================================
메시지 3 - 타입: AIMessage
항공편 좌석을 비즈니스 클래스로 업그레이드하는 것이 가능합니다. 업그레이드 비용은 왕복 기준으로 약 1,200유로 추가됩니다. 비즈니스 클래스에서는 더 넓

==================================================
메시지 4 - 타입: HumanMessage
패키지에 포함된 호텔의 등급은 어떻게 되나요?

==================================================
```

```
메시지 5 - 타입: AIMessage
이 패키지에는 4성급 호텔 숙박이 포함되어 있습니다. 각 호텔은 편안함과 편의성을 제공하며, 중심지에 위치해 관광지와의 접근성이 좋습니다. 모든 호텔은 수
```

업데이트된 코드

- *공식 가이드 사이트*
  - migration 가이드: ConversationSummaryMemory migration 상세 설명
  - LangGraph 메모리 how-to: 대화봇 메모리 최신 튜토리얼
  - API 레퍼런스: deprecation 상세 및 대체 방법

- 최신 코드로 시도해보기

```
# 필요한 모듈 임포트 (최신 LangChain에서 메모리 관리)
from typing import TypedDict
from langchain_core.messages import AIMessage, HumanMessage, SystemMessage, RemoveMessage, trim_messages
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_core.runnables import RunnablePassthrough
from langgraph.checkpoint.memory import MemorySaver
from langgraph.graph import START, MessagesState, StateGraph
from langsmith import traceable
import uuid                                                                    # 세션 ID 생성용


# 필요한 모듈 임포트
from dotenv import load_dotenv
import os
import json

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

# 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_API_KEY') else "설정되지 않음"      # API 키 값은 직접 출력하지 않음
org = "설정됨" if os.getenv('LANGCHAIN_ORGANIZATION') else "설정되지 않음"                      # 직접 출력하지 않음

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchain_project:
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{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 확인하세요:")
    if langchain_tracing_v2 != "true":
        print(f"  - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있지 않습니다 (현재: '{langchain_tracing_v2}').")
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")


from langchain.memory import ConversationSummaryBufferMemory
from langchain_openai import ChatOpenAI

# .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
```

```
# 1_LLM 초기화
llm = ChatOpenAI(
    temperature=0,
    model="gpt-4o-mini",
    openai_api_key=api_key,
)


memory = MemorySaver()


# State 확장 (기본 MessagesState에 summary 필드 추가 — 요약 저장용)

class State(TypedDict):
    messages: list                  # 대화 메시지 리스트 (기본 제공)
    summary: str                    # 요약 텍스트 저장 필드


# StateGraph 생성

workflow = StateGraph(state_schema=State)


# 요약 생성 함수 (기존 요약 + 새 메시지로 LLM 호출해 업데이트)
def update_summary(state: State):
    existing_summary = state.get("summary", "")             # 기존 요약 가져오기 (없으면 빈 문자열)
    messages = state["messages"]                            # 현재 대화 메시지 리스트

    # 요약 프롬프트 템플릿 (기존 요약 기반으로 새 요약 생성)
    #summary_prompt_text = f"""
    #Conversation summary so far: {existing_summary}
    #Extend the summary by taking into account these new messages:
    #"""

    #prompt = ChatPromptTemplate.from_messages([
    #    SystemMessage(content=summary_prompt_text),         # 요약 지시
    #    *messages                                           # 새 메시지 추가
    #])
    # LLM 으로 요약 갱신
    prompt = ChatPromptTemplate.from_messages(
        [SystemMessage(content=f"Conversation summary so far:\n{existing_summary}\n\nUpdate it."),
         *messages]
    )
    new_summary = (prompt | llm).invoke({}).content

    # 2) 최근 4개만 남기고 나머지 삭제
    keep_messages   = messages[-4:]              # 남길 메시지
    delete_messages = [RemoveMessage(id=m.id)    # 지울 메시지
                       for m in messages[:-4]]

    # LLM 호출로 새 요약 생성
    chain = prompt | llm
    summary_response = chain.invoke({})

    # 오래된 메시지 삭제 (최근 2개만 유지 — 필요 시 조정)
    #delete_messages = []
    #if len(messages) > 4:                                  # 4개로 늘려 최근 버퍼 보호
    #    delete_messages = [RemoveMessage(id=m.id) for m in messages[:-4]]  # 최근 4개 제외 삭제

    # 업데이트된 요약과 삭제 메시지 반환
    #return {"summary": summary_response.content, "messages": delete_messages}

    # 3) **남길 메시지 + 삭제 지시**를 함께 반환
    return {
        "summary":  new_summary,
        "messages": keep_messages + delete_messages
    }



# 그래프 노드·엣지 등록 (START → 요약 노드)

workflow.add_edge(START, "summary")
workflow.add_node("summary", update_summary)
                                                # <langgraph.graph.state.StateGraph at 0x119cf22c0>
```

```
<langgraph.graph.state.StateGraph at 0x10a7efe50>
```

```
# 메모리 설정 (MemorySaver로 체크포인트 저장)
memory = MemorySaver()
app = workflow.compile(checkpointer=memory)


# 세션 ID 생성 및 config 설정
thread_id = str(uuid.uuid4())
config = {"configurable": {"thread_id": thread_id}}


# 대화 턴
pairs = [
    ("커피 원두 종류가 뭐야?",
    "커피 원두는 크게 아라비카(Arabica), 로부스타(Robusta), 그리고 리베리카(Liberica)로 나뉘며, 각각의 품종은 고유한 맛과 향을 지니고 있습니다.  아라비카
    ("라떼 만드는 방법 알려줘.",
    "먼저 에스프레소를 한 샷 추출하는데, 이때 사용하는 원두는 아라비카처럼 부드럽고 향이 풍부한 것이 좋으며, 진한 맛을 원한다면 로부스타를 블렌딩해도 괜찮습니다.
    ("커피를 마시기 좋은 시간은?",
    "커피를 마시기에 가장 좋은 시간은 아침 기상 직후가 아니라, 코르티솔 수치가 안정되는 오전 9시 30분에서 11시 30분 사이입니다.  기상 직후에는 우리 몸이 이미 각
    ("카페 추천 메뉴 알려줘.",
    "카페에서 가장 인기 있는 메뉴로는 아이스 아메리카노가 단연 첫 손에 꼽히며, 깔끔하고 시원한 맛 덕분에 계절을 가리지 않고 많은 사람들이 즐깁니다.  그 뒤를 이어
    ("서울 강남구에 유명 카페 있어?",
    "서울 강남구에는 분위기와 맛을 모두 갖춘 유명 카페들이 많으며, 그중에서도 타짜도르(Tazza d'Oro)는 이탈리아 정통 에스프레소를 즐길 수 있는 곳으로, 진한 커피
    ("커피숍 예약 가능해?",
    "이들 카페는 대부분 예약보다는 시간대를 잘 선택해 방문하는 것이 추천되며, 특별한 이벤트나 단체 방문 시에는 사전 문의를 통해 가능 여부를 확인하는 것이 바람직합니
    ("테이크아웃 가능한 메뉴 알려줘.",
    "이들 카페 모두 테이크아웃이 가능하며, 각 매장마다 포장 방식이나 메뉴 구성에 차이가 있으므로 방문 전 확인하면 더욱 만족스러운 이용이 가능합니다. 서울 강남구에
]

# 대화 진행 예시 (pairs 리스트를 HumanMessage/AIMessage 쌍으로 변환해 그래프에 주입)
for human_txt, ai_txt in pairs:
    # 한 턴을 HumanMessage와 AIMessage 쌍으로 app.invoke에 보내 대화 기록으로 저장 (dict → Message 객체 변환)
    app.invoke(
        {"messages": [HumanMessage(content=human_txt), AIMessage(content=ai_txt)]},
        config                                                          # 세션 ID 포함 — 히스토리 유지
    )
```

- 실행 시간 (1m 47.1s)

```
# 최종 요약 + 최근 버퍼 확인
stored = app.get_state(config)
print("=== 최종 요약 ===")
print(stored.values["summary"],"\n")                                # 요약 출력

print("=== 최근 메시지 버퍼 (최근 4개) ===", "\n")
for m in stored.values["messages"]:
    print(m.content)                                                # 최근 메시지 원본 출력
```

- 셀 출력

```
=== 최종 요약 ===
각 카페에서 제공하는 테이크아웃 가능한 메뉴는 다음과 같습니다:

1. **타짜도르 (Tazza d'Oro)**: 이탈리아 정통 에스프레소와 다양한 커피 음료를 테이크아웃할 수 있습니다. 바쁜 출근길에 간편하게 즐길 수 있는 커피

2. **트리오드 (TRIODE)**: 다양한 음료와 함께 퍼먹는 케이크, 디저트가 테이크아웃 가능합니다. 특히 테이크아웃 컵에 담긴 라떼와 아메리카노가 많은 사

3. **베이커스트 브라운 (BAKEST BROWN)**: 다양한 베이커리와 음료를 테이크아웃할 수 있으며, 포장 시 고급스러운 패키지로 제공되어 선물용으로도 적

각 매장마다 메뉴와 포장 방식이 다를 수 있으니, 방문 전 미리 확인하는 것이 좋습니다.

=== 최근 메시지 버퍼 (최근 4개) ===

테이크아웃 가능한 메뉴 알려줘.
이들 카페 모두 테이크아웃이 가능하며, 각 매장마다 포장 방식이나 메뉴 구성에 차이가 있으므로 방문 전 확인하면 더욱 만족스러운 이용이 가능합니다. 서울
```

- *next: 벡터저장소 검색 메모리(VectorStoreRetrieverMemory)*
