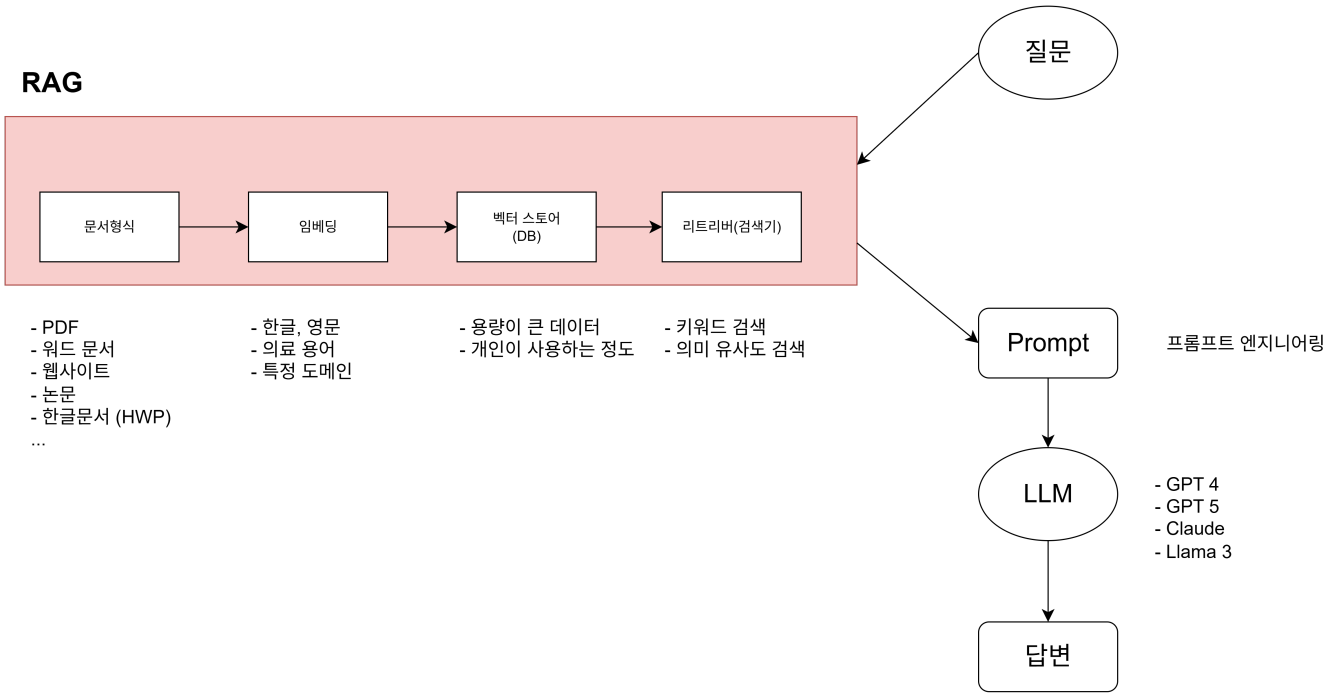
발표 자료 (프롬프트 템플릿)

- LangChain에서 제공하는 PromptTemplate 개념 및 주요 Prompt Template 설명 및 사용 패턴 정리
- LangChain Hub 사용
- Prompt 예제 사이트 공유

1) 프롬프트 템플릿이란 ?

RAG 구조



- 프롬프트 단계: 리트리버(retriever)에서 검색된 문서들을 바탕으로 LLM이 사용할 질문이나 명령을 생성하는 과정
- 검색된 문서에서 적합한 정보 추출 -> 다양한 관점을 하나로 통합 -> 최적화된 답변 이끌어내기
- LLM 프롬프트 템플릿 : AI 모델과의 상호 작용을 표준화하고 효율적으로 만들어주는 구조화된 텍스트 형식
- 프롬프트 템플릿 구조
  - 지시사항 (Instruction)
  - 사용자가 입력한 질문 (Question)
  - 검색된 정보인 문맥 (Context)

```
당신은 질문-답변(Question-Answer) Task를 수행하는 AI 어시스턴트 입니다.
검색된 문맥(context)를 사용하여 질문(question)에 답하세요.
만약, 문맥(context)으로부터 답을 찾을 수 없다면 '모른다'고 말하세요.

한국어로 대답하세요

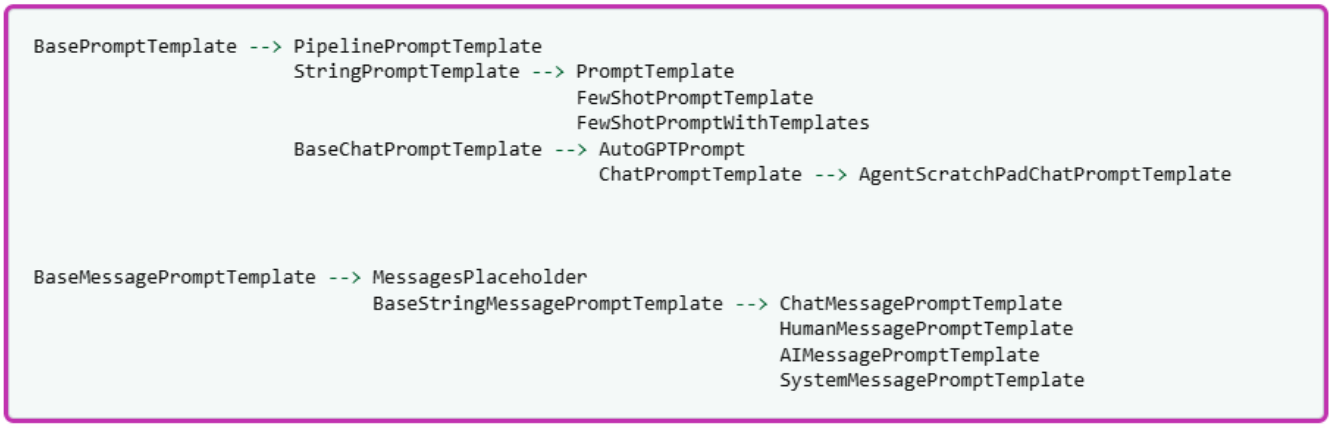
#Question:
{이곳에 사용자가 입력한 질문이 삽입됩니다}

#Context:
{이곳에 검색된 정보가 삽입됩니다}
```

일관된 출력 생성

---

2) LangChain PromptTemplate 구조



- BasePromptTemplate : 공통 인터페이스 (검증, 포맷팅, 옵션)
- StringPromptTemplate : 단일 문자열 템플릿
- ChatPromptTemplate : 다중 메시지 조합 템플릿

BasePromptTemplate은 Runnable 인터페이스의 구현체.

- `invoke` , `batch` , `stream`

왜 사용할까 ?

- 재사용성 : 한번 정의하면 여러 곳에서 변수만 바꿔 사용
- 모듈화 & 유지보수 용이 : 프롬프트 로직과 모델 호출 로직을 분리해 유지보수가 쉬워짐
- 입력 변수 검증 : 내부적으로 Pydantic 기반의 유효성 검사 등도 포함되어 있어, 단순 문자열 포매팅보다 안전하고 구조적.

3) BasePromptTemplate 옵션

- input_variables : 필수 변수 목록 (검증용)
- optional_variables : 선택 변수 (없으면 생략 / 빈값)
- partial_variables : 프롬프트에 미리 바인딩할 기본값 또는 동적값
- input_types : 각 변수의 기대 타입 (문서화/검증에 도움)
- metadata, tags : 추적/로깅/분류용 메타정보
- output_parser: 프롬프트 결과를 구조화하여 파싱
- validate_template: 템플릿-변수 불일치 사전 감지 (기본 True)

4) PromptTemplate

- LLM에 전달할 프롬프트를 동적으로 생성하기 위한 문자열 템플릿 클래스
- 사용자가 제공한 변수 활용해 프롬프트 완성
- 템플릿 구문 형식
  - f-strings (기본)
  - `jinja2` (untrusted source 는 python 실행 시 보안에 취약)
  - `mustache`
- 인스턴스 생성법

```
from langchin_core.prompts import PromptTemplate
# 1. from_template() - recommended

prompt = PromptTemplate.from_template("Say {foo}")
print(prompt.format(foo="bar")) # -> Say bar

# 2. initializer
prompt = PromptTemplate(template="Say {foo}")
```

```
- format : 문자열을 완성하고 리턴
```

- 입력 검증

```
template = "{country}의 수도는 어디인가요?"
prompt = PromptTemplate( template=template, input_variables=["country"], # 검증에 사용 )
```

- 부분 바인딩

```
from datetime import date
prompt = PromptTemplate.from_template( "Today is {today}. Q: {question}"
).partial(today=date.today().isoformat())
```

- 체인 생성 후 실행

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI()
chain = prompt | llm

chain.invoke("대한민국").content
```

```
- invoke : 프롬프트를 조합한 후 PromptValue 형태로 반환하는 wrapper method
```

: "대한민국의 수도는 서울입니다."

파일로부터 template 읽어오기

capital.yaml

```
_type: "prompt"
template: |
  {country}의 수도에 대해서 알려주세요.
  수도의 특징을 다음의 양식에 맞게 정리해 주세요.
  300자 내외로 작성해 주세요.
  한글로 작성해 주세요.

  ----
  [양식]
  1. 면적
  2. 인구
  3. 역사적 장소
  4. 특산품

  #Answer:

input_variables: ["country"]
```

```
from langchain_teddynote.prompts import load_prompt
```

```
prompt2 = load_prompt("prompts/capital.yaml")

print(prompt2.format(country="대한민국"))
```

5) ChatPromptTemplate

채팅 모델에서 사용할 수 있는 프롬프트 템플릿 생성 클래스
여러 역할의 메시지를 구성해 대화 흐름 정의

```
template = ChatPromptTemplate([
    ("system", "You are a helpful AI bot."),
    ("human", "What's the weather today?"),
    ("ai", "It's sunny and warm today."),
])
```

- 메시지는 (role, template_string) 형식
- 프롬프트를 구성하는 메시지들의 시퀀스

1. BaseMessagePromptTemplate
   템플릿 변수, 검증, 부분 치환 등 템플릿 기능을 가장 풍부하게 활용할 때 적합

```
SystemMessagePromptTemplate.from_template("You are a helper")
```

2. BaseMessage
   이미 문자열이 완성된 메시지를 그대로 넣을 때
3. 2-튜플(message type, template)
   `("human", "{user_input}")`, `("system", "You are a helper.")`
   내부적으로 메시지 템플릿으로 변환.
4. 2-튜플(message class, template)
   (SystemMessage, "You are a helper."),
   (HumanMessage, "{question}")
5. 문자열 (자동으로 "human" 역할로 간주)

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.messages import SystemMessage, HumanMessage

prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful assistant."),
    ("human", "{question}"),
    # 위와 동치: (HumanMessage, "{question}"), "{question}"
    SystemMessage("Follow company policy."),       # BaseMessage
```

```
])
msgs = prompt.format_messages(question="How to reset my password?")
```

```
- partial_variables
항상 공통된 방식으로 가져오고 싶은 변수에 사용 (ex. 날짜, 시간)

```python
# 날짜를 반환하는 함수 정의
def get_today():
    return datetime.now().strftime("%B %d")

prompt = PromptTemplate(
    template="오늘의 날짜는 {today} 입니다. 오늘이 생일인 유명인 {n}명을 나열해 주세요. 생년월일을 표기해주세요.",
    input_variables=["n"],
    partial_variables={
        "today": get_today  # dictionary 형태로 partial_variables를 전달
    },
)

# prompt 생성
prompt.format(n=3)

# chain 을 생성합니다.
chain = prompt | llm

# chain 을 실행 후 결과를 확인합니다.
print(chain.invoke({"today": "Jan 02", "n": 3}).content)
```

- MessagePlaceholder
  LangChain은 포맷하는 동안 렌더링할 메시지를 완전히 제어할 수 있는 `MessagePlaceholder` 제공

메시지 프롬프트 템플릿에 어떤 역할을 사용해야 할지 확실하지 않거나 서식 지정 중에 메시지 목록을 삽입하려는 경우 유용

```
from langchain_core.output_parsers import StrOutputParser

from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder

chat_prompt = ChatPromptTemplate.from_messages(
    [
```
```

```
        (
            "system",
            "당신은 요약 전문 AI 어시스턴트입니다. 당신의 임무는 주요 키워드로 대화를
요약하는 것입니다.",
        ),
    MessagesPlaceholder(variable_name="conversation"),
        ("human", "지금까지의 대화를 {word_count} 단어로 요약합니다."),
    ]
)

formatted_chat_prompt = chat_prompt.format(
    word_count=5,
    conversation=[
        ("human", "안녕하세요! 저는 오늘 새로 입사한 테디 입니다. 만나서 반갑습니
다."),
        ("ai", "반가워요! 앞으로 잘 부탁 드립니다."),
    ],
)
```

6) FewShotPromptTemplate

퓨샷 기법 : 프롬프트에서 데모를 제공하여 모델이 더 나은 성능을 발휘하도록 유도하는 상황에서 학습을 가능하게 하는 기법. 복잡한 문맥이나 질문에 대한 답변을 처리할 때 효과적이다.

- Zero-shot prompting : 모델과 상호작용 하는 과정에서 예제나 데모가 포함되지 않음
  Prompt

```
Classify the text into neutral, navgative or positive.
Text : I think the vacation is okay.
Sentiment:
```

Output

```
Neutral
```

- One-shot prompting : 예제나 데모가 1개 포함
- Few-shot prompting : 예제나 데모가 2개 이상 포함
  Prompt

```
A "whatpu" is a small, furry animal native to Tanzania. An example of a
sentence that uses the word whatpu is: We were traveling in Africa and we saw
these very cute whatpus.
```

Output

```
When we won the game, we all started to farduddle in celebrartion.
```

- 책 예제
  스티브 잡스와 아이슈타인 중 누가 더 오래 살았나요 ?
  추가 질문.
  복잡한 질문을 논리적으로 답변하는 과정을 answer 부분에 키와 값의 쌍으로 구성된 딕셔너리 구조로 형식화

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


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": "네이버의 창립자는 언제 태어났나요?",
        "answer": """이 질문에 추가 질문이 필요한가요: 예.
추가 질문: 네이버의 창립자는 누구인가요?
중간 답변: 네이버는 이해진에 의해 창립되었습니다.
추가 질문: 이해진은 언제 태어났나요?
중간 답변: 이해진은 1967년 6월 22일에 태어났습니다.
최종 답변은: 1967년 6월 22일
""",
    },
    {
        "question": "율곡 이이의 어머니가 태어난 해의 통치하던 왕은 누구인가요?",
        "answer": """이 질문에 추가 질문이 필요한가요: 예.
추가 질문: 율곡 이이의 어머니는 누구인가요?
중간 답변: 율곡 이이의 어머니는 신사임당입니다.
추가 질문: 신사임당은 언제 태어났나요?
중간 답변: 신사임당은 1504년에 태어났습니다.
추가 질문: 1504년에 조선을 통치한 왕은 누구인가요?
```

```
중간 답변: 1504년에 조선을 통치한 왕은 연산군입니다.
최종 답변은: 연산군
""",
    },
    {
        "question": "올드보이와 기생충의 감독이 같은 나라 출신인가요?",
        "answer": """이 질문에 추가 질문이 필요한가요: 예.
추가 질문: 올드보이의 감독은 누구인가요?
중간 답변: 올드보이의 감독은 박찬욱입니다.
추가 질문: 박찬욱은 어느 나라 출신인가요?
중간 답변: 박찬욱은 대한민국 출신입니다.
추가 질문: 기생충의 감독은 누구인가요?
중간 답변: 기생충의 감독은 봉준호입니다.
추가 질문: 봉준호는 어느 나라 출신인가요?
중간 답변: 봉준호는 대한민국 출신입니다.
최종 답변은: 예
""",
    },
]

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

print(example_prompt.format(**examples[0]))
```

```
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
```

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

1.

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

```
print(final_prompt)
```

2.

```
answer = llm.stream(final_prompt)
stream_response(answer)
```

3.

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

# chain 생성
chain = prompt | llm | StrOutputParser()

# 결과 생성
answer = chain.stream(
    {"question": "Google이 창립된 연도에 BillGates의 나이는 몇 살인가요?"}
)

stream_response(answer)
```

```
이 질문에 추가 질문이 필요한가요: 예. 추가 질문: Google이 창립된 연도는 언제인가요? 중간 답변: Google은 1998년에 창립되었습니다. 추가 질문: Bill Gates는 언제 태어났나요? 중간 답변: Bill Gates는 1955년 10월 28일에 태어났습니다. 추가 질문: 1998년에 Bill Gates의 나이는 몇 살이었나요? 중간 답변: 1998년에서 1955년을 빼면 43이므로, Bill Gates는 1998년에 43세였습니다. 최종 답변은: 43세
```

- 예제 선택기 (example selector)
  모든 예시가 프롬프트에 그대로 들어가면 비용이 많이 발생
  - SemanticSimilarityExampleSelector : 입력된 질문과 의미가 가장 유사한 예시 선택 사용
  - MaxMarginalRelevanceExampleSelector : 유사도 뿐만 아니라 예시의 다양성까지 고려하여 선택 사용

```
from langchain_core.example_selectors import (
    MaxMarginalRelevanceExampleSelector,
    SemanticSimilarityExampleSelector,
)
```

```python
from langchain_openai import OpenAIEmbeddings
from langchain_chroma import Chroma

# Vector DB 생성 (저장소 이름, 임베딩 클래스)
chroma = Chroma("example_selector", OpenAIEmbeddings())

example_selector = SemanticSimilarityExampleSelector.from_examples(
    # 여기에는 선택 가능한 예시 목록이 있습니다.
    examples,
    # 여기에는 의미적 유사성을 측정하는 데 사용되는 임베딩을 생성하는 임베딩 클래스가 있습니다.
    OpenAIEmbeddings(),
    # 여기에는 임베딩을 저장하고 유사성 검색을 수행하는 데 사용되는 VectorStore 클래스가 있습니다.
    Chroma,
    # 이것은 생성할 예시의 수입니다.
    k=1,
)

question = "Google이 창립된 연도에 Bill Gates의 나이는 몇 살인가요?"

# 입력과 가장 유사한 예시를 선택합니다.
selected_examples = example_selector.select_examples({"question": question})

print(f"입력에 가장 유사한 예시:\n{question}\n")
for example in selected_examples:
    print(f'question:\n{example["question"]}')
    print(f'answer:\n{example["answer"]}')
```

```
- OpenAIEmbeddings : 질문이나 텍스트를 벡터로 변환하기 위해 임베딩 생성하는 클래스
- Chroma : 벡터 스토어 데이터베이스
```

- **MMR(Maximal Marginal Relevance) 알고리즘**
  : 다양성과 관련성 모두 고려하여 결과를 선택하는 방법.
  알고리즘의 목표는 비슷한 내용이 반복되지 않도록 하면서도(다양성) 사용자가 찾고 있는 질문과 가장 잘 맞는 (관련성) 정보를 제공하는 것
- 관련성(relevance): 사용자가 입력한 검색어나 주제와 문서가 얼마나 잘 맞는지를 평가하는 기준입니다. 문서가 사용자의 검색 내용과 얼마나 잘 일치 하는지에 따라 점수를 매기고, 그 점수가 높은 문서가 먼저 선택 됩니다.
- 다양성(diversity): 이미 선택된 문서와 다른 새로운 문서 간의 유사성을 계산해서, 이미 선택된 문서와 비슷한 내용의 문서는 다시 선택 되지 않도록 합니다.

MMR 알고리즘 활용 분야.

- 검색 엔진 : 사용자 쿼리에 대해 관련성 높은 결과를 제공하면서 중복을 최소화해 다양한 정보를 함께 보여줄 수 있음
- 대량의 문서 집합을 처리할 때 중복되는 내용을 줄이면서도 핵심 정보를 추출하여 요약
- 추천 시스템 : 사용자에게 관련성 높은 아이템을 제공하면서도 서로 다른 유형의 추천을 통해 선택의 폭을 넓혀줌
- 목적에 맞는 예제 선택기
  예시와 답변의 유사도를 계산할 때 instruction과 input을 합산해서 고려한다. 부정확성 증가
  예제 선택기를 custom해서 instruction 만 고려.

```
from langchain_teddynote.prompts import CustomExampleSelector

# 커스텀 예제 선택기 생성
custom_selector = CustomExampleSelector(examples, OpenAIEmbeddings())

# 커스텀 예제 선택기를 사용했을 때 결과
custom_selector.select_examples({"instruction": "다음 문장을 회의록 작성해 주세요"})

```python
example_prompt = ChatPromptTemplate.from_messages(
    [
        ("human", "{instruction}:\n{input}"),
        ("ai", "{answer}"),
    ]
)

custom_fewshot_prompt = FewShotChatMessagePromptTemplate(
    example_selector=custom_selector,  # 커스텀 예제 선택기 사용
    example_prompt=example_prompt,  # 예제 프롬프트 사용
)

custom_prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "You are a helpful assistant.",
        ),
        custom_fewshot_prompt,
        ("human", "{instruction}\n{input}"),
    ]
)

chain = custom_prompt | llm
```
```

```
question = {
    "instruction": "회의록을 작성해 주세요",
    "input": "2023년 12월 26일, ABC 기술 회사의 제품 개발 팀은 새로운 모바일 애플리케이션 프로젝트에 대한 주간 진행 상황 회의를 가졌다. 이 회의에는 프로젝트 매니저인 최현수, 주요 개발자인 황지연, UI/UX 디자이너인 김태영이 참석했다. 회의의 주요 목적은 프로젝트의 현재 진행 상황을 검토하고, 다가오는 마일스톤에 대한 계획을 수립하는 것이었다. 각 팀원은 자신의 작업 영역에 대한 업데이트를 제공했고, 팀은 다음 주까지의 목표를 설정했다.",
}

# 실행 및 결과 출력
stream_response(chain.stream(question))
```

7) LangChain Hub

https://smith.langchain.com/hub

LangSmith에 회원 가입을 하고 API 키를 발급 받았다면, 검색 창에서 필요한 내용을 검색해서 원하는 프롬프트를 골라 사용할 수 있다.

Top Download
https://smith.langchain.com/hub/rlm/rag-prompt
rlm : LangChain 공식 튜토리얼에 등록되어 있어 사람들이 쉽게 써 볼수 있기 때문에 다운로드 수가 높고 유용하긴 하지만, 매우 평균적인 프롬프트라는 단점

prompt playgroud
https://smith.langchain.com/o/14259d9d-8aac-4e34-b879-3641af639c00/playground

8) Prompt Example 사이트

핵심 가이드 & 예제
Prompt Engineering Guide
Learn Prompting

모델 제공사 공식 자료
Anthropic Prompt Library
OpenAI Cookbook
Google Gemini Guide

9) 그 외 LangchainTemplate

PiplelinePromptTemplate

- 여러 템플릿을 순차적으로 조합
- 체인이 길어질 때 프롬프트를 모듈 단위로 재사용 가능

DynamicPromptTemplate

- 상황에 맞게 프롬프트 동적 생성
- BaseExampleSelector와 결합해 입력에 따라 예시 선택