**LCEL을 사용하여 간단한 LLM 애플리케이션 구축**

이 빠른 시작에서는 LangChain으로 간단한 LLM 애플리케이션을 빌드하는 방법을 보여드리겠습니다. 이 애플리케이션은 영어에서 다른 언어로 텍스트를 번역합니다. 이것은 비교적 간단한 LLM 애플리케이션입니다. 단일 LLM 호출과 약간의 프롬프트만 있습니다. 그래도 이것은 LangChain을 시작하기에 좋은 방법입니다. 프롬프트와 LLM 호출만으로 많은 기능을 빌드할 수 있습니다!

이 튜토리얼을 읽은 후에는 다음 내용에 대한 전반적인 개요를 알 수 있습니다.

* [언어 모델](https://python.langchain.com/docs/concepts/chat_models/) 사용
* [PromptTemplates](https://python.langchain.com/docs/concepts/prompt_templates/) 및 [OutputParsers](https://python.langchain.com/docs/concepts/output_parsers/) 사용
* [LangChain Expression Language(LCEL)를](https://python.langchain.com/docs/concepts/lcel/) 사용하여 구성 요소를 함께 연결합니다.
* [LangSmith를](https://docs.smith.langchain.com/) 사용하여 애플리케이션 디버깅 및 추적
* [LangServe를](https://python.langchain.com/docs/concepts/architecture/#langserve) 사용하여 애플리케이션 배포

시작해볼까요!

[**설정**](https://python.langchain.com/docs/tutorials/llm_chain/#setup)

**주피터**[**노트북**](https://python.langchain.com/docs/tutorials/llm_chain/#jupyter-notebook)

이 가이드(및 설명서의 다른 대부분의 가이드)는 [Jupyter 노트북을](https://jupyter.org/" \t "_blank) 사용하며 독자도 마찬가지라고 가정합니다. Jupyter 노트북은 LLM 시스템을 사용하는 방법을 배우기에 완벽합니다. 종종 일이 잘못될 수 있기 때문입니다(예상치 못한 출력, API 다운 등). 대화형 환경에서 가이드를 살펴보면 가이드를 더 잘 이해할 수 있는 좋은 방법입니다.

이 튜토리얼과 다른 튜토리얼은 아마도 Jupyter 노트북에서 가장 편리하게 실행할 수 있습니다. 설치 방법에 대한 지침은 [여기를 참조하세요.](https://jupyter.org/install)

[**설치**](https://python.langchain.com/docs/tutorials/llm_chain/#installation)

LangChain을 설치하려면 다음을 실행하세요.

* 씨
* 콘다

pip install langchain

자세한 내용은 [설치 가이드를](https://python.langchain.com/docs/how_to/installation/) 참조하세요 .

[**랭스미스**](https://python.langchain.com/docs/tutorials/llm_chain/#langsmith)

LangChain으로 빌드하는 많은 애플리케이션에는 LLM 호출을 여러 번 호출하는 여러 단계가 포함됩니다. 이러한 애플리케이션이 점점 더 복잡해짐에 따라 체인이나 에이전트 내부에서 정확히 무슨 일이 일어나고 있는지 검사할 수 있는 것이 중요해집니다. 이를 수행하는 가장 좋은 방법은 [LangSmith를](https://smith.langchain.com/" \t "_blank) 사용하는 것입니다 .

위 링크에서 가입한 후 추적 로깅을 시작하려면 환경 변수를 설정해야 합니다.

export LANGCHAIN\_TRACING\_V2="true"  
export LANGCHAIN\_API\_KEY="..."

또는 노트북에 있는 경우 다음을 사용하여 설정할 수 있습니다.

import getpass  
import os  
  
os.environ["LANGCHAIN\_TRACING\_V2"] = "true"  
os.environ["LANGCHAIN\_API\_KEY"] = getpass.getpass()

**언어**[**모델**](https://python.langchain.com/docs/tutorials/llm_chain/#using-language-models)

먼저 언어 모델을 단독으로 사용하는 방법을 알아보겠습니다. LangChain은 상호 교환하여 사용할 수 있는 다양한 언어 모델을 지원합니다. 아래에서 사용하고 싶은 모델을 선택하세요!

* 오픈AI
* 인류학적
* 하늘빛
* Google
* 코히어
* 엔비디아
* 불꽃놀이AI
* 그록
* 미스트랄AI
* 투게더AI
* 한국어: AWS

pip install -qU langchain-openai

import getpass  
import os  
  
os.environ["OPENAI\_API\_KEY"] = getpass.getpass()  
  
from langchain\_openai import ChatOpenAI  
  
model = ChatOpenAI(model="gpt-4")

먼저 모델을 직접 사용해 보겠습니다. ChatModels는 LangChain "Runnables"의 인스턴스로, 이는 s가 s와 상호 작용하기 위한 표준 인터페이스를 노출한다는 것을 의미합니다. 모델을 간단히 호출하려면 메서드에 메시지 목록을 전달할 수 있습니다 .invoke.

from langchain\_core.messages import HumanMessage, SystemMessage  
  
messages = [  
 SystemMessage(content="Translate the following from English into Italian"),  
 HumanMessage(content="hi!"),  
]  
  
model.invoke(messages)

**API 참조:**[HumanMessage](https://python.langchain.com/api_reference/core/messages/langchain_core.messages.human.HumanMessage.html) | [시스템 메시지](https://python.langchain.com/api_reference/core/messages/langchain_core.messages.system.SystemMessage.html)

AIMessage(content='ciao!', response\_metadata={'token\_usage': {'completion\_tokens': 3, 'prompt\_tokens': 20, 'total\_tokens': 23}, 'model\_name': 'gpt-4', 'system\_fingerprint': None, 'finish\_reason': 'stop', 'logprobs': None}, id='run-fc5d7c88-9615-48ab-a3c7-425232b562c5-0')

LangSmith를 활성화했다면 이 실행이 LangSmith에 기록되고 [LangSmith 추적을 볼 수 있습니다.](https://smith.langchain.com/public/88baa0b2-7c1a-4d09-ba30-a47985dde2ea/r" \t "_blank)

[**출력파서**](https://python.langchain.com/docs/tutorials/llm_chain/#outputparsers)

모델의 응답이 AIMessage.인 것을 주목하세요. 여기에는 응답에 대한 다른 메타데이터와 함께 문자열 응답이 포함됩니다. 종종 문자열 응답만 사용하고 싶을 수 있습니다. 간단한 출력 파서를 사용하여 이 응답만 파싱할 수 있습니다.

먼저 간단한 출력 파서를 가져옵니다.

from langchain\_core.output\_parsers import StrOutputParser  
  
parser = StrOutputParser()

**API 참조:**[StrOutputParser](https://python.langchain.com/api_reference/core/output_parsers/langchain_core.output_parsers.string.StrOutputParser.html)

그것을 사용하는 한 가지 방법은 그것을 그 자체로 사용하는 것입니다. 예를 들어, 언어 모델 호출의 결과를 저장한 다음 파서에 전달할 수 있습니다.

result = model.invoke(messages)

parser.invoke(result)

'Ciao!'

더 일반적으로, 우리는 이 출력 파서로 모델을 "체인"할 수 있습니다. 즉, 이 출력 파서는 이 체인에서 매번 호출됩니다. 이 체인은 언어 모델의 입력 유형(문자열 또는 메시지 목록)을 취하고 출력 파서의 출력 유형(문자열)을 반환합니다.

연산자를 사용하여 체인을 쉽게 만들 수 있습니다 |. |연산자는 LangChain에서 두 요소를 결합하는 데 사용됩니다.

chain = model | parser

chain.invoke(messages)

'Ciao!'

이제 LangSmith를 살펴보면 체인에 두 단계가 있음을 알 수 있습니다. 먼저 언어 모델이 호출되고, 그 결과가 출력 파서로 전달됩니다. [LangSmith 추적을 볼 수 있습니다.](https://smith.langchain.com/public/f1bdf656-2739-42f7-ac7f-0f1dd712322f/r" \t "_blank)

**프롬프트**[**템플릿**](https://python.langchain.com/docs/tutorials/llm_chain/#prompt-templates)

지금 우리는 메시지 목록을 언어 모델에 직접 전달하고 있습니다. 이 메시지 목록은 어디에서 오는 것일까요? 일반적으로 사용자 입력과 애플리케이션 로직의 조합으로 구성됩니다. 이 애플리케이션 로직은 일반적으로 원시 사용자 입력을 가져와 언어 모델에 전달할 준비가 된 메시지 목록으로 변환합니다. 일반적인 변환에는 시스템 메시지를 추가하거나 사용자 입력으로 템플릿을 포맷하는 것이 포함됩니다.

PromptTemplates는 이러한 변환을 지원하도록 설계된 LangChain의 개념입니다. 원시 사용자 입력을 받아 언어 모델로 전달할 준비가 된 데이터(프롬프트)를 반환합니다.

여기서 PromptTemplate을 만들어 보겠습니다. 두 개의 사용자 변수를 사용합니다.

* language: 텍스트를 번역할 언어
* text: 번역할 텍스트

from langchain\_core.prompts import ChatPromptTemplate

**API 참조:**[ChatPromptTemplate](https://python.langchain.com/api_reference/core/prompts/langchain_core.prompts.chat.ChatPromptTemplate.html)

먼저, 시스템 메시지로 포맷할 문자열을 만들어 보겠습니다.

system\_template = "Translate the following into {language}:"

다음으로 PromptTemplate을 만들 수 있습니다. 이것은 system\_template번역할 텍스트를 넣을 위치에 대한 더 간단한 템플릿과 조합이 될 것입니다.

prompt\_template = ChatPromptTemplate.from\_messages(  
 [("system", system\_template), ("user", "{text}")]  
)

이 프롬프트 템플릿의 입력은 사전입니다. 이 프롬프트 템플릿을 단독으로 가지고 놀면서 그 자체로 무엇을 하는지 볼 수 있습니다.

result = prompt\_template.invoke({"language": "italian", "text": "hi"})  
  
result

ChatPromptValue(messages=[SystemMessage(content='Translate the following into italian:'), HumanMessage(content='hi')])

두 개의 메시지로 구성된 를 반환하는 것을 볼 수 있습니다 ChatPromptValue. 메시지에 직접 액세스하려면 다음을 수행합니다.

result.to\_messages()

[SystemMessage(content='Translate the following into italian:'),  
 HumanMessage(content='hi')]

[**LCEL을**](https://python.langchain.com/docs/tutorials/llm_chain/#chaining-together-components-with-lcel)**사용하여 구성 요소 연결**

이제 파이프( ) 연산자를 사용하여 위의 모델과 출력 파서를 결합할 수 있습니다 |.

chain = prompt\_template | model | parser

chain.invoke({"language": "italian", "text": "hi"})

'ciao'

[이는 LangChain 모듈을 체인으로 연결하기 위해 LangChain Expression Language(LCEL)를](https://python.langchain.com/docs/concepts/lcel/) 사용하는 간단한 예입니다 . 이 접근 방식에는 최적화된 스트리밍 및 추적 지원을 포함하여 여러 가지 이점이 있습니다.

LangSmith 추적을 살펴보면 세 가지 구성 요소가 모두 [LangSmith 추적](https://smith.langchain.com/public/bc49bec0-6b13-4726-967f-dbd3448b786d/r" \t "_blank) 에 표시되는 것을 볼 수 있습니다 .

[**LangServe**](https://python.langchain.com/docs/tutorials/llm_chain/#serving-with-langserve)**로 서비스 제공**

이제 애플리케이션을 빌드했으니, 이를 제공해야 합니다. 바로 여기서 LangServe가 등장합니다. LangServe는 개발자가 LangChain 체인을 REST API로 배포하도록 돕습니다. LangChain을 사용하기 위해 LangServe를 사용할 필요는 없지만, 이 가이드에서는 LangServe로 앱을 배포하는 방법을 보여드리겠습니다.

이 가이드의 첫 번째 부분은 Jupyter Notebook 또는 스크립트에서 실행되도록 의도되었지만, 이제 그로부터 벗어나겠습니다. Python 파일을 만든 다음 명령줄에서 그것과 상호 작용할 것입니다.

다음을 사용하여 설치:

pip install "langserve[all]"

[**서버**](https://python.langchain.com/docs/tutorials/llm_chain/#server)

애플리케이션을 위한 서버를 만들기 위해 파일을 만들 것입니다 serve.py. 여기에는 애플리케이션을 제공하기 위한 로직이 포함됩니다. 세 가지로 구성됩니다.

1. 우리가 방금 위에서 구축한 체인의 정의
2. FastAPI 앱
3. 체인을 서비스할 경로의 정의는 다음을 통해 수행됩니다.langserve.add\_routes

#!/usr/bin/env python  
from fastapi import FastAPI  
from langchain\_core.prompts import ChatPromptTemplate  
from langchain\_core.output\_parsers import StrOutputParser  
from langchain\_openai import ChatOpenAI  
from langserve import add\_routes  
  
# 1. Create prompt template  
system\_template = "Translate the following into {language}:"  
prompt\_template = ChatPromptTemplate.from\_messages([  
 ('system', system\_template),  
 ('user', '{text}')  
])  
  
# 2. Create model  
model = ChatOpenAI()  
  
# 3. Create parser  
parser = StrOutputParser()  
  
# 4. Create chain  
chain = prompt\_template | model | parser  
  
# 5. App definition  
app = FastAPI(  
 title="LangChain Server",  
 version="1.0",  
 description="A simple API server using LangChain's Runnable interfaces",  
)  
  
# 6. Adding chain route  
add\_routes(  
 app,  
 chain,  
 path="/chain",  
)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import uvicorn  
  
 uvicorn.run(app, host="localhost", port=8000)

**API 참조:**[ChatPromptTemplate](https://python.langchain.com/api_reference/core/prompts/langchain_core.prompts.chat.ChatPromptTemplate.html) | [StrOutputParser](https://python.langchain.com/api_reference/core/output_parsers/langchain_core.output_parsers.string.StrOutputParser.html) | [ChatOpenAI](https://python.langchain.com/api_reference/openai/chat_models/langchain_openai.chat_models.base.ChatOpenAI.html)

그리고 그게 전부입니다! 이 파일을 실행하면:

python serve.py

우리 체인이 [http://localhost:8000](http://localhost:8000/) 에서 제공되는 것을 볼 수 있어야 합니다 .

[**놀이터**](https://python.langchain.com/docs/tutorials/llm_chain/#playground)

모든 LangServe 서비스에는 스트리밍 출력과 중간 단계에 대한 가시성을 갖춘 애플리케이션을 구성하고 호출하기 위한 간단한 [내장 UI가](https://github.com/langchain-ai/langserve/blob/main/README.md#playground) 제공됩니다 . <http://localhost:8000/chain/playground/> 로 이동하여 시도해 보세요! 이전과 동일한 입력을 전달하면 {"language": "italian", "text": "hi"}이전과 동일하게 응답해야 합니다.

[**클라이언트**](https://python.langchain.com/docs/tutorials/llm_chain/#client)

이제 서비스와 프로그래밍적으로 상호작용할 클라이언트를 설정해 보겠습니다. [langserve.RemoteRunnable](https://python.langchain.com/docs/langserve/" \l "client) 을 사용하면 쉽게 할 수 있습니다 . 이를 사용하면 클라이언트 측에서 실행되는 것처럼 제공된 체인과 상호작용할 수 있습니다.

from langserve import RemoteRunnable  
  
remote\_chain = RemoteRunnable("http://localhost:8000/chain/")  
remote\_chain.invoke({"language": "italian", "text": "hi"})

LangServe의 다양한 다른 기능에 대해 자세히 알아보려면 [여기를 클릭하세요](https://python.langchain.com/docs/langserve/) .

[**결론**](https://python.langchain.com/docs/tutorials/llm_chain/#conclusion)

다 됐어요! 이 튜토리얼에서는 첫 번째 간단한 LLM 애플리케이션을 만드는 방법을 배웠습니다. 언어 모델을 사용하는 방법, 출력을 구문 분석하는 방법, 프롬프트 템플릿을 만드는 방법, LCEL로 체인하는 방법, LangSmith로 만든 체인에 뛰어난 관찰성을 도입하는 방법, LangServe로 배포하는 방법을 배웠습니다.

이것은 당신이 유능한 AI 엔지니어가 되기 위해 배우고 싶어하는 것의 피상적인 부분일 뿐입니다. 다행히도 - 우리는 다른 많은 리소스를 가지고 있습니다!

LangChain의 핵심 개념에 대해 더 자세히 알아보려면 자세한 [개념 가이드를](https://python.langchain.com/docs/concepts/) 참조하세요 .

이러한 개념에 대해 더 구체적인 질문이 있으면 다음 방법 가이드 섹션을 확인하세요.

* [랭체인 표현 언어(LCEL)](https://python.langchain.com/docs/how_to/#langchain-expression-language-lcel)
* [프롬프트 템플릿](https://python.langchain.com/docs/how_to/#prompt-templates)
* [채팅 모델](https://python.langchain.com/docs/how_to/#chat-models)
* [출력 파서](https://python.langchain.com/docs/how_to/#output-parsers)
* [랭서브](https://python.langchain.com/docs/langserve/)

그리고 LangSmith 문서:

* [랭스미스](https://docs.smith.langchain.com/)