

섹션 2: LangChain LCEL 사용

1. LCEL (LangChain Expression Language)

- LangChain 컴포넌트를 조합하여 복잡한 작업 흐름을 구성할 수 있게 해주는 도구

1.1 Prompt + LLM

- LCEL을 사용한 기본적인 프롬프트와 LLM 연결

LCEL: 프롬프트와 LLM 연결

<> 코드 예시

pipe 연산자를 활용한 체인 구성

```
from langchain_core.prompts import ChatPromptTemplate

prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful assistant."),
    ("user", "{input}"),
])

chain = prompt | llm
response = chain.invoke("3 더하기 2는 얼마인가요?")
```

□ 설명 및 다이어그램

LCEL을 사용하면 pipe 연산자(|)를 통해 프롬프트와 LLM을 쉽게 연결할 수 있습니다.

프롬프트

 →

LLM

 →

응답

장점:

- 간결하고 직관적인 코드
- 모듈화된 구조로 유지보수 용이
- 다양한 컴포넌트를 쉽게 조합 가능

- 동적 프롬프트 템플릿 생성 및 활용



1.2 Prompt + LLM + Output Parser

- 구조화된 출력을 위한 Output Parser 사용

LCEL: 출력 파싱

<> 파서 종류 및 코드 예시

- StrOutputParser: 문자열 파싱**

```
from langchain_core.output_parsers import StrOutputParser
chain = prompt | llm | StrOutputParser()
```
- JsonOutputParser: JSON 파싱**
JSON 형식의 출력을 파싱하여 Python 딕셔너리로 변환
- PydanticOutputParser: Pydantic 객체 파싱**
출력을 Pydantic 모델에 맞춰 파싱하고 검증

☞ 사용 사례 및 장점

주요 사용 사례:

- 구조화된 데이터 추출
- 형식화된 응답 생성
- LLM 출력의 일관성 보장
- 복잡한 객체 생성 및 검증

{ '원소': '산소', '원자번호': 8 }

장점:

- 출력 형식의 표준화
- 에러 처리 및 검증 간소화
- 다양한 데이터 형식 지원
- 파이프라인 구성의 유연성 증가

2. Chat Completion Methods

- invoke: 완성된 출력을 전달
- stream: 입력에 대한 응답을 실시간 스트림을 생성하여 전달
- batch: 입력 리스트에 대한 응답을 배치 단위로 생성

3. Runnable

- Runnable은 LCEL의 핵심 개념으로, 다양한 컴포넌트를 연결하고 실행하는 인터페이스
- RunnableParallel

Runnable: 병렬 실행

<> 코드 예시 및 설명

여러 작업을 동시에 실행하고 결과를 딕셔너리로 반환

```
from langchain_core.runnables import RunnableParallel

parallel_chain = RunnableParallel({
    "topic": question_chain,
    "language": language_chain,
    "question": itemgetter("question")
})
```

설명: RunnableParallel을 사용하여 여러 체인을 동시에 실행하고, 각 체인의 결과를 지정된 키와 함께 딕셔너리 형태로 반환합니다.

🔗 병렬 실행 구조 다이어그램

```
graph LR
    Input[입력] --> RunnableParallel[RunnableParallel]
    topic_chain[topic_chain] --> ResultDict[결과 딕셔너리]
    language_chain[language_chain] --> ResultDict
    itemgetter[itemgetter] --> ResultDict
    ResultDict --> Output[출력]
```

- RunnablePassthrough
- RunnableLambda

데이터 전달 및 변환

<> RunnablePassthrough & RunnableLambda

🔗 RunnablePassthrough

입력을 그대로 전달

⚙️ RunnableLambda

커스텀 함수로 입력 변환

```
chain = RunnableParallel({ "passed": RunnablePassthrough(),
    "modified": RunnableLambda(lambda x: x.upper()) })
```

🔗 동작 방식 및 사용 사례

```
graph LR
    InputData[입력 데이터] --> RunnableParallel[RunnableParallel]
    RunnablePassthrough[RunnablePassthrough] --> ResultDict[결과 딕셔너리]
    RunnableLambda[RunnableLambda] --> ResultDict
```

주요 사용 사례:

- 데이터 전처리
- 후처리
- 조건부 로직 적용

4. RAG 파이프라인 구현 실습

- RAG 프롬프트 템플릿 활용
- 벡터저장소 검색기를 활용한 Retrival Chain 구성
- LCEL 과 Runnable 활용한 RAG Chain 완성
- Gradio 챗봇 구현 (stream 출력 모드)

QA Bot

Chatbot

잘 모르겠습니다.

테슬라 역사에 대해서 설명해

테슬라는 2003년 마틴 에버하드와 마크 타페닝에 의해 설립되었습니다. 2004년에는 페이팔과 Zip2의 공동 창업자인 일론 머스크가 참여하면서 큰 전환점을 맞았습니다. 머스크는 최대 주주이자 회장으로서 회사를 현재의 성공으로 이끌었습니다.

Retry

Undo

Clear

Type a message...

Submit

Use via API  · gradio로 제작되었습니다 