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

---

3. 도구 호출 에이전트 *(Tool Calling Agent)* - ②

- **환경설정**

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv

# API 키 정보 로드
load_dotenv()                                # True
```

```
from langsmith import Client
from langsmith import traceable

import os

# 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 "설정되

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langc
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{langchain_tracing_v
    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: '{la
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

- 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.
```

```
# ========================================
# 경고 메시지 무시
# ========================================
import os
os.environ['TOKENIZERS_PARALLELISM'] = 'false'
```

```
import sys
from pathlib import Path

# 루트 디렉토리를 Python 경로에 추가
root_dir = Path().absolute().parent
sys.path.append(str(root_dir))

print(f"✅ 루트 디렉토리 추가: {root_dir}")
```

- 응답 시간: `0.0s`
- ✅ 루트 디렉토리 추가: `루트/20250727-langchain-note`

```
from langchain_google_genai import ChatGoogleGenerativeAI

# API 키 확인
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")

# LLM 초기화
gemini_lc = ChatGoogleGenerativeAI(
    #model="gemini-3-flash-preview",
    model="gemini-2.5-flash",
    temperature=1,                          # gemini-3-flash의 경우 temperature
    )

result=gemini_lc.invoke("대한민국의 수도는?")

# 테스트
print(result.content)
print(result.text)
```

- `gemini-3-flash-preview`셀 출력: `6.4s`

```
[{'type': 'text', 'text': '대한민국의 수도는 **서울**입니다.', 'extras': {'signature
```

```
대한민국의 수도는 **서울**입니다.
```

- `gemini-2.5-flash` 셀 출력 (`1.2s`)

```
대한민국의 수도는 **서울**입니다.
대한민국의 수도는 **서울**입니다.
```

---

6) 중간 단계 출력을 사용자 정의 함수로 출력

- 다음 3개의 함수를 정의 → 중간 단계 출력을 사용자 정의하기
  - **`tool_callback`**: 도구 호출 출력을 처리하는 함수
  - **`observation_callback`**: 관찰 (`Observation`) 출력을 처리하는 함수
  - **`result_callback`**: 최종 답변 출력을 처리하는 함수

```
from langchain.tools import tool
from typing import List, Dict, Annotated
from GoogleNews import GoogleNews
from langchain_experimental.utilities import PythonREPL


# 도구 생성
@tool
def search_news(query: str) -> List[Dict[str, str]]:
    """Search Google News by input keyword"""
    news_tool = GoogleNews()

    # 1. 검색 실행
    # 메서드 이름 변경하기: search_by_keyword -> search
    news_tool.search(query)


    # 2. 결과 반환 방식 변경하기: (search_by_keyword() -> result(
    # ERROR: return news_tool.search_by_keyword()[:5]
    return news_tool.result()[:5]
```

```
# 도구 생성
@tool
def python_repl_tool(
    code:Annotated[str, "The python code to execute to gener
):
    """Use this to execute python code. If you want to see t
    result = ""
    try:
        result = PythonREPL().run(code)
    except BaseException as e:
        print(f"Failed to execute. Error: {repr(e)}")
    finally:
        return result

print(f"① 도구 이름: {search_news.name}")
```

```
print(f"① 도구 설명: {search_news.description}")
print(f"② 도구 이름: {python_repl_tool.name}")
print(f"② 도구 이름: {python_repl_tool.description}")
```

- 셀 출력

```
① 도구 이름: search_news
① 도구 설명: Search Google News by input keyword
② 도구 이름: python_repl_tool
② 도구 이름: Use this to execute python code. If you want to see the output of
```

```
# tool 정의

tools = [search_news, python_repl_tool]
```

```
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.agents import create_agent

# LLM 정의
llm = gemini_lc

# 시스템 메시지 내용만 문자열로 정의
system_instruction = (
                    "You are a helpful assistant. "
                    "Make sure to use the `search_news` tool for searching key
                    )

# Agent 생성 (v1.0.5 기준)
agent = create_agent(
    model=llm,
    tools=tools,
    system_prompt=system_instruction,
    )
```

```
from langchain_core.messages import HumanMessage

# Agent 바로 실행하기
# 스트리밍 모드
result = agent.stream(
    {"messages": [HumanMessage(content="AI 투자와 관련된 뉴스를 검색해주세요.")]},
    config={"max_iterations": 10}
)

for step in result:
    # 중간 단계 출력
    print(step)
```

- `gemini-2.5-flash` 셀 출력 (`4.1s`)

```
{'model': {'messages': [AIMessage(content='', additional_kwargs={'function_ca
```

- `gemini-3-flash-preview` 셀 출력 (`12.0s`)

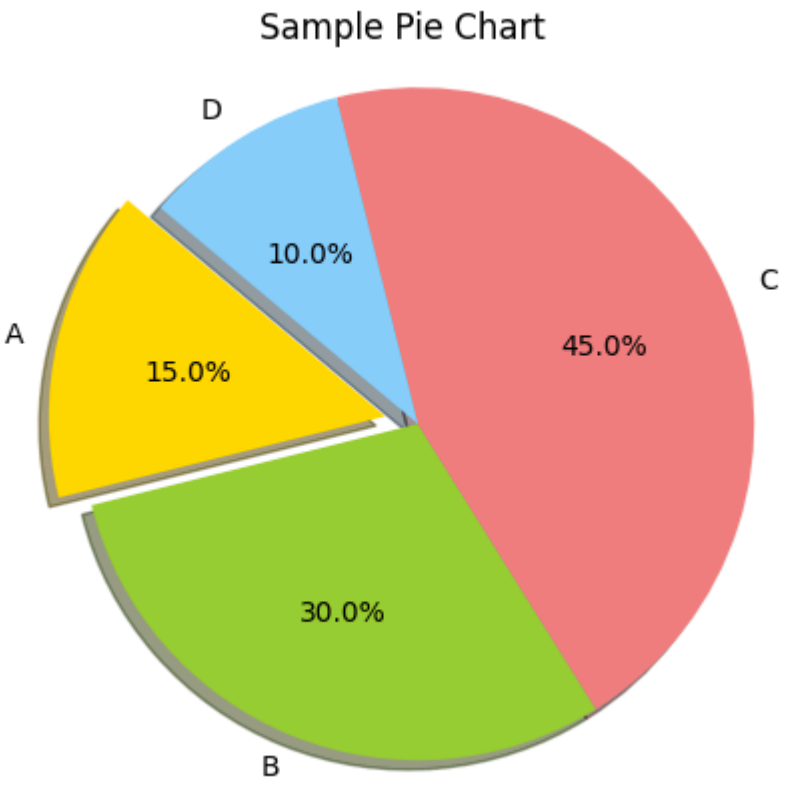
```
{'model': {'messages': [AIMessage(content=[], additional_kwargs={'function_ca
{'tools': {'messages': [ToolMessage(content='[{\'title\': "Qiming Venture Par
{'model': {'messages': [AIMessage(content=[], additional_kwargs={'function_ca
{'tools': {'messages': [ToolMessage(content='[{\'title\': \'2026 Investment C
{'model': {'messages': [AIMessage(content=[{'type': 'text', 'text': 'AI 투자와
```

```
from langchain_core.messages import AIMessage, ToolMessage

def parse_agent_stream(stream):
    """
    LangGraph Agent의 스트림 출력을 파싱하여 깔끔하게 보여주는 함수
    """
    for chunk in stream:
        # 1. 모델(LLM)의 응답 단계인 경우
        if "model" in chunk:
            model_messages = chunk["model"]["messages"]
            for message in model_messages:
                if isinstance(message, AIMessage):
                    # 도구 호출이 포함된 경우 (아직 최종 답변 아님)
                    if message.tool_calls:
                        tool_name = message.tool_calls[0]['name']
                        tool_args = message.tool_calls[0]['args']
                        print(f"🛠️ 도구 호출: {tool_name} (인자: {tool_args})")
                    # 최종 답변인 경우 (content가 있는 경우)
                    elif message.content:
                        print(f"🤖 AI 답변: {message.content}")

        # 2. 도구(Tool) 실행 결과 단계인 경우
        elif "tools" in chunk:
            tool_messages = chunk["tools"]["messages"]
            for message in tool_messages:
                if isinstance(message, ToolMessage):
                    print(f"✅ 도구 실행 완료 (결과 길이: {len(message.content)}자)"
```

```
# 질의에 대한 답변을 스트리밍으로 출력 요청
result = agent.stream(
    {"messages" : [HumanMessage(content="matplotlib 을 사용하여 pie 차트를 그리는 코드
    config={"recursion_limit": 10}
)

# 사용자 정의 파서 함수로 출력하기
# 함수에 스트림 객체를 통째로 전달
# 함수 안에서 for 루프를 돌면서 하나씩 꺼내 처리함
parse_agent_stream(result)
```

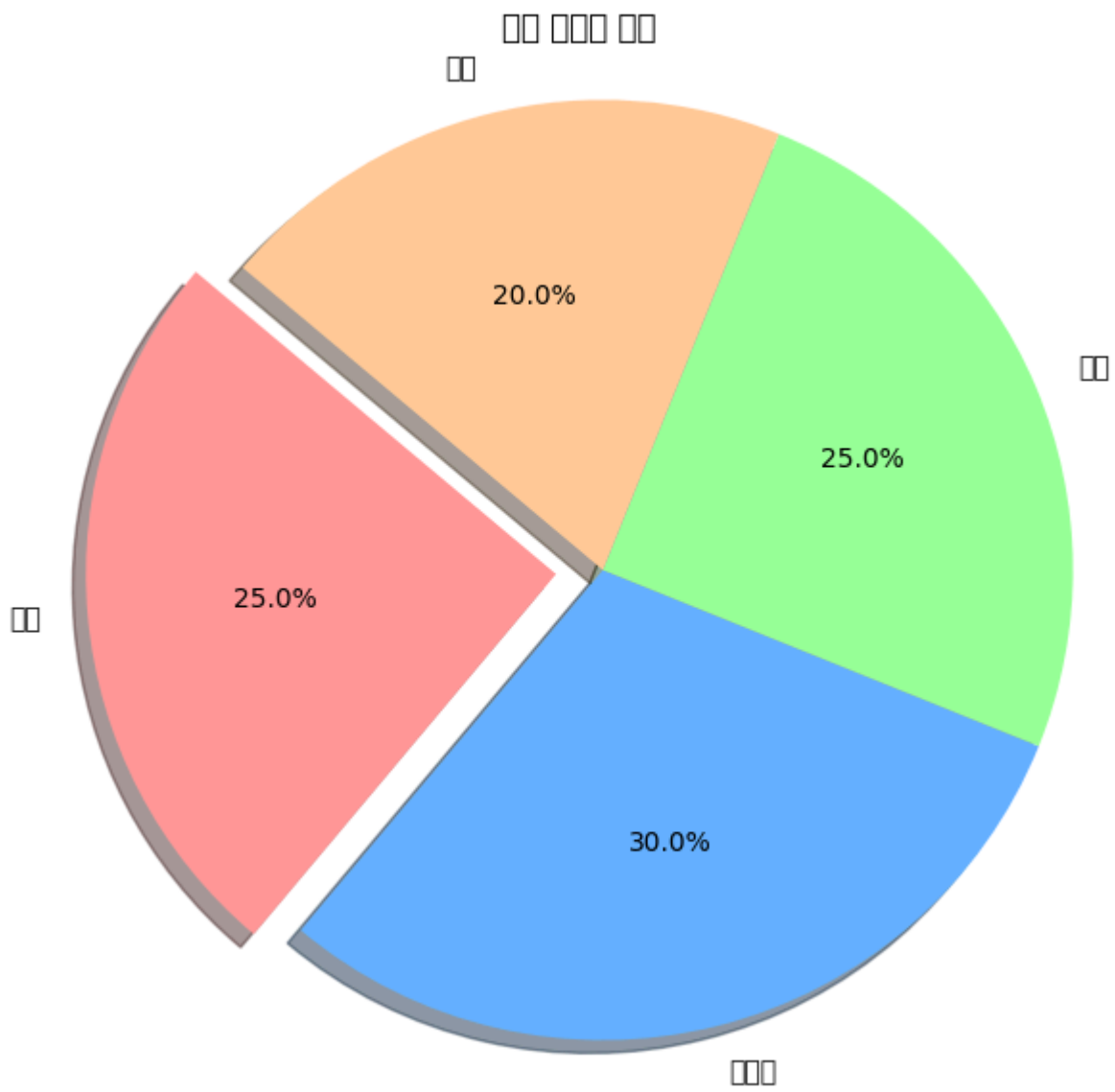
- `gemini-2.5-flash` 셀 출력 (`4.4s`)

🛠️ 도구 호출: python_repl_tool (인자: {'code': "\nimport matplotlib.pyplot as pl



- ✅ 도구 실행 완료 (결과 길이: 0자)
- 🤖 AI 답변: 알겠습니다. matplotlib을 사용하여 파이 차트를 그리는 코드를 작성하고 실행했습니다. 시각적인 차트가 표시되어야 합니다.

- 셀 출력 (`9.3s`)
  - WARNING:langchain_experimental.utilities.python:Python REPL can execute arbitrary code. Use with caution.
  - 🛠️ 도구 호출: python_repl_tool (인자: {'code': "import matplotlib.pyplot as plt\n\n# 데이터 준비\nlabels = ['사과', '바나나', '딸기', '포도']\nsizes = [25, 30, 25, 20]\ncolors = ['#ff9999','#66b3ff','#99ff99','#ffcc99']\nexplode = (0.1, 0, 0, 0) # 첫 번째 조각(사과)을 약간 떼어냄\n\n# 차트 그리기\nplt.figure(figsize=(7, 7))\nplt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',\n shadow=True, startangle=140)\n\n# 한글 폰트 설정 (환경에 따라 깨질 수 있어 기본 폰트 사용 시 주의 필요)\n# 여기서는 표준적인 matplotlib 코드를 제공합니다.\nplt.title('과일 선호도 조사')\nplt.axis('equal') # 파이 차트가 원형을 유지하도록 설정\n\nplt.show()"})



- ✅ 도구 실행 완료 (결과 길이: 0자)
- 🤖 AI 답변: [{'type': 'text', 'text': "`matplotlib`을 사용하여 파이 차트(Pie Chart)를 그리는 코드를 작성하고 실행했습니다.\n\n### 파이 차트 생성 코드\n\n`python\nimport matplotlib.pyplot as plt\n\n# 데이터 준비\nlabels = ['Apple', 'Banana', 'Strawberry', 'Grape'] # 항목\nsizes = [25, 30, 25, 20] # 비율\ncolors = ['#ff9999','#66b3ff','#99ff99','#ffcc99'] # 색상\nexplode = (0.1, 0, 0, 0) # 첫 번째 조각 강조 (돌출)\n\n# 차트 설정\nplt.figure(figsize=(7, 7))\nplt.pie(sizes, \n explode=explode, \n labels=labels, \n colors=colors, \n autopct='%1.1f%%', # 비율 표시 형식\n shadow=True, # 그림자 효과\n startangle=140) # 시작 각도\n\n# 차트 제목 및 비율 설정\nplt.title('Fruit Preference Survey')\nplt.axis('equal') # 파이 차트가 완벽한 원이 되도록 설정\n\n# 출력\nplt.show()\n`\n\n### 주요 코드 설명:\n1. **`labels` & `sizes`**: 차트에 표시할 항목의 이름과 각 항목이 차지하는 크기(비율)를 정의합니다.\n2. **`explode`**: 특정 조각을 원형에서 분리하여 강조하고 싶을 때 사용합니다. 위 코드에서는 첫 번째 항목인 'Apple'을 0.1만큼 밖으로 뺐습니다.\n3. **`autopct`**: 각 조각 위에 백분율을 자동으로 계산해서 표시해 줍니다 (`%1.1f%%`는 소수점 첫째 자리까지 표시하라는 의미입니다).\n4. **`startangle`**: 차트의 시작 각도를 정해줍니다. 기본값은 0도(3시 방향)이며, 각도를 조

정해 조각들의 배치 방향을 바꿀 수 있습니다.\n5. `plt.axis('equal')`: 원이 타원형으로 일그러지지 않고 정원이 되도록 설정합니다.", 'extras': {'signature': 'EpsCCpgCAXLI2nwJp3c7tbxN537F9oCkC2dFhTWJfHE9GeMegfz/JH6xlCTu823VsAJZYGFY+7P9uMkTsUPaevwgQkR1fy4Zw1W7TqrJRz6S09KFaZNb7UlOfwSzkaDUH+KMuZUimbm38gH9CMAb9Va3fzGY+9H+UCi1Q+h2dLXfged0FrkSmNqTCQ29ktSpojUNWFh51b8KW4bN8uFG9qt39fxyLQd/uos3S6NvQoqrH9lnB1H4rlelvsUp13vVlwRKTt0O3Jo+usVMYVrj0NBHeqHE/gELcBGSy4tVWDJtdsEt0Xo5nWyhBkbAtvVqu6U0aAbtnuJpeE8q8D7cned8HPec79H0DIVBhuuxSZrjLUq+rkXz0gZ9nhp7cg==''}}]

- `callback` 을 수정해 사용하는 방법

```
# 도구 호출 콜백 함수 정의하기

def tool_callback(tool) -> None:
    print("<<<<<<< 도구 호출 >>>>>>")
    # 사용된 도구의 이름을 출력하기
    print(f"Tool: {tool.get('tool')}")
    print("<<<<<<< 도구 호출 >>>>>>")
```

```
# 관찰 결과를 출력하는 콜백 함수 정의하기

def observation_callback(observation) -> None:
    print("<<<<<<< 관찰 내용 >>>>>>")
    print(
        f"Observation: {observation.get('observation')[0]}"
    )

    # 관찰 내용 출력하기
    print("<<<<<<< 관찰 내용 >>>>>>")
```

```
# 최종 결과를 출력하는 콜백 함수 정의하기
def result_callback(result: str) -> None:
    print("<<<<<<< 최종 답변 >>>>>>")
    print(result)
    print("<<<<<<< 최종 답변 >>>>>>")
```

```
# AgentCallbacks 객체를 생성하여 각 단계별 콜백 함수를 설정하기

from langchain_core.callbacks import BaseCallbackHandler

# 1. AgentCallbacks 클래스 직접 정의
class AgentCallbacks(BaseCallbackHandler):
    def __init__(self, tool_callback=None, observation_callback=None, result_c
        self.tool_callback = tool_callback
        self.observation_callback = observation_callback
        self.result_callback = result_callback

    # 도구가 실행될 때 호출 (tool_callback 연결)
```

```
    def on_tool_start(self, serialized, input_str, **kwargs):
        if self.tool_callback:
            # 기존 함수가 기대하는 딕셔너리 형태로 변환
            self.tool_callback({'tool': serialized.get('name'), 'tool_input':

    # 도구 실행이 끝났을 때 호출 (observation_callback 연결)
    def on_tool_end(self, output, **kwargs):
        if self.observation_callback:
            self.observation_callback({'observation': [str(output)]})

    # 에이전트/체인이 끝났을 때 호출 (result_callback 연결)
    # LangGraph 환경에서는 on_chain_end가 호출
    def on_chain_end(self, outputs, **kwargs):
        if self.result_callback:
            # LangGraph의 결과 = 보통 messages 리스트가 담긴 딕셔너리
            if isinstance(outputs, dict) and "messages" in outputs:
                last_message = outputs["messages"][-1]
                # 마지막 메시지가 AI의 답변이라면 출력
                if hasattr(last_message, "content") and last_message.content:
                    self.result_callback(last_message.content)
            # Legacy AgentExecutor의 경우 output 키에 결과가 담길 수 있음
            elif isinstance(outputs, dict) and "output" in outputs:
                self.result_callback(outputs["output"])
```

```
# 2. 객체 생성하기

agent_callbacks = AgentCallbacks(
    tool_callback=tool_callback,
    observation_callback=observation_callback,
    result_callback=result_callback,
)
```

```
# 1. 에이전트 실행 (이때 콜백들이 자동으로 작동하여 로그를 찍습니다)
# parse_agent_stream 함수는 사용하지 않음

result2 = agent.stream(
    {"messages": [HumanMessage
                (content="AI 투자와 관련된 뉴스를 검색해주세요.")]},
    config={
        "callbacks": [agent_callbacks],
        "max_iterations": 10
        }
    )

for step in result2:
    # 중간 단계 출력
    print(step)
```

- `gemini-2.5-flash` 셀 출력 (`5.5s`)

```
{'model': {'messages': [AIMessage(content='', additional_kwargs={'function_ca
<<<<<<< 도구 호출 >>>>>>>
Tool: search_news
```

```
<<<<<<< 도구 호출 >>>>>>
<<<<<<< 관찰 내용 >>>>>>
Observation: content='[{\'title\': \'The request / response that are contrary
<<<<<<< 관찰 내용 >>>>>>
<<<<<<< 최종 답변 >>>>>>
[{'title': 'The request / response that are contrary to the Web firewall secu
<<<<<<< 최종 답변 >>>>>>
{'tools': {'messages': [ToolMessage(content='[{\'title\': \'The request / res
<<<<<<< 최종 답변 >>>>>>
[{'type': 'text', 'text': 'AI 투자와 관련된 뉴스 검색 결과입니다:\n\n*   **소프트뱅크, A
<<<<<<< 최종 답변 >>>>>>
{'model': {'messages': [AIMessage(content=[{'type': 'text', 'text': 'AI 투자와
<<<<<<< 최종 답변 >>>>>>
[{'type': 'text', 'text': 'AI 투자와 관련된 뉴스 검색 결과입니다:\n\n*   **소프트뱅크, A
<<<<<<< 최종 답변 >>>>>>
```

- `gemini-3-flash-preview` 셀 출력 (`14.7s`)

```
{'model': {'messages': [AIMessage(content=[], additional_kwargs={'function_ca
```

```
<<<<<<< 도구 호출 >>>>>>
Tool: search_news
<<<<<<< 도구 호출 >>>>>>
<<<<<<< 관찰 내용 >>>>>>
Observation: content='[{\'title\': \'The request / response that are contrary
<<<<<<< 관찰 내용 >>>>>>
<<<<<<< 최종 답변 >>>>>>
[{'title': 'The request / response that are contrary to the Web firewall secu
<<<<<<< 최종 답변 >>>>>>
{'tools': {'messages': [ToolMessage(content='[{\'title\': \'The request / res
{'model': {'messages': [AIMessage(content=[], additional_kwargs={'function_ca
<<<<<<< 도구 호출 >>>>>>
Tool: search_news
<<<<<<< 도구 호출 >>>>>>
<<<<<<< 관찰 내용 >>>>>>
Observation: content="[{'title': 'AI Revolutionizes Drug Development, Halving
<<<<<<< 관찰 내용 >>>>>>
<<<<<<< 최종 답변 >>>>>>
[{'title': 'AI Revolutionizes Drug Development, Halving Timelines', 'media':
<<<<<<< 최종 답변 >>>>>>
{'tools': {'messages': [ToolMessage(content="[{'title': 'AI Revolutionizes Dr
{'model': {'messages': [AIMessage(content=[], additional_kwargs={'function_ca
<<<<<<< 도구 호출 >>>>>>
Tool: search_news
<<<<<<< 도구 호출 >>>>>>
<<<<<<< 관찰 내용 >>>>>>
```

```
Observation: content='[{\'title\': \'해외 축구 라이브 의 비밀을 풀다: 전문가들이 공유하는
<<<<<<< 관찰 내용 >>>>>>
<<<<<<< 최종 답변 >>>>>>
[{'title': '해외 축구 라이브 의 비밀을 풀다: 전문가들이 공유하는 핵심 팁', 'media': 'www.te
<<<<<<< 최종 답변 >>>>>>
{'tools': {'messages': [ToolMessage(content='[{\'title\': \'해외 축구 라이브 의 비
<<<<<<< 최종 답변 >>>>>>
[{'type': 'text', 'text': "최근 AI(인공지능) 투자와 관련된 주요 뉴스를 정리해 드립니다. 글
<<<<<<< 최종 답변 >>>>>>
```

---

7) 이전 대화내용을 기억하는 Agent

- `RunnableWithMessageHistory` 를 사용 → `AgentExecutor` 를 감싸줌
- `RunnableWithMessageHistory`

```
from langgraph.checkpoint.memory import MemorySaver

# 1. 메모리 저장소 생성 (In-memory)
memory = MemorySaver()
```

```
from langchain_community.chat_message_histories import ChatMessageHistory
from langchain_core.runnables.history import RunnableWithMessageHistory

# session_id 를 저장할 딕셔너리 생성
store = {}
```

```
# session_id 를 기반으로 세션 기록을 가져오는 함수

def get_session_history(session_ids):

    # session_id 가 store에 없는 경우
    if session_ids not in store:

        # 새로운 ChatMessageHistory 객체를 생성하여 store에 저장
        store[session_ids] = ChatMessageHistory()

    # 해당 세션 ID에 대한 세션 기록 반환
    return store[session_ids]
```

- 교재 속 코드 `ERROR` 발생

```
# 채팅 메시지 기록이 추가된 에이전트 생성하기
agent_with_chat_history = RunnableWithMessageHistory(
    agent,
    get_session_history,        # 대화 session_id
```

```
    # 프롬프트의 질문이 입력되는 key: "input"
    input_messages_key="input",
    # 프롬프트의 메시지가 입력되는 key: "chat_history"
    history_messages_key="chat_history",
)
```

- `LangGraph` 에이전트의 표준 메모리 방식인 `MemorySaver` *(Checkpointer)* 사용하는 코드로 수정하기

```
# 2. 메모리 기능이 추가된 에이전트 다시 생성
# checkpointer를 인자로 전달하면 자동으로 대화 기록이 연동됩니다. (LangGraph 기능)
agent_with_chat_history = create_agent(
    model=llm,
    tools=tools,
    system_prompt=system_instruction,
    checkpointer=memory,
)
```

- 교재 속 코드

```
result3 = agent_with_chat_history.stream(
    {"messages": [HumanMessage(content="안녕? 내 이름은 앨리스야.")]},
    config={"configurable": {"session_id": "abc123"}},
    )

for step in result3:
    # 중간 단계 출력
    print(step)
```

↓

- `gemini-2.5-flash` 셀 출력 (`1.1s`)

```
{'model': {'messages': [AIMessage(content='안녕하세요, 앨리스님! 무엇을 도와드릴까요?',
```

- `gemini-3-flash-preview` 셀 출력 (`2.4s`)

```
{'model': {'messages': [AIMessage(content=[{'type': 'text', 'text': '반가워요,
```

- or

```
# 질의에 대한 답변을 스트리밍으로 출력 요청
result3 = agent_with_chat_history.stream(
    {"messages": [HumanMessage(content="안녕? 내 이름은 앨리스야.")]},
    config={"configurable": {"session_id": "abc123"}},
    )

# 사용자 정의 파서 함수로 출력하기
# 함수에 스트림 객체를 통째로 전달
# 함수 안에서 for 루프를 돌면서 하나씩 꺼내 처리함
parse_agent_stream(result3)
```

↓

- `gemini-2.5-flash` 셀 출력 (`1.2s`)
  - 🤖 AI 답변: 안녕하세요 앨리스님! 무엇을 도와드릴까요?
- `gemini-3-flash-preview` *할당량 부족으로 실행* ❌

- 교재 속 `session_id`로 실행 → **`ERROR`**

```
# 질의에 대한 답변을 스트리밍으로 출력 요청하기

result3 = agent_with_chat_history.stream(
    {"messages": [HumanMessage(content="내 이름이 뭐라고?")]},
    # session_id 설정
    config={"configurable":{"session_id":"abc123"}},
    )

# 출력 확인하기
parse_agent_stream(result3)
```

↓

- 셀 출력 (`0.9s`)
  - 🤖 AI 답변: 저는 당신의 이름을 모릅니다.

- `session_id` ❌ → **`thread_id`** ⭕

```
# 3. 대화 실행 (thread_id 설정)

config = {"configurable": {"thread_id": "abc123"}}

result3 = agent_with_chat_history.stream(
    {"messages": [HumanMessage(content="안녕? 내 이름은 앨리스야.")]},
    config=config
```

```
    )
for step in result3:
    # 'model' 단계의 결과가 있을 때 텍스트만 출력
    if "model" in step:
        messages = step["model"]["messages"]
        for message in messages:
            if isinstance(message, AIMessage) and message.content:
                print(f"🤖 : {message.content}")
```

- `gemini-2.5-flash` 셀 출력 (`1.3s`)
  - 🤖 : 안녕하세요 앨리스님! 무엇을 도와드릴까요?
- *`gemini-3-flash-preview`* *할당량 부족으로 실행* ❌

```
# 4. 기억력 테스트
result4 = agent_with_chat_history.stream(
    {"messages": [HumanMessage(content="내 이름이 뭐라고?")]},
    # 동일한 thread_id 사용
    config=config
    )

for step in result4:
    if "model" in step:
        messages = step["model"]["messages"]
        for message in messages:
            if isinstance(message, AIMessage) and message.content:
                print(f"🤖 : {message.content}")
```

- `gemini-2.5-flash` 셀 출력 (`1.1s`)
  - 🤖 : 앨리스님이시죠! 제가 맞게 기억하고 있네요.
- *`gemini-3-flash-preview`* *할당량 부족으로 실행* ❌

- 다른 질문: *질의에 대한 답변을 스트리밍으로 출력 요청하기*

```
# 질의에 대한 답변을 스트리밍으로 출력 요청
result5 = agent_with_chat_history.stream(
    {"messages": [HumanMessage(
        content="내 이메일 주소는 alice@alice.com 이야. 회사 이름은 앨리스 주식회사야.")]
    # 동일한 thread_id 사용
    config=config
    )

# 출력 확인
for step in result5:
    if "model" in step:
        messages = step["model"]["messages"]
        for message in messages:
```

```
if isinstance(message, AIMessage) and message.content:
```

- `gemini-2.5-flash` 셀 출력 (`1.3s`)
  - 🤖 : 알겠습니다. 앨리스님의 이메일 주소는 alice@alice.com이고, 회사 이름은 앨리스 주식회사로 기억하겠습니다.
- *`gemini-3-flash-preview` 할당량 부족으로 실행* ❌

```
# 질의에 대한 답변을 스트리밍으로 출력 요청
result6 = agent_with_chat_history.stream(
    {"messages": [HumanMessage(
        content="Meta 관련 최신 뉴스 5개를 검색해서 제목을 나열해 이메일의 본문으로 작성해줘."
        "수신인에는 '셜리 상무님'을 적어주고, 발신인에는 내 인적정보를 적어줘."
        "정중한 어조로 작성하고, 메일의 시작과 끝에는 적절한 인사말과 맺음말을 적어줘.")]},
    # 동일한 thread_id 사용
    config=config
    )

# 출력 확인
for step in result6:
    if "model" in step:
        messages = step["model"]["messages"]
        for message in messages:
            if isinstance(message, AIMessage) and message.content:
                print(f"🤖 : {message.content}")
```

- `gemini-2.5-flash` 셀 출력 (`5.5s`)

```
🤖 : [{'type': 'text', 'text': "수신: 셜리 상무님\n\n안녕하십니까, 셜리 상무님.\n\n앨리스
```

- *`gemini-3-flash-preview` 할당량 부족으로 실행* ❌

- next: ***`04. Claude, Gemini, Ollama, Together.ai 를 활용한 Agent`***
