고급 프롬프트 공학

퓨 샷 학습, Chain of Thought, 자기 일관성, Tree of Thoughts

충북대학교 의과대학
박 승

충북대학교
CHUNGBUK NATIONAL UNIVERSITY

학습 목표

1. 제로 샷의 개념을 이해하고 이를 활용하여 응답 정확도를 향상시킬 수 있다.
2. Chain of Thought를 활용하여 복잡한 추론 과정을 명시적으로 생성할 수 있다.
3. 자기 일관성의 작동 메커니즘을 설명하고, 이를 통해 최적의 답을 선택할 수 있다.
4. Tree of Thoughts의 구조를 이해하고 적용하여 문제 해결 성능을 개선할 수 있다.
5. 고급 프롬프트 공학 기법을 구현하고, 실제 문제 해결에 적용할 수 있다.

CHUNGBUK
NATIONAL UNIVERSITY

고급 프롬프트 공학

프롬프트 공학

- 자연어 처리 시스템에서 원하는 출력 결과를 얻기 위해 입력 프롬프트를 설계하고 최적화하는 기술
- AI 모델의 성능을 극대화하고 사용자가 의도한 결과를 정확하게 도출하는 데 중요

프롬프트 가이드라인

고급 프롬프트 공학

1. 제로 샷 (zero-shot) 프롬프트

- 어떠한 데모나 예제를 제공하지 않고 과업 지침을 직접 LLM에게 전달

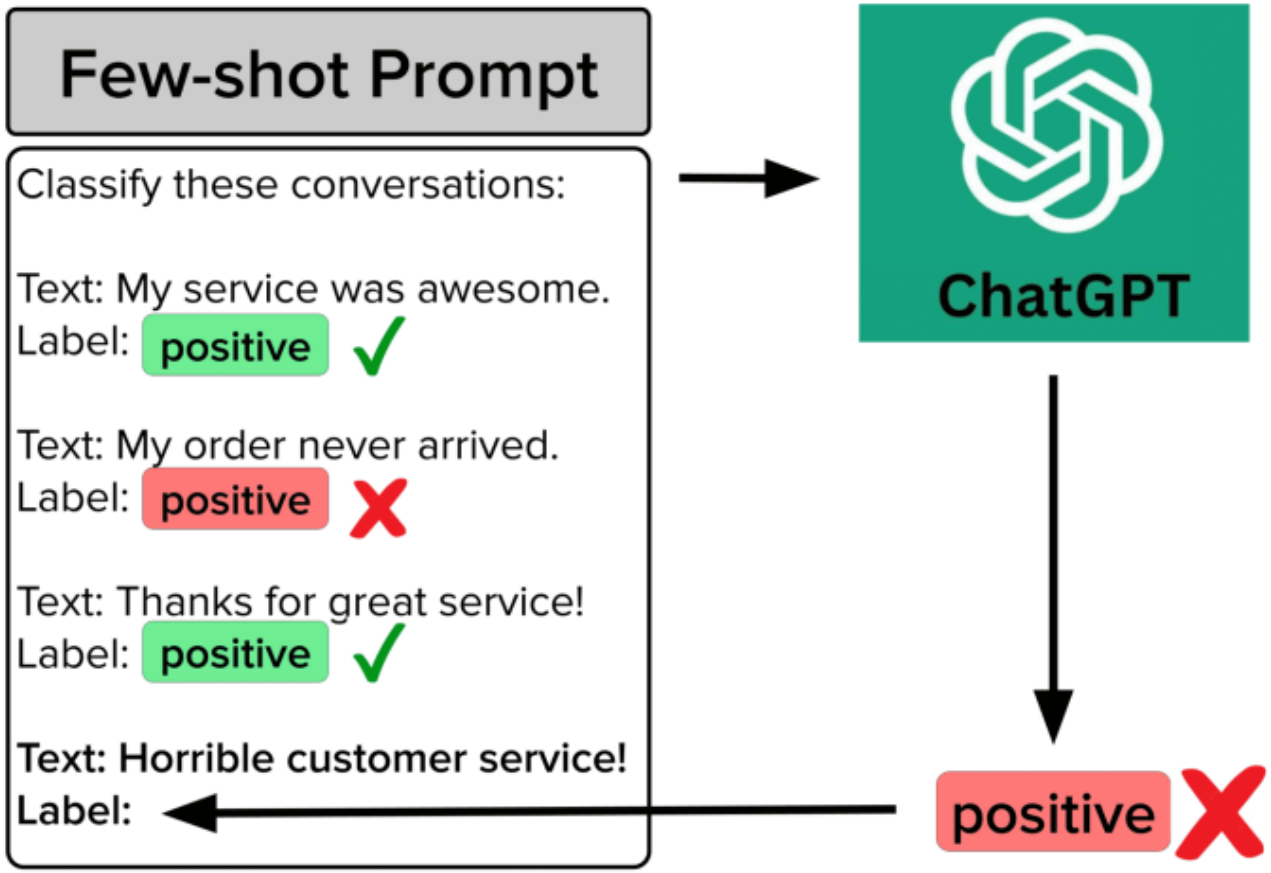
```
import os
from langchain_core.prompts import PromptTemplate
from langchain_openai import ChatOpenAI

# 환경변수에 OpenAI API 키 설정
api_key = "sk-proj-zcYjewY8DYjQ4gjIKzBeASFA1ru_McHtN4kA50N-QjVLGEXQx9pivZHXI7ste6bOI7yOlT86l

# GPT 모델 설정
model = ChatOpenAI(model="gpt-4-turbo")

# 입력 텍스트의 감정을 분류하는 프롬프트 템플릿 정의
prompt = PromptTemplate(
    input_variables=["text"],
    template="""Classify the sentiment of this text: {text}.
    Provide only the sentiment classification as 'positive', 'negative', or 'neutral'."""
)

# 프롬프트와 모델을 결합하여 체인 구성
chain = prompt | model

# 예시 텍스트를 이용해 체인 실행
result = chain.invoke({"text": "I hated that movie, it was terrible!"})
print(result.content.strip())
```

```
negative
```

고급 프롬프트 공학

2. 퓨 샷 (few-shot) 학습

- 명시적인 지침 없이 작업과 관련된 몇 가지 입력-출력 예제만을 LLM에게 제공
- 답변 품질과 정확도를 크게 개선 가능



고급 프롬프트 공학

2. 퓨 샷 (few-shot) 학습

- 1. GPT 모델 설정 및 예시 데이터 입력

```python
import os
from langchain_core.prompts import PromptTemplate, FewShotPromptTemplate
from langchain_openai import ChatOpenAI

# 환경변수에 OpenAI API 키 설정
api_key = "sk-proj-zcYjewY8DYjQ4gjIKzBeASFA1ru_McHtN4kA50N-QjVLGEXQx9pivZHXI7ste6bOI7yOlT86l3T3BlbkFJ-bs1tH1w_R
os.environ["OPENAI_API_KEY"] = api_key

# GPT 모델 설정
model = ChatOpenAI(model="gpt-4-turbo")

# 예시 데이터에 사용할 개별 프롬프트 템플릿 정의
example_prompt = PromptTemplate(
    input_variables=["input", "output"],
    template="Text: {input}\nSentiment: {output}"
)

# Few-shot 학습에 사용할 예시 데이터
examples = [
    {"input": "I absolutely love the new update! Everything works seamlessly.", "output": "positive"},
    {"input": "It's okay, but I think it could use more features.", "output": "neutral"},
    {"input": "I'm disappointed with the service, I expected much better performance.", "output": "negative"}
]
```

2. 퓨 샷 (few-shot) 학습

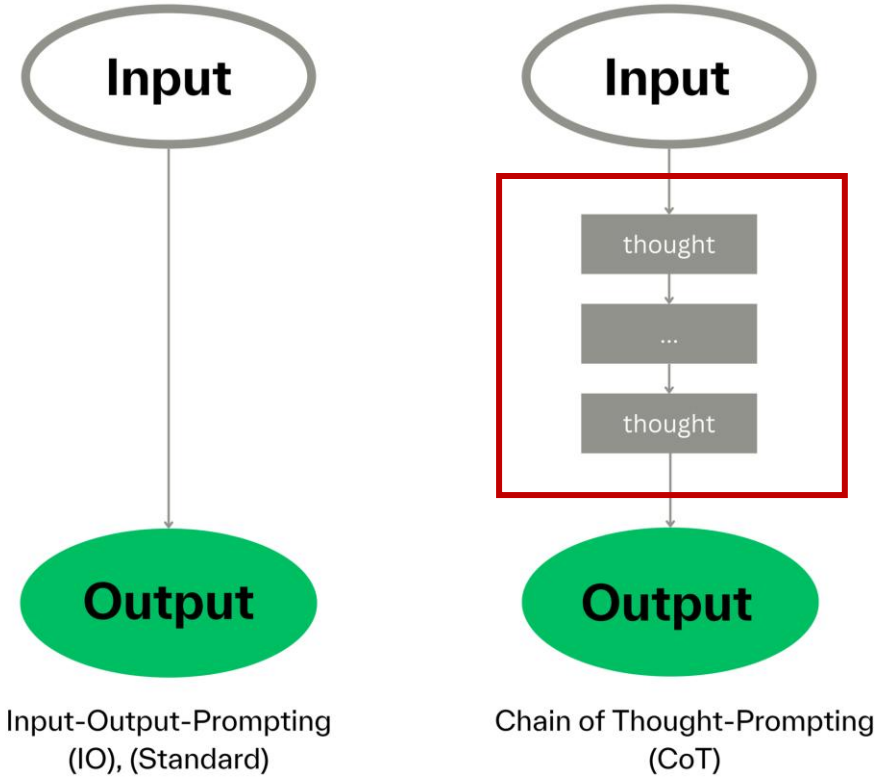
- 2. 프롬프트 템플릿 설정 및 체인 실행

```
# FewShotPromptTemplate 설정
prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    suffix="""Classify the sentiment of this text: {text}.
    Provide only the sentiment classification as 'positive', 'negative', or 'neutral'.""",
    input_variables=["text"]
)

# 프롬프트와 모델을 결합하여 체인 구성
chain = prompt | model

# 예시 텍스트를 이용해 체인 실행
result = chain.invoke({"text": "I hated that movie, it was terrible!"})
print(result.content.strip())
```

```
negative
```

3. Chain of Thought (CoT) 프롬프트

- 중간 추론 단계를 포함한 응답을 접두사로 추가하여 모델이 추론할 수 있는 능력 부여



3. Chain of Thought (CoT) 프롬프트

- 1. GPT 모델 및 CoT 지시문 설정

```
import os
from langchain_core.prompts import PromptTemplate
from langchain_openai import ChatOpenAI

# 환경변수에 OpenAI API 키 설정
api_key = "sk-proj-zcYjewY8DYjQ4gjIKzBeASFA1ru_McHtN4kA50N-QjVLGEXQx9pivZHXI7s
os.environ["OPENAI_API_KEY"] = api_key

# GPT 모델 설정
model = ChatOpenAI(model="gpt-4-turbo")

# Chain of Thought 지시문 설정
cot_instruction = "Let's think step by step!"

# Chain of Thought 방식의 질문 템플릿 생성
reasoning_prompt = "{question}\n" + cot_instruction
prompt = PromptTemplate(
    template=reasoning_prompt,
    input_variables=["question"]
)
```

3. Chain of Thought (CoT) 프롬프트

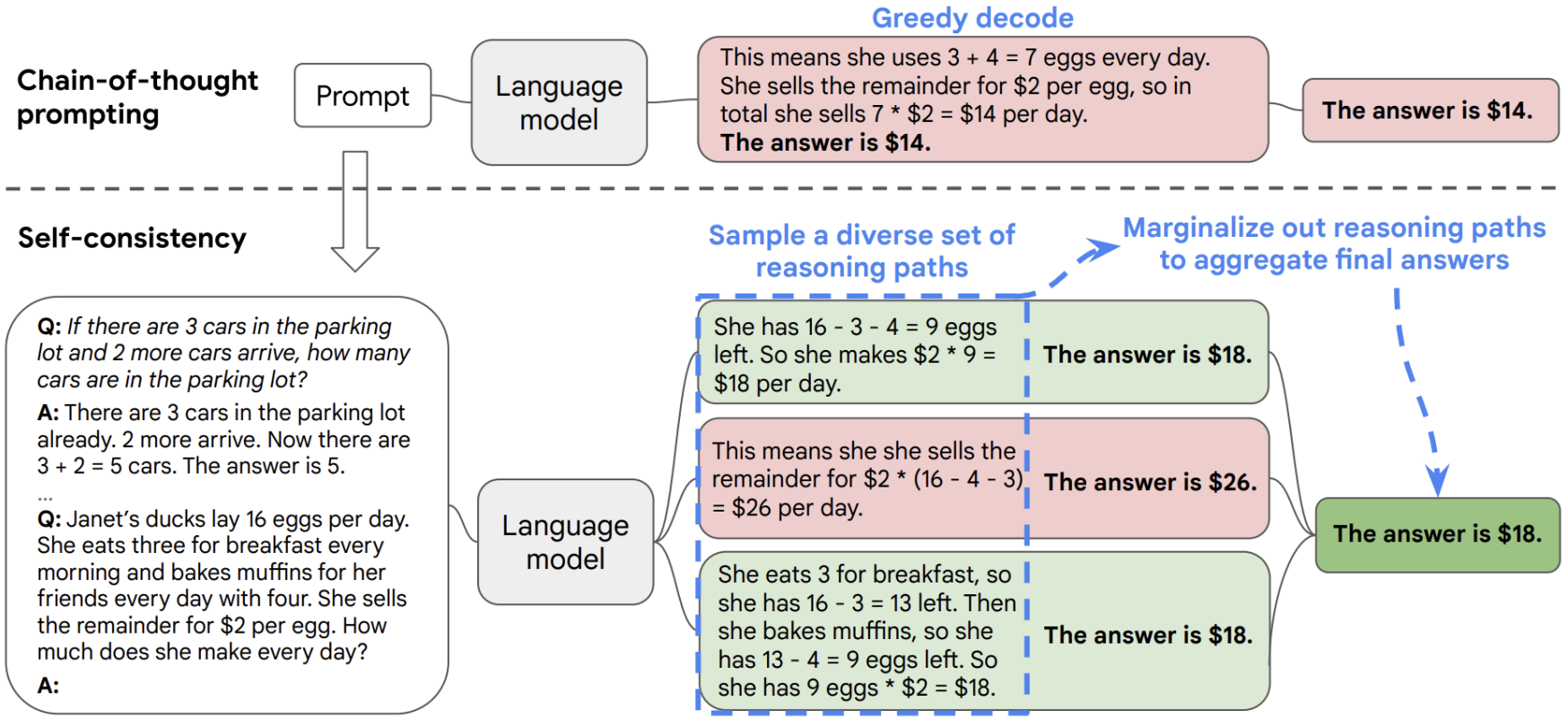
- 2. 체인 구성 및 실행

```python
# 프롬프트와 모델을 결합하여 체인 구성
chain = prompt | model

# 체인을 실행하여 문제 풀이 결과 출력
result = chain.invoke({
    "question": """There were 5 apples originally. I ate 2 apples.
    My friend gave me 3 apples. How many apples do I have now?"""
})
print(result.content.strip())
```

```
Let's break it down step by step:

1. Initially, you had 5 apples.
2. You ate 2 apples, so we subtract 2 from the initial total:
   \( 5 - 2 = 3 \) apples left.

3. Then, your friend gave you 3 more apples. We add these to your current total of apples:
   \( 3 + 3 = 6 \) apples.

So, now you have 6 apples in total.
```

4. 자기 일관성 (Self-consistency)

- 질문에 대해 여러 후보 답변을 생성한 후, 가장 일관된 또는 가장 빈번한 답변을 최종 출력으로 선택



고급 프롬프트 공학

4. 자기 일관성 (Self-consistency)

- 1. 라이브러리 설정 & 여러 솔루션을 생성하는 첫 번째 체인 구성

```python
import os
from langchain_core.prompts import PromptTemplate
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser

# 환경변수에 OpenAI API 키 설정
api_key = "sk-proj-zcYjewY8DYjQ4gjIKzBeASFA1ru_McHtN4kA50N-QjVLGEXQx9pivZHXI7ste6bOI
os.environ["OPENAI_API_KEY"] = api_key

# GPT 모델 설정
model = ChatOpenAI(model="gpt-4-turbo")

# 첫 번째 체인: 여러 솔루션 생성 프롬프트
solutions_template = """
Generate {num_solutions} distinct answers to this question:
{question}

Solutions:
"""
solutions_prompt = PromptTemplate(
    template=solutions_template,
    input_variables=["question", "num_solutions"]
)
solutions_chain = solutions_prompt | model | StrOutputParser()
```

고급 프롬프트 공학

4. 자기 일관성 (Self-consistency)

- 2. 여러 솔루션 중 다수 결과를 채택하는 두 번째 체인 구성

```python
# 두 번째 체인: 솔루션의 일관성 확인 프롬프트
consistency_template = """
For each answer in the solutions below, count how many times it occurs. Finally, choose the answer that occurs most.

Solutions:
{solutions}

Most frequent solution:
"""
consistency_prompt = PromptTemplate(
    template=consistency_template,
    input_variables=["solutions"]
)
consistency_chain = consistency_prompt | model | StrOutputParser()

# 두 체인을 RunnableSequence로 직접 연결하여 순차적으로 실행하고 중간 결과 출력
solutions = solutions_chain.invoke({
    "question": "Which year was the Declaration of Independence of the United States signed?",
    "num_solutions": "5"
})
print("Generated solutions:")
print(solutions.strip())

final_result = consistency_chain.invoke({"solutions": solutions})

print("\nMost frequent solution:")
print(final_result.strip())
```

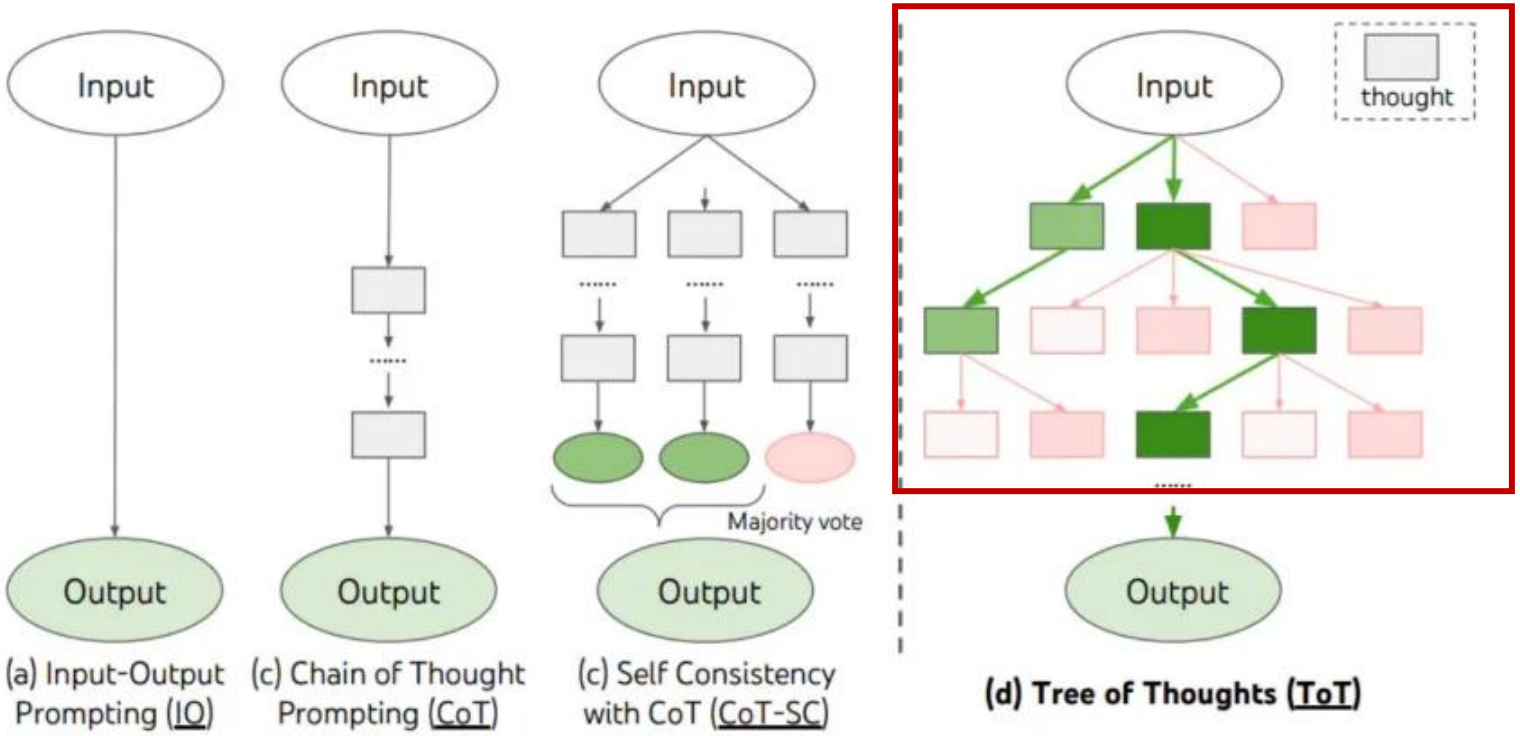
4. 자기 일관성 (Self-consistency)

- 3. 실행 결과

```
Generated solutions:
1. The Declaration of Independence was officially adopted and signed on July 4, 1776.
2. The historic signing of the United States Declaration of Independence occurred in the year 1776.
3. In 1776, the founding fathers signed the Declaration of Independence, marking the birth of the nation.
4. The document declaring the United States as an independent nation, the Declaration of Independence, was signed in the year 1776.
5. The United States Declaration of Independence was formally signed in the year 1776, establishing the 13 colonies' independence from British rule.

Most frequent solution:
Each solution mentions that the Declaration of Independence was signed in 1776. This is a consistent answer across all five solutions. Thus, the answer
"1776", as the year the Declaration of Independence was signed, occurs five times.

Most frequent solution: The Declaration of Independence was signed in the year 1776.
```

5. Tree of Thoughts (ToT) 프롬프트

- 문제를 해결하기 위해 다양한 사고 경로(branch)를 탐색



5. Tree of Thoughts (ToT) 프롬프트

- 1. 라이브러리 설정

```python
import os
from langchain_core.prompts import PromptTemplate
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser

# 환경변수에 OpenAI API 키 설정
api_key = "sk-proj-zcYjewY8DYjQ4gjIKzBeASFA1ru_McHtN4kA50N-QjVLGEXQx9pivZHXI7ste6bOI7yOlT8613T3Blb
os.environ["OPENAI_API_KEY"] = api_key

# GPT 모델 설정
model = ChatOpenAI(model="gpt-4-turbo")
```

5. Tree of Thoughts (ToT) 프롬프트

- 2. 솔루션 생성 및 평가 프롬프트 설정

```
# 솔루션 생성 프롬프트 설정
solutions_prompt = PromptTemplate(
    template="""
    Generate {num_solutions} distinct solutions for the problem: {problem}.
    Consider factors like: {factors}

    Solutions:
    """,
    input_variables=["problem", "factors", "num_solutions"]
)

# 솔루션 평가 프롬프트 설정
evaluation_prompt = PromptTemplate(
    template="""
    Evaluate each solution below by analyzing pros, cons, feasibility, and probability of success.

    Solutions:
    {solutions}

    Evaluations:
    """,
    input_variables=["solutions"]
)
```

CHUNGBUK NATIONAL UNIVERSITY

5. Tree of Thoughts (ToT) 프롬프트

- 3. 심층적 추론 및 솔루션 순위화 프롬프트 설정

```
# 심층적 추론 프롬프트 설정
reasoning_prompt = PromptTemplate(
    template="""
    For the most promising solutions below, explain scenarios, implementation strategies, partnerships needed, and potential obstacles.

    Evaluations:
    {evaluations}

    Enhanced Reasoning:
    """,
    input_variables=["evaluations"]
)

# 솔루션 순위화 프롬프트 설정
ranking_prompt = PromptTemplate(
    template="""
    Rank the solutions below from most to least promising based on evaluations and enhanced reasoning.

    Enhanced Reasoning:
    {enhanced_reasoning}

    Ranked Solutions:
    """,
    input_variables=["enhanced_reasoning"]
)
```

5. Tree of Thoughts (ToT) 프롬프트

- 4. 체인 구축 및 실행

```python
# 각 단계 결과 명시적으로 전달하는 함수 설정
def chain_executor(problem, factors, num_solutions):
    solutions = (solutions_prompt | model | StrOutputParser()).invoke({
        "problem": problem,
        "factors": factors,
        "num_solutions": num_solutions
    })

    evaluations = (evaluation_prompt | model | StrOutputParser()).invoke({"solutions": solutions})
    enhanced_reasoning = (reasoning_prompt | model | StrOutputParser()).invoke({"evaluations": evaluations})
    ranked_solutions = (ranking_prompt | model | StrOutputParser()).invoke({"enhanced_reasoning": enhanced_reasoning})

    return ranked_solutions

# 체인 실행 및 결과 출력
result = chain_executor(
    problem="Prompt engineering",
    factors="Requirements for high task performance, low token use, and few calls to the LLM",
    num_solutions="3"
)

print(result.strip())
```

5. Tree of Thoughts (ToT) 프롬프트

- 5. 실행 결과

```
1. **Model-Aided Prompt Design Tool**
   - **Reasoning:** This solution appears highly practical and user-centric, providing an accessible platform that continuously improves based on user fe
edback. It addresses the need for effective communication with AI systems by simplifying prompt creation, potentially benefiting a broad range and large
number of users. While the maintenance cost and user adoption are concerns, these are relatively manageable compared to technological complexities involv
ed in other scenarios. Successful implementation and user engagement could make this a very impactful tool.

2. **Contextual Prompt Caching Mechanism**
   - **Reasoning:** The development of a caching mechanism that understands context involves significant technological innovation and offers substantial
speed and efficiency improvements in using AI models. This solution ranks slightly lower than the Model-Aided Prompt Design Tool primarily due to the hig
her barrier in terms of technological complexity and data privacy issues, which can be challenging and costly to address. However, the potential gains in
performance make it promising.

3. **Incremental Prompt Engineering Framework**
   - **Reasoning:** This framework has the potential to fine-tune prompts to a high degree of accuracy and effectiveness, which is valuable for optimizin
g user interactions with AI. Nevertheless, the continuous need for computing resources and the slow rate of improvement due to its incremental nature are
significant drawbacks. Industries that require rapid innovation and adaptation could find the gradual improvement process too slow, making this solution
somewhat less promising compared to the others that offer more immediate results or broader usability.

Each solution presents a distinct approach to enhancing interactions with AI through prompt engineering, with considerations varying from user interface
ease to backend technological advancements. While all hold potential, the feasibility and impact of each can vary based on the specific context in which
they are implemented.
```

감사합니다

Q&A

충북대학교
CHUNGBUK NATIONAL UNIVERSITY