CH02 - 프롬프트(Prompt)

프롬프트

: 언어 모델이 사용할 질문이나 명령을 생성하는 과정

프롬프트의 필요성

- **문맥(Context) 설정**

  → 프롬프트는 언어 모델이 특정 상황이나 배경에서 작동하도록 문맥을 설정합니다. 이를 통해 모델이 제공된 정보를 바탕으로 보다 **정확하고 관련성 높은 답변**을 생성할 수 있습니다.

- **정보 통합**

  → 여러 출처에서 검색된 다양한 관점의 정보를 **통합하고 정리**하여, 모델이 이를 효율적으로 활용할 수 있는 형태로 제공합니다.

- **응답 품질 향상**

  → 질문에 대한 답변의 품질은 프롬프트 구성에 크게 좌우됩니다. **명확하고 구조화된 프롬프트**는 모델이 더 정확하고 유용한 결과를 생성하도록 돕습니다.

I. RAG 프롬프트 구조 한눈에 보기

1. RAG란?

RAG(**Retrieval-Augmented Generation**)는

AI가 학습된 지식뿐 아니라 **검색된 외부 문서(Context)**를 참고해 답변을 생성하는 방식이다.

→ 즉, 단순히 “기억으로 답변”하는 게 아니라 “자료 기반으로 답변”하도록 만드는 구조.

---

2. RAG 프롬프트 구성 요소

1. **지시사항 (Instruction)**

   → AI에게 역할을 명확히 알려주는 부분 (예: “너는 질문-답변을 수행하는 AI야.”)

2. **질문 (Question)**

→ 사용자가 입력한 질문

3. **문맥 (Context)**

   → 검색 시스템을 통해 가져온 관련 문서나 정보

3. 프롬프트 예시

```
당신은 질문-답변(Task)을 수행하는 AI 어시스턴트입니다.
검색된 문맥(Context)을 사용하여 질문(Question)에 답하세요.
만약 문맥으로부터 답을 찾을 수 없다면 '모른다'고 말하세요.

#Question:
<여기에 사용자가 입력한 질문이 들어감>

#Context:
<여기에 검색된 문서 내용이 들어감>
```

II. LangChain 프롬프트(Prompt) 정리

1. PromptTemplate

: LLM에게 보낼 **문장 틀(Template)**을 정의하는 객체.

변수를 사용해 동적으로 프롬프트를 구성할 수 있다.

예시

```
from langchain_core.prompts import PromptTemplate

template = "{country}의 수도는 어디인가요?"
prompt = PromptTemplate.from_template(template)

print(prompt.format(country="대한민국"))
# 출력: 대한민국의 수도는 어디인가요?
```

핵심 포인트

- `{}` 안의 변수 이름은 나중에 `.format()`으로 채운다.
- 복잡한 문장을 재활용하거나 여러 변수값을 자동으로 넣을 수 있다.

2. input_variables & partial_variables

2-1. input_variables

→ 템플릿 안에서 반드시 입력받아야 하는 변수들.

예: `{country}`, `{topic}` 등

2-2. partial_variables

→ 미리 정의된 고정값이나 함수 결과를 자동으로 넣을 때 사용.

예시

```
from datetime import date
def get_today():
    return date.today().strftime("%Y-%m-%d")

template = "오늘 날짜는 {date}이며, 질문은 {question}입니다."
prompt = PromptTemplate(
    template=template,
    input_variables=["question"],
    partial_variables={"date": get_today()}
)
```

핵심 포인트

- `input_variables`는 사용자 입력
- `partial_variables`는 시스템이 미리 채워주는 값

---

3. 파일에서 템플릿 불러오기

YAML 파일 형태로 템플릿을 분리 관리할 수 있다.

예시

```
from langchain.prompts import load_prompt
prompt = load_prompt("prompts/fruit_color.yaml")
```

이 방법을 쓰면 코드와 프롬프트를 분리해 **유지보수 및 협업에 유리**하다.

---

4. ChatPromptTemplate

단일 문장이 아닌 **대화 메시지 목록 형태의 프롬프트**를 만들 때 사용.

(ChatGPT처럼 “시스템 → 유저 → AI” 대화 구조 표현 가능)

예시

```
from langchain_core.prompts import ChatPromptTemplate

chat_template = ChatPromptTemplate.from_messages([
    ("system", "당신은 친절한 AI 어시스턴트입니다. 이름은 {name}입니다."),
    ("human", "반가워요!"),
    ("ai", "안녕하세요! 무엇을 도와드릴까요?"),
    ("human", "{user_input}")
])
```

핵심 포인트

- `"system"`, `"human"`, `"ai"` 메시지 타입을 구분
- 나중에 `.format(name="도이", user_input="오늘 날씨 알려줘")` 식으로 변수 주입

---

5. MessagesPlaceholder

`ChatPromptTemplate` 안에 이미 존재하는 **대화 기록(이전 메시지들)**을 통째로 넣을 때 사용.

예시

```
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder

chat_template = ChatPromptTemplate.from_messages([
    ("system", "당신은 AI 상담사입니다."),
    MessagesPlaceholder(variable_name="chat_history"),
    ("human", "{user_input}")
])
```

여기서 `chat_history`는 다음과 같은 형식의 리스트로 들어간다.

```
chat_history = [
    ("human", "안녕하세요"),
```

```
    ("ai", "안녕하세요! 무엇을 도와드릴까요?")
]
```

정리 요약

구분	기능	주요 목적
`PromptTemplate`	단일 문장 템플릿	간단한 질문/응답 구조
`input_variables`	사용자 입력 변수	템플릿 값 채우기
`partial_variables`	고정값/자동 함수결과	재사용성 향상
`ChatPromptTemplate`	대화형 프롬프트 구성	system/human/ai 구분
`MessagesPlaceholder`	이전 대화기록 삽입	대화 맥락 유지

III. Few-Shot 프롬프트(FewShotPromptTemplate) 정리

1. 개념

Few-Shot Prompt란, **모델에게 몇 가지 예시(Examples)를 함께 제공하여**

그 패턴을 학습(참조)하게 하는 방식이다.

> 👉 “이런 질문에는 이렇게 답하라”는 식의 예시를 주고,
>
> 모델이 그 구조를 모방하게 하는 **유도형 프롬프트**다.

이 방식은 모델이 문맥 없이 대답하는 **Zero-Shot Prompt**보다 일관되고 정확한 응답을 유도할 수 있다.

---

2. 기본 사용 예시

예시 코드

```
from langchain_core.prompts.few_shot import FewShotPromptTemplate
from langchain_core.prompts import PromptTemplate

examples = [
    {
        "question": "스티브 잡스와 아인슈타인 중 누가 더 오래 살았나요?",
        "answer": """이 질문에 추가 질문이 필요한가요: 예.
추가 질문: 스티브 잡스는 몇 살에 사망했나요?
```

```
중간 답변: 스티브 잡스는 56세에 사망했습니다.
추가 질문: 아인슈타인은 몇 살에 사망했나요?
중간 답변: 아인슈타인은 76세에 사망했습니다.
최종 답변은: 아인슈타인""",
    },
    {
        "question": "한국의 수도는 어디인가요?",
        "answer": "서울입니다."
    },
]

example_prompt = PromptTemplate.from_template(
    "Question:\n{question}\nAnswer:\n{answer}"
)

prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    suffix="Question:\n{question}\nAnswer:",
    input_variables=["question"],
)

question = "Google이 창립된 연도에 Bill Gates의 나이는 몇 살인가요?"
print(prompt.format(question=question))
```

핵심 포인트

- `examples` : 예시 데이터 리스트
- `example_prompt` : 예시 하나의 구조 정의
- `suffix` : 마지막에 새 질문이 들어갈 부분
- `input_variables` : 새 입력 변수 이름

---

3. Example Selector (예시 선택기)

Few-Shot 예시가 많아지면, 모든 예시를 넣을 경우 **프롬프트가 너무 길어져 토큰 낭비**가 발생한다.

이를 해결하기 위해 **입력 질문과 가장 유사한 예시만 선택**하는 기능이 `ExampleSelector` 클래스들이다.

* SemanticSimilarityExampleSelector

유사도 기반 자동 예시 선택기

예시 코드

```
from langchain_core.prompts.example_selector import SemanticSimilarityE
xampleSelector
from langchain_community.vectorstores import FAISS
from langchain_openai import OpenAIEmbeddings
from langchain_core.prompts import PromptTemplate, FewShotPromptTem
plate

examples = [
    {"input": "고양이는 어떤 동물인가요?", "output": "고양이는 포유류입니다."},
    {"input": "고양이는 무엇을 먹나요?", "output": "주로 생선과 고기를 먹습니다."},
    {"input": "고양이는 어디에서 삽니까?", "output": "보통 집에서 삽니다."},
]

example_prompt = PromptTemplate(
    input_variables=["input", "output"],
    template="질문: {input}\n답변: {output}"
)

example_selector = SemanticSimilarityExampleSelector.from_examples(
    examples,
    OpenAIEmbeddings(),
    FAISS,
    k=2  # 가장 유사한 2개만 선택
)

few_shot_prompt = FewShotPromptTemplate(
    example_selector=example_selector,
    example_prompt=example_prompt,
    suffix="질문: {input}\n답변:",
    input_variables=["input"]
)
```

```
print(few_shot_prompt.format(input="고양이는 어디서 주로 사나요?"))
```

핵심 포인트

- 예시를 **벡터로 임베딩(embedding)** → 입력과 비교 → 가장 유사한 k개 선택
- `FAISS`나 `Chroma` 같은 벡터스토어 사용 가능
- 효율적이며, 질문에 맞는 예시만 포함되어 응답 품질 향상

---

* MaxMarginalRelevanceExampleSelector (MMR)

유사도 + 다양성 균형 조절

예시 코드

```
from langchain_core.prompts.example_selector import MaxMarginalReleva
nceExampleSelector

mmr_selector = MaxMarginalRelevanceExampleSelector.from_examples(
    examples,
    OpenAIEmbeddings(),
    FAISS,
    k=2,
    fetch_k=4  # 후보 중 4개 중에서 다양성 있는 2개 선택
)
```

특징

- 비슷한 예시만 몰리지 않도록 조정
- **"유사도는 유지하되, 다양성은 확보"**
- 검색된 예시가 중복된 주제를 다루지 않게 함

---

4. FewShotChatMessagePromptTemplate

개념

- 기존 FewShotPromptTemplate은 단일 텍스트 예시 기반
- FewShotChatMessagePromptTemplate은 **대화형(Chat) 형태의 예시 기반 템플릿**

즉, “system / human / ai” 메시지 구조로 구성된 예시를 Few-Shot 형태로 제공할 수 있다.

예시 코드

```
from langchain_core.prompts import (
    ChatPromptTemplate,
    FewShotChatMessagePromptTemplate,
    HumanMessagePromptTemplate,
    AIMessagePromptTemplate
)

examples = [
    {"question": "고양이와 개의 차이점은?", "answer": "고양이는 독립적이고, 개는 사회적입니다."},
    {"question": "사과와 바나나의 차이점은?", "answer": "사과는 단단하고, 바나나는 부드럽습니다."}
]

example_prompt = ChatPromptTemplate.from_messages([
    HumanMessagePromptTemplate.from_template("{question}"),
    AIMessagePromptTemplate.from_template("{answer}")
])

few_shot_chat_prompt = FewShotChatMessagePromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    prefix="다음은 사람과 AI의 예시 대화입니다:",
    suffix="이제 아래 질문에 답해주세요.\n{input}",
    input_variables=["input"]
)

print(few_shot_chat_prompt.format(input="커피와 차의 차이점은?"))
```

핵심 포인트

- system / human / ai 메시지 형식을 지원
- ChatGPT처럼 자연스러운 대화 흐름 학습 가능
- 실제 챗봇 설계 시 매우 유용

5. 유사도 검색 문제와 해결 방안

문제점

- 임베딩 품질이 낮거나 데이터가 다양할 경우,

  입력 질문과 관련 없는 예시가 선택될 수 있음.

  → 예: “고양이 울음소리” 질문인데 “고양이 먹이” 예시 선택

해결 방법

1. **임베딩 품질 향상** → 더 성능 좋은 모델 사용 (`text-embedding-3-large` 등)
2. **질문 전처리** → 핵심 키워드 추출 후 임베딩
3. **MMR Selector 사용** → 유사도와 다양성 균형 유지

정리 요약

구분	설명	주요 기능
`FewShotPromptTemplate`	예시 여러 개를 포함한 텍스트 기반 프롬프트	기본적인 few-shot 구조
`SemanticSimilarityExampleSelector`	유사도 기반 예시 자동 선택	입력과 가장 비슷한 예시 선택
`MaxMarginalRelevanceExampleSelector`	유사도 + 다양성 기반 선택	중복된 예시 제거
`FewShotChatMessagePromptTemplate`	대화형 예시 기반 프롬프트	system / human / ai 구조 지원
유사도 검색 문제 해결	임베딩 개선, 전처리, MMR 활용	비관련 예시 방지, 품질 향상

IV. LangChain Hub 정리

1. 개요

- Hub는 프롬프트 템플릿을 공유하고 재사용할 수 있는 리포지토리다.
- 즉, 미리 만들어진 프롬프트를 `pull` 해서 사용하거나, 내가 만든 프롬프트를 `push` 해서 공유할 수 있다.
- 버전 관리(commit hash)도 가능해서 특정 버전의 프롬프트를 지정해서 불러올 수 있다.

2. 허브로부터 프롬프트 가져오기 (`pull`)

예시 코드

```
from langchain import hub

# 가장 최신 버전의 프롬프트를 가져옵니다.
prompt = hub.pull("rlm/rag-prompt")

print(prompt)

# 특정 버전(커밋 해시)을 지정해서 가져오기
prompt = hub.pull("rlm/rag-prompt:50442af1")
```

핵심 포인트

- `"rlm/rag-prompt"` 은 허브에 올라간 리포지토리 ID 예시이다.
- `:50442af1` 처럼 커밋 해시를 붙이면 특정 버전을 명시 가능
- 반환된 프롬프트에는 `input_variables`, `metadata`, `messages` 등이 포함돼 있다.

3.허브에 프롬프트 등록하기 (`push`)

예시 코드

```
from langchain.prompts import ChatPromptTemplate
from langchain import hub

prompt = ChatPromptTemplate.from_template(
    "주어진 내용을 바탕으로 다음 문장을 요약하세요. 답변은 반드시 한글로 작성하세
요\n\nCONTEXT: {context}\n\nSUMMARY:"
)
# 허브에 업로드
hub.push("teddynote/simple-summary-korean", prompt)
```

핵심 포인트

- `"teddynote/simple-summary-korean"` 은 새 리포지토리 ID 예시이다.
- 업로드 성공 시 `https://smith.langchain.com/hub/teddynote/simple-summary-korean/0e296563` 처럼 출력된다.

- 이로써 다른 사용자도 해당 프롬프트를 `pull` 해서 사용할 수 있게 된다.

정리 요약

요소	설명
`hub.pull(...)`	허브에서 프롬프트를 다운로드
`hub.push(...)`	허브에 사용자 프롬프트 업로드
리포지토리 ID	허브에 등록된 프롬프트 식별자
커밋 해시 지정	특정 버전의 프롬프트를 불러오기 가능
`metadata`, `messages` 항목	프롬프트의 구조 및 버전 정보 포함

V. LangChain 개인화된 프롬프트 모음 (Hub에 업로드)

1. Stuff Documents Prompt

개념

- 여러 문서를 하나로 묶어 한 번에 요약하거나 질의응답을 수행하는 프롬프트.
- 즉, 모든 문서를 “Stuff(채워 넣기)” 방식으로 context에 삽입하여 모델에 전달.

예시 코드

```
from langchain import hub
from langchain.prompts import PromptTemplate

PROMPT_OWNER = "teddynote"
prompt_title = "stuff-documents"

template = """Please summarize the following documents briefly in Korean.

DOCUMENTS:
{context}

SUMMARY:"""

prompt = PromptTemplate.from_template(template)
hub.push(f"{PROMPT_OWNER}/{prompt_title}", prompt)
```

특징

- 단일 프롬프트에 모든 문서를 그대로 포함시켜 모델이 한 번에 처리
- 간단하지만 context 길이가 길면 한계 발생 (대형 문서 비효율적)

---

2. Map Prompt

개념

- 여러 문서를 각각 요약(Map)한 후 나중에 합치는 방식.
- “각 문서를 따로 처리 → 중간 요약 생성” 구조로, 대규모 문서 처리에 적합.

예시 코드

```
prompt_title = "map-prompt"

template = """Please summarize the following document focusing on key id
eas.

DOCUMENT:
{context}

SUMMARY:"""

prompt = PromptTemplate.from_template(template)
hub.push(f"{PROMPT_OWNER}/{prompt_title}", prompt)
```

특징

- 각 문서별 독립 요약 가능
- 이후 Reduce Prompt와 결합하면 효율적인 대규모 문서 요약 파이프라인 완성

---

3. Reduce Prompt

개념

- 여러 개의 요약(Map 단계 결과)을 다시 받아 핵심만 모으는 “축약(Reduce)” 단계 프롬프트.
- Map 단계에서 생성된 여러 요약 → 하나의 통합 요약으로 압축.

예시 코드

```
prompt_title = "reduce-prompt"

template = """You are a professional summarizer.
Combine the following summaries into one concise and comprehensive su
mmary in Korean.

LIST OF SUMMARIES:
{context}

FINAL SUMMARY:"""

prompt = PromptTemplate.from_template(template)
hub.push(f"{PROMPT_OWNER}/{prompt_title}", prompt)
```

특징

- 요약의 요약 → 핵심 정보만 남기기
- Map 단계와 함께 사용 시 효율적 대용량 요약 체인 구성 가능

---

4. Metadata Tagger Prompt

개념

- 텍스트(예: 리뷰, 기사 등)에서 주요 속성이나 키워드를 추출하여 **구조화된 JSON 형식**으로 정리.
- 정보 추출 및 태깅(IE, Information Extraction)용 프롬프트.

예시 코드

```
prompt_title = "metadata-tagger"

template = """Analyze the following product review and extract structured
metadata as JSON.

REVIEW:
{context}

RESPONSE FORMAT (JSON):
{
```

```
  "keywords": [],
  "design_feedback": "",
  "satisfaction_points": "",
  "improvement_points": "",
  "rating": ""
}"""

prompt = PromptTemplate.from_template(template)
hub.push(f"{PROMPT_OWNER}/{prompt_title}", prompt)
```

특징

- 감정 분석, 피드백 정리, 요약된 데이터 추출 등에서 활용
- JSON 구조를 강제하여 결과 일관성 보장

---

5. Chain of Density Prompt

개념

- 문서 요약을 점차 압축하면서 **정보 밀도를 높이는** 단계적 요약 방식.
- “단계별로 더 짧게, 더 밀도 있게” 요약을 반복해 가장 핵심적인 문장만 남긴다.

예시 코드

```
prompt_title = "chain-of-density"

template = """You will be given an article.
Perform progressive summarization by increasing information density in ea
ch step.

ARTICLE:
{context}

1️⃣ Step 1 (General Summary)
2️⃣ Step 2 (Condensed Summary)
3️⃣ Step 3 (High-Density Summary)

Output each step clearly labeled."""
```

```
prompt = PromptTemplate.from_template(template)
hub.push(f"{PROMPT_OWNER}/{prompt_title}", prompt)
```

특징

- 장문의 기사나 논문을 여러 수준의 요약 단계로 나눠볼 때 유용
- 단계별 비교를 통해 “요약 품질 개선” 연구에도 활용 가능

---

6. RAG 문서 프롬프트 (Retrieval-Augmented Generation)

개념

- 검색 기반 생성(RAG) 구조에서 사용되는 표준 프롬프트.
- 외부 문서(Context)를 주고 질문(Question)에 근거한 답변을 생성하도록 함.

예시 코드

```
prompt_title = "rag-prompt"

template = """You are a Q&A assistant.
Use the following context to answer the question.
If the answer cannot be found, say "I don't know."

# Question:
{question}

# Context:
{context}

# Answer:"""

prompt = PromptTemplate.from_template(template)
hub.push(f"{PROMPT_OWNER}/{prompt_title}", prompt)
```

특징

- 문서 검색 + 생성형 AI 결합의 핵심 프롬프트
- RAG 시스템(예: LangChain Retriever, VectorStore 등)에서 표준적으로 사용
- 답변 근거를 context 내에서만 찾도록 유도 → “환각(Hallucination)” 방지

정리 요약

프롬프트 유형	주요 목적	특징
Stuff Documents	여러 문서를 한 번에 요약	간단하지만 context 길이 제약 존재
Map Prompt	문서별 개별 요약	병렬 처리 가능, 대규모 문서에 효율적
Reduce Prompt	여러 요약을 통합 압축	Map과 함께 사용 시 시너지
Metadata Tagger	텍스트에서 구조화된 정보 추출	JSON 포맷 강제, 분석 자동화
Chain of Density	단계적 요약으로 정보 밀도 향상	연구/기사 요약용으로 적합
RAG 문서 프롬프트	검색 기반 생성형 응답	근거 기반 QA, 환각 최소화