

LangChain 프레임워크 활용 방안 연구

Lesson 2 프로그래밍을 활용한 Prompt 실행

- Prompt engineering 이해하기
- 여러 종류의 prompt와 구조
- One, two, or few-shot 학습을 활용하여 LLM 응답 향상시키기
- Prompt 실제 적용 예시 이해하기

1. Prompt Engineering 이해하기

LangChain으로 만든 애플리케이션은 프롬프트를 통해 LLM과 상호작용한다. 이는 마치 똑똑하지만 경험이 부족한 동료에게 지시를 내리는 것과 비슷하다. 정확하고 관련 있는 응답을 얻으려면 프롬프트는 명확하고, 구체적이며, 작업에 적합해야 한다. 효과적인 프롬프트 작성은 매우 중요하며, LangChain은 프롬프트를 통해 애플리케이션의 동작을 제어할 수 있다.

프롬프트 엔지니어링, 즉 LLM의 출력을 유도하기 위해 프롬프트를 설계하고 다듬는 일은 LLM 기반 애플리케이션 개발의 핵심 기술이다. 원하는 결과를 얻기 위해 프롬프트를 만들고, 테스트하고, 계속 수정하는데 많은 시간을 사용하게 된다. LangChain은 이 과정을 중심에 두고, 단순한 지시부터 고급 계획과 추론이 필요한 복잡한 템플릿까지 다양한 프롬프트를 만들고 관리할 수 있도록 여러 도구를 제공한다.

2. 프로그래밍을 활용한 프롬프팅

LangChain 애플리케이션은 잘 설계된 프롬프트에 의존해서, 다음 단계에서 사용할 수 있는 결과물을 생성한다. ChatGPT 같은 사용자 인터페이스에 직접 입력하는 prompt와 달리, LangChain에서는 프롬프트를 코드로 생성해서 LLM에 보낼 수 있다. 다음에서는 실제 프로그래밍 예제를 통해 Prompt Engineering에 대하여 이해해 보고자 한다. 아래 소개하게 될 예시를 따라하기 위해서는 먼저 OpenAI API key와 Jupyter notebook 환경 설정이 필요하다.

2-1 Jupyter Notebook 환경설정

- Window cmd 또는 Anaconda Jupyter 터미널에서 다음 예제와 같이 폴더를 생성하고 생성된 폴더로 이동한다.

```
C:\Github\building-llm-applications\ch02>
```

- Virtual Environment 생성하고 활성화(activate) 하기

```
C:\Github\building-llm-applications\ch02>python -m venv env_ch02
C:\Github\building-llm-applications\ch02>.\env_ch02\Scripts\activate
```

- 위 과정이 완료되면 아래와 같은 결과를 볼 수 있다. 괄호 안에 있는 (env_ch02) Virtual Environment가 활성화 되었음을 보여준다.

```
(env_ch02) C:\Github\building-llm-applications\ch02>
```

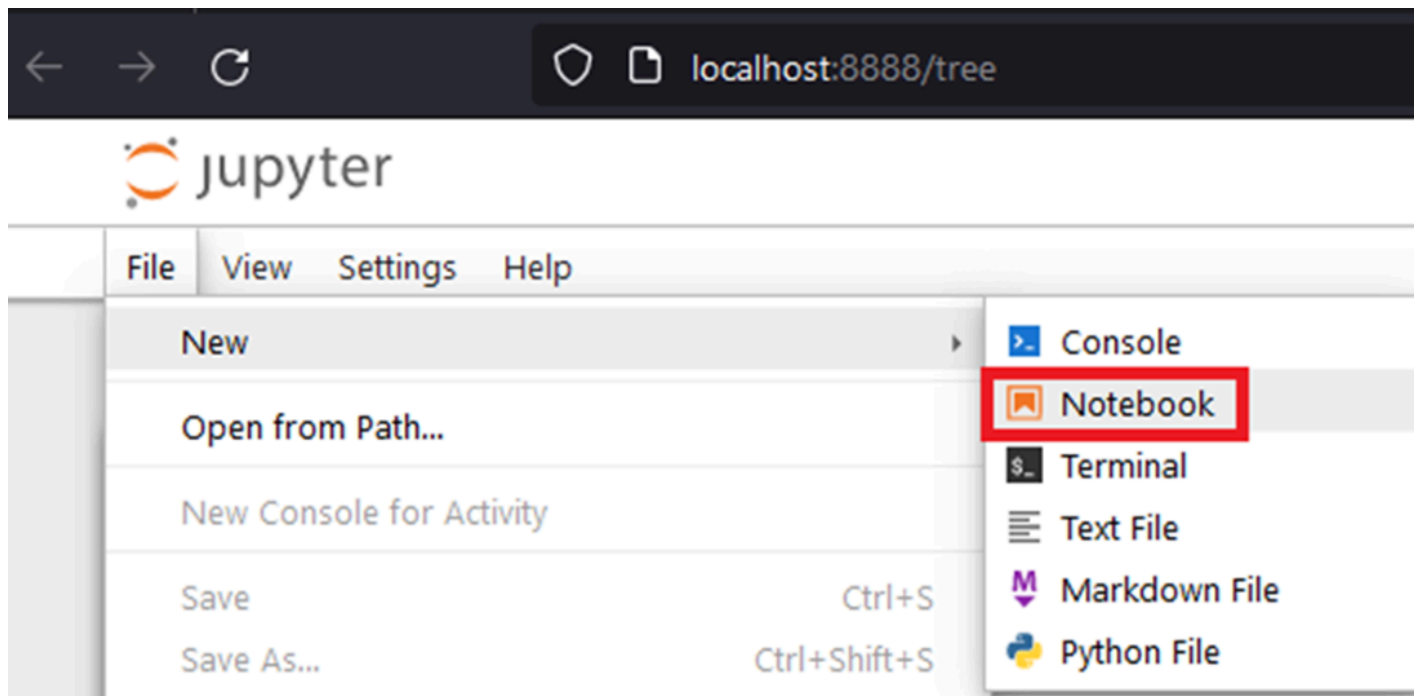
- 예제 실행에 필요한 라이브러리를 인스톨한다.

```
(env_ch02) C:\Github\building-llm-applications\ch02>pip install notebook openai langchain
langchain-openai
```

- 관련 라이브러리 인스톨이 완료되면 "jupyter notebook"을 터미널에 입력하여 Notebook을 시작한다.

```
(env_ch02) C:\Github\building-llm-applications\ch02>jupyter notebook
```

- File > New > Notebook 으로 이동하여 새로운 노트북을 생성하고 파일 이름을 prompt_example.ipynb로 변경한다.



2-2. 가장 기본적인 Prompt 실행 예시

- 필요한 라이브러리를 불러들이고 OpenAI API Key를 입력한다.

```
from openai import OpenAI
import getpass

OPENAI_API_KEY = getpass.getpass('Enter your OPENAI_API_KEY')
```

- OpenAI 클라이언트 셋업한다.

```
client = OpenAI(api_key=OPENAI_API_KEY)
```

- 다음과 같은 방법을 활용하여 prompt를 생성하고 사용자의 질문과 함께 실행할 수 있다.

```
prompt_input = """Write a short message to remind users to be vigilant about phishing attacks."""

response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": prompt_input}
    ],
    temperature= 0.7,
    max_tokens= 400
)

print(response.choices[0].message.content)
```

- 코드를 실행했을 때 다음과 같은 결과 값을 얻을 수 있다.

```
Attention users,
Please be vigilant and stay alert regarding phishing attacks. Cybercriminals are
constantly devising new ways to trick you into revealing sensitive information such as
passwords, credit card details, and personal data. Remember to never click on suspicious
links, open attachments from unknown sources, or share confidential information via
email or other unsecured platforms. Stay protected by regularly updating your passwords,
enabling two-factor authentication whenever possible, and keeping your antivirus
software up to date. If you suspect any phishing attempts, report them immediately to
```

the relevant authorities or your IT department.
Stay safe and secure online!
Best regards,
[Your Name/Company Name]

2-2. LangChain PromptTemplate을 활용한 실행

이 부분은 Lesson1에서 설명했기 때문에 간단한 코드 예시만 보여주고 넘어가도록 한다.

```
from langchain_core.prompts import PromptTemplate
prompt_template = PromptTemplate.from_template("You are an experienced copywriter. Write a {num_words} words summary the the following text, using a {tone} tone: {text}")

prompt = prompt_template.format(text=segovia_aqueduct_text, num_words=20, tone="knowledgeable and engaging")

response = llm.invoke(prompt)
print(response.content)
```

3. 프롬프트의 종류

LangChain 애플리케이션에서 좋은 결과를 얻으려면 효과적인 프롬프트를 만드는 것이 중요하다. 텍스트 분류, 감정 분석, 요약, 텍스트 생성, 질문 응답 등 어떤 작업이든 간에, 각각의 작업에는 잘 설계된 프롬프트가 필요하다. 다음 섹션에서는 이러한 특정 작업에 맞춘 프롬프트를 어떻게 만들 수 있는지 살펴보고, 애플리케이션이 의도한 대로 작동하도록 만드는 방법을 알아볼 것이다.

3-1. 텍스트 분류(Text Classification)

텍스트 분류 작업에서는 입력된 텍스트를 미리 정의된 여러 범주 중 하나로 분류하는 것을 목표로 한다. 아래 프롬프트는 그 예시를 보여준다.

Instruction: 다음 텍스트를 다음 범주 중 하나로 분류하십시오: 역사(history), 기술(tech), 원예(gardening).

Text: 헤드폰은 음악 애호가와 게이머 모두에게 몰입감 있는 오디오 경험을 제공한다.

출력:

"헤드폰은 음악 애호가와 게이머 모두에게 몰입감 있는 오디오 경험을 제공한다"는 텍스트는 기술과 관련된 제품과 기능에 대해 다루고 있으므로 "기술(tech)"로 분류되어야 한다.

출력이 지나치게 자세하다고 생각되는 경우, 프롬프트를 수정하여 범주만 출력하도록 할 수 있다.

Instruction: 다음 텍스트를 다음 범주 중 하나로 분류하십시오: 역사(history), 기술(tech), 원예(gardening).

Text: 헤드폰은 음악 애호가와 게이머 모두에게 몰입감 있는 오디오 경험을 제공한다.

Output: 범주만 표시하십시오

Output:
기술(Tech)

이제, 동일한 프롬프트가 적용된 상태에서 텍스트만 바뀌서 문장을 분류하라고 지시하면 다음과 같은 결과를 얻을 수 있다.

Instruction: 다음 텍스트를 다음 범주 중 하나로 분류하십시오: 역사(history), 기술(tech), 원예(gardening).

Text: 율리우스 카이사르는 로마 공화정의 몰락과 로마 제국의 탄생에 중요한 역할을 한 로마의 장군이자 정치가였다.

Output: 범주만 표시하십시오

Output:
역사 (History)

3-2. 다른 종류의 Prompt 기법 간단 요약

위에 소개한 프롬프트 외에도 다음과 같은 프롬프트 종류가 있다.

프롬프트 종류	프롬프트 예시
감성분석 sentiment analysis	다음 텍스트를 긍정적, 중립적, 또는 부정적으로 분류하십시오
텍스트요약 Text summarization	다음 텍스트에 대해 30단어 요약을 작성하십시오.
텍스트 작성 Composing text	다이버 위치에 관한 글을 작성하십시오. 다음 사실들을 포함해야 함: - 롤렉스 오이스터 퍼페추얼은 최초의 방수 시계 중 하나였다. - 오메가는 1948년에 최초의 목적 설계 다이버 워치인 씨마스터(Seamaster)를 제작했다.
질문응답 Question answering	다음 텍스트를 읽고, 1980년대에 인기가 있었던 가정용 컴퓨터들이 무엇인지 말하십시오.
추론 Reasoning	이 수열에서 제곱수들을 더하십시오: 19, 13, 1, 17, 4, 64, 900

3-3. 추론(Reasoning) 프롬프트 좀 더 자세하게

LLM 애플리케이션을 개발할 때, 특히 복잡한 추론이 필요한 경우에는 다양한 프롬프트 기법의 한계를 이해하는 것이 중요하다. 예를 들어, "제로샷 트레이닝(zero-shot training)"은 예시를 전혀 제공하지 않고, 지시만으로 LLM이 작업을 이해하고 수행하길 기대하는 방식이다. 이 방법이 어떤 상황에서는 잘 작동할 수 있지만, 회문(palindrome(회문): 앞에서부터 읽으나 뒤에서부터 읽으나 동일한 문자열이나 수열을 의미한다)을 식별하는 것처럼 더 복잡한 작업에서는 항상 정확한 결과를 내지는 못한다. 다음 섹션에서는 제로샷 트레이닝보다 더 고도화된 프롬프트 기법들을 통해, 추론 작업에서 더 신뢰할 수 있는 성능을 내도록 LLM 프롬프트를 다듬는 방법을 알아볼 것이다

3-3-1. One-shot training

프롬프트에 예시를 하나만 제공하는 것을 "원샷 트레이닝(one-shot training)"이라고 한다. 아래는 회문(palindrome) 문제를 해결하기 위해 하나의 예시를 제공한 방식이다.

Instruction: Sum the palindromes in this sequence: 13, 1331, 121, 73, 99, 56, 232, 7
Example: 33 is a palindrome

위 프롬프트 수행 시 다음과 같은 결과가 생성될 수 있다. (*openAI는 모델을 지속해서 업데이트 하기 때문에 과거에는 틀린 답변이 나왔어도 지금은 올바른 답을 생성할 수 도 있다. 어쨌든 여기서 중요한 포인트는 각 프롬프팅 기법의 한계, 활용 방법, 효과를 등을 확실히 이해하고, 상황에 맞게 적절히 적용하는 능력을 개발하는 것이다.)

I apologize for the previous oversight. Let's correctly identify and sum the palindromes in this sequence: 13, 1331, 121, 73, 99, 56, 232, and 7.

The palindromes in the sequence are 1331, 121, 232, and 7. To find their sum:
 $1331 + 121 + 232 + 7 = 1691$

So, the sum of the palindromes in this sequence is indeed 1691.

안타깝게도, one-shot training은 문제를 해결하지 못했다. 다음 예제에서는 two-shot training을 활용하여 문제를 해결해 보고자 한다.

3-3-2. Two-shot training

다음 블록은 수정된 two-shot prompt의 예시이다. 아래에서는 one-shot의 예와는 달리 2가지 예시(two-shot)를 제시하고 있다.

Instruction: Sum the palindromes in this sequence: 13, 1331, 121, 73, 99, 56 232, 7

Examples:

33 is a palindrome
44 is a palindrome

프롬프트 실행 시 다음과 같은 결과 값을 기대할 수 있다.

Certainly, let's identify and sum the palindromes in the sequence: 13, 1331, 121, 73, 99, 56, 232, and 7.

The palindromes in this sequence are 1331, 121, 232, and 7. To find their sum:
 $1331 + 121 + 232 + 7 = 1691$
So, the sum of the palindromes in this sequence is 1691.

GPT는 여전히 올바른 답변을 생성하지 못한다. 예를 들어 위 답변은 숫자 99를 미포함하고 있다.

다음 단계에서는 이와 같은 문제를 해결하기 위해서 다른 방법을 적용해 보기로 한다. 이번에는 문제를 여러 "단계(steps)"로 나누고 각 단계에 대한 명확한 지시를 제공하는 방법이다. 이는 절차 중 특정 단계에서 어려움을 겪는 동료를 차근차근 안내하는 것과 비슷하다.

3-3-3. Providing Steps prompting

"단계(steps)"를 반영하여 수정된 프롬프트는 다음과 같다.

Instruction: Sum the palindromes in this sequence: 13, 1331, 121, 73, 99, 56 232, 7
Examples:
33 is a palindrome
44 is a palindrome
Steps: 1) identify the palindromes; 2) add them up

프롬프트 실행 시 얻어진 결과는 다음과 같다.

Certainly, let's follow your instructions step by step:
Step 1: Identify the palindromes in the sequence.
Palindromes in the sequence:
- 1331
- 121
- 99
- 232
- 7
Step 2: Add them up.
 $1331 + 121 + 99 + 232 + 7 = 1790$
So, the sum of the palindromes in the sequence is 1790.

마침내 올바른 답변을 찾아내는 것을 볼 수 있다.

3-3-4. Few-shot training

어떤 경우에는 예시 두 개만으로 LLM에게 새로운 개념을 학습시키기에 충분하지 않을 수 있다. 이런 상황에서는 여러 개의 예시를 제공해야 하며, 이를 "퓨샷 트레이닝(few-shot training)"이라고 한다. 이를 설명하기 위해 숫자를 Abra, Kadabra, 또는 Abra Kadabra로 분류하는 게임을 만들어보자. 먼저 분류 기준에 대한 정보 없이 어떤 결과가 나오는지 살펴보자.

Classify the following numbers as Abra, Kadabra or Abra Kadabra:
3, 5, 8, 2, 7, 12, 13

위 프롬프트 실행 시 다음과 같은 결과 값이 나올 것이다.

The classification of numbers into "Abra," "Kadabra," or "Abra Kadabra" doesn't follow a conventional mathematical pattern. These are just words, not related to numbers. Therefore, it's not possible to classify the numbers you provided as "Abra," "Kadabra," or "Abra Kadabra" based on any mathematical or logical criteria.

다음은 few-shot training을 활용한 프롬프트이다. 여기서는 two-shot training과 달리 더 많은 예시를 더 구체적으로 제시하고 있는 것을 볼 수 있다.

Classify the following numbers as Abra, Kadabra or Abra Kadabra:

3, 4, 5, 7, 8, 10, 11, 13, 35

Examples:

6 // not divisible by 5, not divisible by 7 // None

15 // divisible by 5, not divisible by 7 // Abra

12 // not divisible by 5, not divisible by 7 // None

21 // not divisible by 5, divisible by 7 // Kadabra

70 // divisible by 5, divisible by 7 // Abra Kadabra

위 프롬프트 실행 시 다음과 같은 결과 값을 기대할 수 있다.

Here is the output:

Let's classify the numbers as Abra, Kadabra, or Abra Kadabra based on whether they are divisible by 5 and/or 7:

3: Not divisible by 5, not divisible by 7 - None

4: Not divisible by 5, not divisible by 7 - None

5: Divisible by 5, not divisible by 7 - Abra

7: Not divisible by 5, divisible by 7 - Kadabra

8: Not divisible by 5, not divisible by 7 - None

10: Divisible by 5, not divisible by 7 - Abra

11: Not divisible by 5, not divisible by 7 - None

13: Not divisible by 5, not divisible by 7 - None

35: Divisible by 5, divisible by 7 - Abra Kadabra

So, here's the classification:

Abra: 5, 10, 35

Kadabra: 7

Abra Kadabra: 35

None: 3, 4, 8, 11, 13

ChatGPT는 정확한 결과를 제시했을 뿐 아니라, 그에 대한 추론 과정까지 설명하는 것을 볼 수 있다.

3-3-5. 프로그래밍으로 Few-shot training 적용하기 예시

다음 예제는 python에서 FewShotPromptTemplate을 이용하여 Few-shot 프롬프트를 실행하는 예시를 보여준다.

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

```
examples = [
    {
        "number": 6,
        "reasoning": "not divisible by 5 nor by 7",
        "result": "None"
    },
    {
        "number": 15,
        "reasoning": "divisible by 5 but not by 7",
        "result": "Abra"
    },
    {
        "number": 12,
        "reasoning": "not divisible by 5 nor by 7",
        "result": "None"
    },
    {
        "number": 21,
        "reasoning": "divisible by 7 but not by 5",

```



```

    "result": "Kadabra"
  },
  {
    "number": 70,
    "reasoning": "divisible by 5 and by 7",
    "result": "Abra Kadabra"
  }
]

example_prompt = PromptTemplate(input_variables=["number", "reasoning", "result"],
template="{number} \\ {reasoning} \\ {result}")

few_shot_prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    suffix="Classify the following numbers as Abra, Kadabra or Abra Kadabra:
{comma_delimited_input_numbers}",
    input_variables=["comma_delimited_input_numbers"]
)

prompt_input = few_shot_prompt.format(comma_delimited_input_numbers="3, 4, 5, 7, 8, 10,
11, 13, 35.")

response = llm.invoke(prompt_input)

print(response.content)

```

3-3-6. Chain of Thought(CoT) 프롬프트

Chain of Thought이라는 Prompt기법을 명확하게 이해하기 위해 다음과 같은 수학적 정의를 누군가가 정의했다고 생각해보자:

- 숫자들의 수열에 홀수가 두 개 이상 포함되어 있고, 그 홀수들의 합이 3으로 나누어 떨어진다면 그 수열을 "Strange(이상한)"이라고 부를 것이다.

먼저 LLM이 "Strange"라는 방금 정의된 수열의 규칙을 알고 있지 않은지 확인해보도록하자.

Is the following sequence "Strange"?
1, 3, 4, 5, 7, 10, 15, 18, 22, 24

LLM output:

The sequence you provided appears to be increasing, but it doesn't follow a common mathematical pattern like arithmetic or geometric progressions. Whether it's considered "strange" depends on the context or criteria you're using to define "strange." In a mathematical sense, it might be considered an unusual or non-standard sequence. If you have specific criteria or rules that define what a "strange" sequence is, please provide more details for a more precise classification.

예상했던 것처럼 GPT는 Strange의 정의를 이해하지 못한다. 이번 섹션에서는 Chain of Thought 프롬프트 기법을 활용하여 GPT가 Strange라는 새로운 정의를 이해할 수 있도록 유도해보겠다. Chain of Thought은 Wei et al에 의해 쓰여진 (<https://arxiv.org/abs/2201.11903>) 논문에서 잘 설명되어 있으며, 본 논문에서 제시한 Chain of Thought(CoT) 프롬프트의 핵심 아이디어는 다음과 같다:

- Few-shot training을 위한 몇 가지 샘플 수열(예시)
- 각 샘플 수열에 대해, 해당 수열이 Strange인지 판단하기 위한 논리적 단계의 설명

CoT 구체적 예시:

Q: Is the following sequence "Strange"?
3, 4, 5, 7, 10, 18, 22, 24

A:
 Examples:
 Q: is the following sequence strange: 1, 4, 6, 8, 20
 A: 1 is an odd number; I need at least two odd numbers // Not Strange
 Q: is the following sequence strange: 5, 6, 7, 8, 20
 A: 5 and 7 are odd numbers; the sum of 5 and 7 is 12; 12 is divisible by 3 // Strange
 Q: is the following sequence strange: 1, 5, 6, 7, 8, 20
 A: 1, 5 and 7 are odd numbers; the sum of 1, 5 and 7 is 13; 13 is not divisible by 3 // Not Strange
 Q: is the following sequence strange: 5, 6, 7, 8, 9, 20
 A: 5, 7, 9 are odd numbers; the sum of 5, 7 and 9 is 21; 21 is divisible by 3 // Strange

보여지는 바와 같이 CoT를 활용했을 때 GPT가 사용자의 의도를 파악하고 정확한 답변을 생성하는 것을 확인할 수 있다.

Thank you for providing the examples and clarifications. Based on the criteria you've presented:
 The sequence "3, 4, 5, 7, 10, 18, 22, 24" has 3 odd numbers: 3, 5, and 7. The sum of these odd numbers is 15. 15 is divisible by 3. So, according to the criteria, this sequence is **Strange**.

3-3-7. In-Context Training(문맥 내 학습)이란 무엇인가?

“문맥 내 학습(In-Context Learning)”은 사용자로부터 받은 익숙하지 않은 작업이나 지시에 대해, 프롬프트 안에 포함된 예시들을 활용하여 LLM에게 학습시키는 모든 방식을 의미한다. 일반적인 기법으로는 One-shot, two-shot, few-shot 트레이닝이나, 단계별 안내 방식(일명 “Chain of Thought”)이 있다. 문맥 내 학습은 Fine-Tuning과 비교했을 때, GPU와 같은 고급 하드웨어나 Transformer 구조에 대한 깊은 지식 없이도 적용이 가능하기 때문에 비용 효율적이고 자원 소모가 적은 접근 방식이다.

4. 프롬프트의 구조

앞서 살펴본 모든 프롬프트 요소들을 결합하면 다음과 같은 일반화된 구조가 만들어진다:

① **Persona**: 언어 모델(LLM)이 어떤 역할을 수행하길 원하는지 지정한다.

예: 능숙한 마케팅 매니저, 수상 경력 있는 SF 작가, 컴퓨터 잡지의 유능한 교정자 등. 기대 수준을 명확히 설정할수록 좋다.

② **맥락 (Context)**: 요청과 관련된 배경 정보를 상세히 제공한다. 정확한 정보는 LLM이 사용자의 기대에 더 잘 부합하는 응답을 생성할 가능성을 높여준다.

③ **지시 (Instruction)**: LLM에게 원하는 작업을 명확하게 지시한다. 번역, 작성, 재작성, 향상, 분류, 정렬, 찾기 등 원하는 작업을 구체적으로 서술해야 한다.

④ **입력 (Input)**: 입력 내용 자체를 말하며, 문단, 질문, 핵심 요약 리스트 등 다양한 형태가 될 수 있다. "Text", "Question" 등으로 라벨링하면 이해를 도울 수 있다.

⑤ **처리 단계 (Steps)**: LLM이 출력을 생성할 때 따라야 할 단계나 절차를 설명한다.

⑥ **톤 (Tone)**: 응답의 어조를 지정한다.

예: 격식 있는(formal), 캐주얼한(informal), 위트 있는(witty), 열정적인(enthusiastic), 차분한(sober), 친근한(friendly) 등. 조합도 가능하다.

⑦ **출력 형식 (Output Format)**: 필요 시 원하는 출력 형식을 지정한다. 지정하지 않으면 기본적으로 텍스트로 응답하지만, 표(table), CSV (컬럼명 포함), JSON (속성명 포함) 등 구조화된 형식도 요청 가능하다.

⑧ **예시 (Examples)**: LLM이 특정 작업에 대해 학습이 부족할 수 있는 경우, 특히 텍스트 분류나 추론 작업에서는 몇 개의 인라인 예시를 통해 미니 트레이닝을 제공할 수 있다. 단 몇 개의 예시만으로도 응답 품질이 크게 향상될 수 있다.

5. Prompt Engineering 전체 요약

- 프롬프트는 LLM에게 지시를 내리는 구체적인 요청으로, 주어진 배경 정보를 바탕으로 모델의 응답을 유도하는 역할을 한다.
- 프롬프트는 작업의 성격에 따라 텍스트 분류, 감정 분석, 텍스트 요약, 생성, 질의응답 등 특정 목적에 맞게 설계된다.
- 프롬프트가 기대한 결과를 내지 못할 경우, 이는 LLM이 해당 작업에 익숙하지 않다는 의미이며, 학습이 필요하다는 신호다.
- 이때 학습은 예시를 추가함으로써 이루어지며, 예시 개수에 따라 원샷(one-shot), 투샷(two-shot), 퓨샷(few-shot) 트레이닝으로 분류된다.
- 다양한 프롬프트를 만들다 보면 보통 공통된 구조가 보이는데, Persona, 컨텍스트, 지시문, 입력, 단계, 예시 등의 요소로 구성된다.

- 사용자는 본인의 목적에 맞게 이 구조를 선택적으로 사용하거나 필요한 항목을 추가/제외하며 맞춤화할 수 있다. 명시적인 섹션 이름을 생략하는 것도 가능하다.
- 일반 사용자들은 ChatGPT와 같은 대화형 LLM 인터페이스를 통해 프롬프트를 직접 작성하고 실행하지만, 소프트웨어 개발자들은 보통 LLM API나 LangChain을 이용해 코드로 프롬프트를 자동 생성하고 실행한다.
- LangChain은 PromptTemplate이라는 기본 기능부터 보다 고급 기능인 FewShotPromptTemplate까지 다양한 프롬프트 엔지니어링 도구를 제공한다.
- LLM 기반 애플리케이션을 개발할 때는 파라미터화된 프롬프트, 즉 프롬프트 템플릿을 설계하는 것이 매우 중요하다.