
- 출처: LangChain 공식 문서, 조코딩의 랭체인으로 AI 에이전트 서비스 만들기
- 깃허브 저장소 출처: https://github.com/sw-woo/hanbit-langchain

식당 리뷰 평가 AI 만들기

- *출처: 위에 표기*

```
# 환경변수 처리 및 클라이언트 생성
from langsmith import Client
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 "

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.
```

```
import os
from dotenv import load_dotenv
import openai

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

# 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.
```

SequentialChain

- **SequentialChain**
  - **여러 개의 LLM 체인 (혹은 다른 체인) 객체를 한 줄 로 엮어, 앞 단계 출력을 그 다음 단계의 입력으로 자동 전달하는 오케스트레이터**
  - **≒ 데이터 흐름을 명시적으로 관리해주는 컨베이터 벨트**
  - 정의한 순서 대로 체인 을 실행 하며 각 단계 결과를 키–값 딕셔너리 에 저장해 이어지는 단계에 주입
    - *개발자는 중간 데이터를 따로 수집하거나 전달하는 코드를 작성할 필요 없음*

```
from langchain.chains.llm import LLMChain
from langchain.chains.sequential import SequentialChain
```

LLM 초기화

- **temperature = 0.7 → 창의성이 높도록 설정**

```
# LLM 생성

llm = ChatOpenAI(
    temperature=0.7,
    openai_api_key=api_key,
    model="gpt-4o-mini",
    )
```

프롬프트 템플릿 정의하기

- **step1. 리뷰 요약 = Prompt1** : 리뷰를 한 문장으로 요약하는 작업
- **step2. 긍정/부정 점수 평가 = Prompt2** : 리뷰를 바탕으로 0점에서 10점 사이에서 점수를 매기는 작업
- **step3. 리뷰에 대한 공손한 답변 작성 = Prompt3** : 요약된 리뷰에 대해 공손한 답변을 작성하는 작업

LLM Chain

- **각 프롬프트 템플릿은 LLM Chain으로 구성 → 모델이 지정된 작업 수행**

```
from langchain_core.prompts import PromptTemplate
```

```
# 프롬프트 템플릿 설정(Prompot1)
prompt1 = PromptTemplate.from_template(
    "다음 식당 리뷰를 한 문장으로 요약하세요.\n\n{review}"
)

# 체인_1
chain1 = LLMChain(llm=llm, prompt=prompt1, output_key="summary")
```

- 셀 출력

```
/var/folders/h3/l7wnkv352kqftv0t8ctl2ld40000gn/T/ipykernel_13657/1921000103.py:10: LangChainDeprecationWarning: T
chain1 = LLMChain(llm=llm, prompt=prompt1, output_key="summary")
```

- 경고 메시지 해석
  - **이 기능은 곧 없어질 예정이니, 다른 방법으로 바꿔 쓰라는 의미**
    - LLMChain 클래스 = **LangChain 0.1.17 버전 부터 더 이상 권장되지 않음 → LangChain 1.0 버전 에서는 아예 삭제 예정**
  - 대신 **LCEL (LangChain Expression Language)** 방식인 **`prompt | llm`** 형태로 체인을 만드는 방법 **사용 권장**

---

*

- 기존 코드

```
from langchain_core.prompts import PromptTemplate
from langchain.chains import LLMChain

prompt1 = PromptTemplate.from_template(
    "다음 식당 리뷰를 한 문장으로 요약하세요.\n\n{review}"
)

chain1 = LLMChain(llm=llm, prompt=prompt1, output_key="summary")
```

---

*

- 권장 코드

```
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

# 1. 프롬프트 템플릿
prompt1 = PromptTemplate.from_template(
    "다음 식당 리뷰를 한 문장으로 요약하세요.\n\n{review}"
)

# 2. LCEL 방식 체인 구성
# prompt → llm → 문자열 파서
chain1 = prompt1 | llm | StrOutputParser()

# 3. 실행
result = chain1.invoke({"review": "음식이 맛있고 서비스가 친절했어요."})
print(result)
```

```
# 프롬프트 템플릿 설정(Prompot2)
prompt2 = PromptTemplate.from_template(
    "다음 식당 리뷰를 읽고 0점부터 10점 사이에서 긍정/부정 점수를 매기세요. 숫자만 대답하세요.\n\n{review}"
)

# 체인_2
chain2 = LLMChain(llm=llm, prompt=prompt2, output_key="sentiment_score")
```

```
# 프롬프트 템플릿 설정(Prompot3)
prompt3 = PromptTemplate.from_template(
    "다음 식당 리뷰 요약에 대해 공손한 답변을 작성하세요.\n리뷰 요약:{summary}"
)

# 체인_3
chain3 = LLMChain(llm=llm, prompt=prompt3, output_key="reply")
```

Chain 생성

```
all_chain = SequentialChain(
    chains=[chain1, chain2, chain3],
    input_variables=['review'],
    output_variables=['summary', 'sentiment_score', 'reply'],
)
```

reivew 식당 리뷰 입력

```
# 식당 리뷰 입력
review = """
이 식당은 맛도 좋고 분위기도 좋았습니다. 가격 대비 만족도가 높아요.
하지만, 서비스 속도가 너무 느려서 조금 실망스러웠습니다.
전반적으로는 다시 방문할 의사가 있습니다.
"""

# 체인 실행 및 결과 출력
try:
    result = all_chain.invoke(input={'review': review})
    print(f'summary 결과 \n {result["summary"]} \n')
    print(f'sentiment_score 결과 \n {result["sentiment_score"]} \n')
    print(f'reply 결과 \n {result["reply"]}')
except Exception as e:
    print(f"Error: {e}")
```

- 교재 속 답변 (gpt-3.5-turbo 모델 기준)

  summary 결과
  맛과 분위기는 좋지만 서비스가 느리고 실망스러웠습니다. 그래도 가격 대비 만족도가 높아서 다시 방문할 의사가 있습니다.

  sentiment_score 결과
  8

  reply 결과
  저희 식당을 방문해주셔서 감사합니다. 솔직한 리뷰를 통해 서비스 부분에 대한 문제점을 알게 되었고, 죄송하다는 말씀을 드립니다. 더 빠른 서비스를 제공할

LCEL 방식으로 바꾼 코드

- gemini-2.5-flash-lite 사용

```
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser
from dotenv import load_dotenv
import os

# .env 파일 로드
load_dotenv()
api_key = os.getenv("GOOGLE_API_KEY")

# 1. LLM 설정 (OpenAI Chat 모델)
gemini_lc = ChatGoogleGenerativeAI(
        model="gemini-2.5-flash-lite",
        temperature=0.7,
```

```
        max_output_tokens=4096,
    )

# 2. 출력 파서 (모델 응답을 문자열로 변환)
parser = StrOutputParser()

# 3. 프롬프트 템플릿 정의
prompt1 = PromptTemplate.from_template(
    "다음 식당 리뷰를 한 문장으로 요약하세요.\n\n{review}"
)
prompt2 = PromptTemplate.from_template(
    "다음 식당 리뷰를 읽고 0점부터 10점 사이에서 긍정/부정 점수를 매기세요. 숫자만 대답하세요.\n\n{review}"
)
prompt3 = PromptTemplate.from_template(
    "다음 식당 리뷰 요약에 대해 공손한 답변을 작성하세요.\n리뷰 요약:{summary}"
)

# 4. LCEL 방식 체인 구성
#    각 체인은 prompt → llm → parser 순서로 연결
chain1 = prompt1 | gemini_lc | parser
chain2 = prompt2 | gemini_lc | parser
chain3 = prompt3 | gemini_lc | parser

# 5. 전체 실행 로직
review = """
이 식당은 맛도 좋고 분위기도 좋았습니다. 가격 대비 만족도가 높아요.
하지만, 서비스 속도가 너무 느려서 조금 실망스러웠습니다.
전반적으로는 다시 방문할 의사가 있습니다.
"""

# 6. 순차 실행 (LCEL에서는 직접 순서를 제어)
summary = chain1.invoke({"review": review})
sentiment_score = chain2.invoke({"review": review})
reply = chain3.invoke({"summary": summary})

# 7. 결과 출력
print(f"summary 결과 \n{summary}\n")
print(f"sentiment_score 결과 \n{sentiment_score}\n")
print(f"reply 결과 \n{reply}")
```

- 셀 출력 (3.0s)

```
summary 결과
맛과 분위기, 가격 대비 만족도는 높았으나 느린 서비스 속도가 아쉬웠지만 전반적으로 재방문 의사가 있는 식당입니다.

sentiment_score 결과
7

reply 결과
## 식당 리뷰 요약에 대한 공손한 답변

**[고객님의 소중한 리뷰에 진심으로 감사드립니다.]**

맛과 분위기, 가격 대비 만족도에 대해 긍정적인 평가를 남겨주셔서 저희에게 큰 힘이 됩니다. 특히 저희 식당을 전반적으로 만족하시고 재방문 의사까지 밝혀주

다만, 서비스 속도에 대한 아쉬움도 솔직하게 말씀해주셔서 이 부분에 대해서도 깊이 인지하고 있습니다. 고객님께서 더욱 편안하고 만족스러운 식사를 경험하실

다시 한번 귀한 시간을 내어 리뷰를 작성해주셔서 감사드립니다. 다음 방문 시에는 개선된 모습으로 더욱 만족스러운 경험을 선사해 드릴 수 있도록 노력하겠습

**[000 드림 (혹은 식당 이름)]**
```

- **gpt-4o-mini vs gemini-2.5-flash-lite** 결과 비교

	gpt-3.5-turbo
summary 결과	맛과 분위기는 좋지만 서비스가 느리고 실망스러웠습니다. 그래도 가격 대비 만족도가 높아서 다시 방문할 의사가 있습니다.
sentiment_score 결과	8
reply 결과	저희 식당을 방문해주셔서 감사합니다. 솔직한 리뷰를 통해 서비스 부분에 대한 문제점을 알게 되었고, 죄송하다는 말씀을 드립니다. 더 빠른 서비스를 제공할 수 있도록 노력하겠습니다. 맛과 분위기에 대