LangChain 2주차 정리

category	LAB
sub category	GDGoC
상태	분류대기
마감일	@2025년 10월 31일
완료	☐

LangChain 프롬프트 시스템 구조 및 역할 정리

1. 프롬프트(Prompt)의 개념과 역할

LangChain에서 프롬프트는 **LLM이 수행할 작업을 구체적으로 지시하는 입력 구조**로 사용됨.

일반 문자열 형태의 단순 프롬프트 외에도, 구조화된 템플릿(`PromptTemplate`, `ChatPromptTemplate`)을 통해 재사용성과 동적 구성 가능.

LangChain의 프롬프트 시스템은 다음 두 가지 형태로 나뉨:

- **PromptTemplate**: 단일 텍스트 입력을 기반으로 변수 치환 수행.
- **ChatPromptTemplate**: 시스템, 사용자, AI 등의 역할(role)을 구분하여 메시지 단위로 구성.

2. PromptTemplate

2.1 기본 구조

`PromptTemplate`은 문자열 템플릿 내부의 `{변수}`에 값을 대입하여 동적으로 프롬프트를 생성함.

```
from langchain_core.prompts import PromptTemplate
template = "{country}의 수도는 어디인가요?"
prompt = PromptTemplate.from_template(template)
prompt.format(country="대한민국")
```

→ 결과: "대한민국의 수도는 어디인가요?"

`input_variables`를 명시하면 유효성 검증 수행 가능.

치환되지 않은 변수가 존재할 경우 예외 발생.

2.2 부분 변수(partial_variables)

`partial_variables`를 사용하면 일부 변수를 미리 고정하거나, 실행 시점에 자동으로 갱신되도록 설정할 수 있음.

주로 날짜, 시간 등 반복적으로 사용되는 값에 유용함.

```
from datetime import datetime
def get_today():
    return datetime.now().strftime("%B %d")

prompt = PromptTemplate(
    template="오늘의 날짜는 {today} 입니다. {n}명의 생일자를 알려주세요.",
    input_variables=["n"],
    partial_variables={"today": get_today}
)
```

→ 매 실행 시점마다 `{today}`가 현재 날짜로 자동 대체됨.

2.3 Prompt와 LLM의 연결 (Chain)

프롬프트와 LLM 객체를 파이프라인 형태(`|`)로 연결하여 Chain 구성 가능.

```
from langchain_openai import ChatOpenAI
llm = ChatOpenAI()
chain = prompt | llm
chain.invoke({"country": "대한민국"})
```

→ 프롬프트 포맷 → 모델 호출 → 결과 반환 과정을 단일 구조로 통합.

3. ChatPromptTemplate

`ChatPromptTemplate`은 역할(role) 기반 메시지 구성을 지원함.

시스템(System), 사용자(Human), AI 등의 역할별 지시문을 조합하여 대화형 맥락 구성 가능.

```
from langchain_core.prompts import ChatPromptTemplate
chat_template = ChatPromptTemplate.from_messages([
    ("system", "당신은 친절한 AI 어시스턴트입니다."),
    ("human", "{question}")
])
```

`format_messages()`를 통해 실제 메시지 목록 생성 후 LLM에 전달함.

복잡한 대화 이력 삽입 시 `MessagesPlaceholder`를 사용하면 `conversation` 변수로 이전 메시지 목록을 통째로 주입 가능.

```
from langchain_core.prompts import MessagesPlaceholder
chat_prompt = ChatPromptTemplate.from_messages([
    ("system", "요약 전문 AI입니다."),
    MessagesPlaceholder(variable_name="conversation"),
    ("human", "지금까지의 대화를 {word_count} 단어로 요약하세요.")
])
```

이 구조는 다중 턴 대화, 요약, 메타정보 삽입 등 복합형 프롬프트 설계에 적합함.

4. FewShotPromptTemplate

4.1 개념

Few-shot 프롬프트는 **예시(Example)를 기반으로 모델의 응답 패턴을 유도하는 구조**임.

`FewShotPromptTemplate`은 사전 정의된 예시들을 자동으로 삽입하여 프롬프트를 생성함.

```
from langchain_core.prompts.few_shot import FewShotPromptTemplate
prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=PromptTemplate.from_template("Q: {question}\nA: {an
swer}"),
    suffix="Q: {question}\nA:",
    input_variables=["question"]
)
```

`format()` 호출 시, 예시와 함께 새 질문이 포함된 프롬프트 생성.

예시 기반의 학습 효과로 일관된 출력 형식 확보.

4.2 Example Selector

예시가 많을 경우, 입력과 의미적으로 가장 유사한 예시를 자동으로 선택하기 위해 `ExampleSelector`를 사용함.

`SemanticSimilarityExampleSelector`는 임베딩(embedding)을 이용해 입력과 예시 간 유사도를 계산.

```
from langchain_core.example_selectors import SemanticSimilarityExample
Selector
from langchain_openai import OpenAIEmbeddings
from langchain_chroma import Chroma

example_selector = SemanticSimilarityExampleSelector.from_examples(
    examples,
    OpenAIEmbeddings(),
    Chroma,
    k=1
)
```

입력에 따라 적합한 예시를 선택하여 FewShotPromptTemplate 내부에 자동 삽입함.

이는 Contextual few-shot learning을 구현하는 핵심 구조임.

4.3 FewShotChatMessagePromptTemplate

`FewShotChatMessagePromptTemplate`은 Chat 형식의 Few-shot 구조를 지원함.

예시를 채팅 메시지(`human`, `ai`) 쌍으로 저장하고, 새로운 입력에 대해 가장 유사한 예시를 삽입함.

```
from langchain_core.prompts import FewShotChatMessagePromptTemplat
e
few_shot_prompt = FewShotChatMessagePromptTemplate(
    example_selector=example_selector,
    example_prompt=example_prompt
)
```

이를 통해 대화형 예시 기반 응답 생성이 가능하며, 문장 교정·요약·회의록 작성 등 다양한 태스크에 활용됨.

4.4 Custom Example Selector

기본 `SemanticSimilarityExampleSelector`는 `instruction`과 `input`을 동시에 고려하지 못할 수 있음.

이를 보완하기 위해 사용자는 커스텀 클래스를 정의해 instruction 중심의 유사도 계산을 수행함.

```
from langchain_teddynote.prompts import CustomExampleSelector
custom_selector = CustomExampleSelector(examples, OpenAIEmbeddings
())
```

→ 텍스트 생성 태스크의 정확도를 높이는 중요한 보조 컴포넌트로 활용됨.

5. LangChain Hub

5.1 개념 및 역할

LangChain Hub는 **공유 가능한 프롬프트 저장소**로, 사용자 정의 프롬프트를 버전 관리 및 배포할 수 있는 플랫폼임.

GitHub 유사한 구조를 가지며, 각 프롬프트는 `{owner}/{repo}` 형식으로 관리됨.

- **pull()**: Hub에서 프롬프트 다운로드
- **push()**: Hub에 프롬프트 업로드

```
from langchain import hub
prompt = hub.pull("rlm/rag-prompt")
hub.push("teddynote/simple-summary-korean", prompt)
```

`push()` 수행 시 **LangSmith 인증키**(`LANGCHAIN_API_KEY`) 필요하며, 해당 리포에 대한 소유권이 있어야 함.

5.2 프롬프트 버전 관리

Hub는 commit hash 기반 버전 관리 지원.

특정 버전의 프롬프트를 명시적으로 호출 가능.

```
prompt = hub.pull("rlm/rag-prompt:50442af1")
```

→ 특정 커밋 해시를 사용해 동일 버전 재현 가능.

5.3 주요 프롬프트 유형 예시

(1) Summary Prompt

문서 요약 목적의 프롬프트 정의.

```
prompt_template = """Please summarize the sentence according to the foll
owing REQUEST.
REQUEST:
1. Summarize the main points in bullet points.
2. Each summarized sentence must start with an emoji.
CONTEXT:
```

```
{context}
SUMMARY:"""
```

`hub.push("teddynote/summary-stuff-documents", prompt)` 형태로 등록.

(2) Map & Reduce Prompts

Map: 개별 문서 요약

Reduce: 전체 요약 통합

RAG 파이프라인 및 Chain of Density 방식에서 자주 사용됨.

(3) Chain of Density Prompt

점차 정보 밀도를 높여가는 반복 요약 프로세스를 정의한 구조.

JSON 형태로 다단계 요약 결과 반환.

(4) RAG Prompt (Korean)

문맥(context) 기반 질의응답용 프롬프트.

주어진 문서 내에서 답을 찾지 못할 경우 명시적으로 “정보 없음”을 반환하도록 설계됨.

종합 요약

LangChain의 프롬프트 체계는 **모듈화·재사용·버전관리·자동화**를 목표로 함.

기본 PromptTemplate을 통해 유연한 프롬프트 구성 가능하며,

FewShot 및 ExampleSelector를 통해 문맥 기반 학습을 구현함.

Hub와 LangSmith를 통해 프롬프트를 공유·추적·배포할 수 있으며,

이로써 LLM 애플리케이션의 프롬프트 엔지니어링 과정이

재현 가능하고 협업 친화적인 워크플로우로 발전함.