프롬프트 엔지니어링 리포트

1. 개요

프롬프트 엔지니어링은 언어 모델이 제공하는 응답의 품질을 높이기 위해 프롬프트(질문 또는 명령어)를 설계하는 기법을 말합니다. 특히 대규모 언어 모델에서 프롬프트의 설계는 모델의 성능에 큰 영향을 미칩니다. 본 리포트에서는 프롬프트 엔지니어링의 주요 기법인 Few shot prompting, Chain of Thought(COT) 기법, Self-Consistency(SC) 기법, Tree of Thought(ToT) 기법, ReAct 기법, Retrieval-Augmented Generation(RAG) 기법에 대해 설명하고, 각 기법의 예제를 Azure OpenAI와 파이썬을 사용하여 제시합니다.

2. Few Shot Prompting

Few Shot Prompting은 가장 기초적인 프롬프트 엔지니어링 기법입니다. 어떤 질문이 학습되지 않은 내용에 관한 것이라도, 그 질문과 관련된 예시를 적절히 제공하면 언어 모델이 그 예시의 의미를 추론하여 답하는 원리입니다. 샘플의 수에 따라서 아무 것도 없는 경우는 zero shot, 하나의 예시만 있는 경우는 one shot, 몇 개의 예시를 제공하는 것을 few shot이라고 합니다.

예제

from openai import AzureOpenAI

client = AzureOpenAI(

api\_key=str\_api\_key, # Azure OpenAI Key

api\_version=str\_api\_version, # Azure OpenAI API version

azure\_endpoint=str\_endpoint # Azure OpenAI endpoint

)

prompt = """

Q: The capital of France is Paris. What is the capital of Germany?

A: Berlin

Q: The capital of Spain is Madrid. What is the capital of Italy?

A: Rome

Q: The capital of Portugal is Lisbon. What is the capital of Japan?

A:

"""

context = [{"role": "user", "content": prompt}]

response = client.chat.completions.create(

model="gpt-4o",

messages=context,

temperature=0,

top\_p=0

)

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

답변

Tokyo

3. Chain of Thought (CoT) 기법

Chain of Thought(COT) 기법은 모델이 복잡한 문제를 단계별로 해결하도록 유도하는 프롬프트 기법입니다. 각 단계별로 문제를 세분화하여 해결함으로써 최종적으로 정확한 답변에 도달할 수 있도록 합니다.

예제

prompt = """

Q: If there are 3 apples and you take away 2, how many do you have?

A: Let's break it down step by step. There are 3 apples initially. If you take away 2 apples, you are left with 1 apple.

"""

context = [{"role": "user", "content": prompt}]

response = client.chat.completions.create(

model="gpt-4o",

messages=context,

temperature=0,

top\_p=0

)

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

답변

Actually, the question is asking how many apples you have after taking 2 away. If you take away 2 apples, you have those 2 apples with you. So, you have 2 apples.

4. Self-Consistency (SC) 기법

Self-Consistency(SC) 기법은 동일한 질문에 대해 여러 번 응답을 생성하고, 그 응답들 중 가장 일관된 응답을 선택하는 방식입니다. 이를 통해 더 높은 품질의 응답을 얻을 수 있습니다.

예제

from openai import AzureOpenAI

from collections import defaultdict

import re

client = AzureOpenAI(

api\_key = str\_api\_key, # Azure Open AI Key

api\_version = str\_api\_version, # Azure OpenAI API model

azure\_endpoint = str\_endpoint # Azure Open AI endpoint

)

template = "What is the square root of 144?"

def get\_most\_frequent\_answer(template, iterations=10):

answers = defaultdict(int) # Exists if accessed with nonexistent key

for idx in range(iterations):

context = [{"role": "user", "content": template}]

response = client.chat.completions.create(

model="gpt-35-turbo-16k",

messages=context,

temperature=0.3

)

response\_content = response.choices[0].message.content

print(f"\nSample {idx+1}:")

print(response\_content)

# Extract answer

match = re.search(r"square root of 144 is (\d+)", response\_content)

if match:

parsed\_answer = match.group(1)

answers[parsed\_answer] += 1

# Select the most frequent answer

sorted\_answers = sorted(answers.items(), key=lambda x: x[1], reverse=True)

print(f"\nFrequency table: {sorted\_answers}")

# Check if sorted\_answers is empty before accessing it

if sorted\_answers:

most\_frequent\_answer = sorted\_answers[0][0]

else:

most\_frequent\_answer = "No valid answers found"

return most\_frequent\_answer

most\_frequent\_answer = get\_most\_frequent\_answer(template)

print("Most frequent answer:", most\_frequent\_answer)  
답변  
Sample 1:

The square root of 144 is 12.

Sample 2:

The square root of 144 is 12.

Sample 3:

The square root of 144 is 12.

Sample 4:

The square root of 144 is 12.

Sample 5:

The square root of 144 is 12.

Sample 6:

The square root of 144 is 12.

Sample 7:

The square root of 144 is 12.

Sample 8:

The square root of 144 is 12.

Sample 9:

The square root of 144 is 12.

Sample 10:

The square root of 144 is 12.

Frequency table: [('12', 10)]

Most frequent answer: 12

5. Tree of Thought (ToT) 기법

Tree of Thought(ToT) 기법은 문제를 여러 경로로 해결해보는 기법입니다. 각 경로는 문제를 해결하는 다른 방법을 나타내며, 가장 최적의 경로를 선택하여 문제를 해결합니다.

예제

prompt = """

Q: How can you get from Paris to Berlin?

A: There are several ways. Let's explore them:

1. You can fly directly from Paris to Berlin.

2. You can take a train from Paris to Berlin.

3. You can drive from Paris to Berlin.

4. You can take a bus from Paris to Berlin.

The most efficient way is to fly directly.

"""

context = [{"role": "user", "content": prompt}]

response = client.chat.completions.create(

model="gpt-4o",

messages=context,

temperature=0,

top\_p=0

)

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

답변

Yes, you're right! Here are some more details on each option:

1. \*\*Flying\*\*: The fastest way to travel from Paris to Berlin is by plane. Several airlines offer direct flights between Paris Charles de Gaulle Airport (CDG) or Orly Airport (ORY) and Berlin Brandenburg Airport (BER). The flight duration is approximately 1.5 to 2 hours.

2. \*\*Train\*\*: You can take a high-speed train from Paris to Berlin. The journey typically involves a transfer, often in cities like Frankfurt or Cologne. The total travel time is around 8 to 10 hours, depending on the connection.

3. \*\*Driving\*\*: If you prefer to drive, the distance between Paris and Berlin is about 1,050 kilometers (650 miles). The drive can take around 10 to 12 hours, depending on traffic and road conditions. You would travel through several countries, including France, Belgium, and Germany.

4. \*\*Bus\*\*: Long-distance buses are another option. Companies like FlixBus offer routes between Paris and Berlin. The bus journey can take anywhere from 12 to 15 hours, depending on the specific route and stops.

Each mode of transportation has its own advantages, so the best choice depends on your preferences for speed, comfort, and cost.

6. ReAct 기법

ReAct 기법은 모델이 행동(Act)과 추론(Reasoning)을 결합하여 문제를 해결하는 방식입니다. 모델은 먼저 행동을 취하고, 그 다음 그 행동에 대한 이유를 설명합니다.

예제

from openai import AzureOpenAI

client = AzureOpenAI(

api\_key = str\_api\_key, #Azure Open AI Key

api\_version = str\_api\_version, #Azue OpenAI API model

azure\_endpoint = str\_endpoint #Azure Open AI end point(매직에꼴)

)

template = """

Thought, Action, Observation 단계를 번갈아 가며 질문에 답해가는 과정을 통해 <과제/>를 해결합니다.

1. Thought: 현재 상황에 대한 추론

2. Action:

- Search[keyword]: <도구상자/>에서 도구 하나를 꺼내서 keyword 검색

- Finish: {"해결책": <해결책을 단답형으로 제시하고 작업을 완료>}

3. Observation: 도구를 사용한 결과를 객관적으로 관찰

```

<도구상자>

- 온도 검색[도시명] : {"서울":20.1, "자카르타":32.1, "헬싱키" -1},

- 입을 옷 검색[더운 날씨, 선선한 날씨, 추운 날씨 등]: {"더운 날씨":"반팔 티셔츠", "선선한 날씨": "긴팔 티셔츠", "추운 날씨": "패딩"}

</도구상자>

```

- 매회차별 1-SET, 2-SET, ... ,N-SET로 표기할 것.

- 매회차별 필요한 부분을 나누어 생각(Thought)할 것.

- Action을 출력할 때는 도구명과 keyword를 표기할 것.

```

<과제>

내일 자카르타로 떠날 예정입니다. 어떤 옷을 챙겨가면 될까요?

</과제>

"""

context = [{"role": "user", "content": template}]

response = client.chat.completions.create(

model="gpt-4o",

messages=context,

temperature=0,

top\_p=0

)

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

답변

1-SET

1. Thought: 자카르타의 현재 온도를 알아야 어떤 옷을 챙겨야 할지 결정할 수 있습니다.

2. Action: 온도 검색[자카르타]

3. Observation: {"자카르타":32.1}

2-SET

1. Thought: 자카르타의 온도가 32.1도인 것을 확인했습니다. 이는 더운 날씨에 해당합니다. 더운 날씨에 적합한 옷을 검색해보겠습니다.

2. Action: 입을 옷 검색[더운 날씨]

3. Observation: {"더운 날씨":"반팔 티셔츠"}

3-SET

1. Thought: 자카르타의 날씨가 더워서 반팔 티셔츠를 챙기면 될 것 같습니다.

2. Action: Finish: {"해결책": "반팔 티셔츠"}

7. Retrieval-Augmented Generation (RAG) 기법

Retrieval-Augmented Generation(RAG) 기법은 모델이 외부 지식 베이스를 검색하여 관련 정보를 가져오고, 그 정보를 바탕으로 응답을 생성하는 방식입니다. 이를 통해 모델은 더 정확하고 신뢰할 수 있는 정보를 제공할 수 있습니다.

예제

system\_role = """

신조어 사전:

{"자만추": "자연스러운 만남 추구"

"좋댓구알": "좋아요, 댓글, 구독 알림설정의 줄임말",

"어쩔티비": "어쩌라고 안 물어봤는데 를 뜻하는 말"

"가심비": "가격 대비 심리적 만족도가 주는 효용"

"갓생": "일상의 소소한 성취감을 추구하는 삶"}

신조어 사전에서 답하세요. 신조어 사전에 없다면 "모르는 단어입니다"라고 답하세요."""

template = """

신조어 {신조어}의 뜻을 말해주세요.

""".format(신조어="자만추")

context = [{"role": "system", "content": system\_role},

{"role": "user", "content": template}]

response = client.chat.completions.create(

model="gpt-4o",

messages=context,

temperature=0,

top\_p=0

)

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

답변

"자만추"는 "자연스러운 만남 추구"를 의미합니다.

8. 결론

프롬프트 엔지니어링은 언어 모델의 응답 품질을 향상시키기 위한 중요한 기법입니다. Few Shot Prompting, Chain of Thought, Self-Consistency, Tree of Thought, ReAct, Retrieval-Augmented Generation 등의 기법은 각기 다른 방식으로 모델의 성능을 극대화합니다. 본 리포트에서는 이러한 기법들에 대한 개념과 Azure OpenAI 및 파이썬을 활용한 예제를 제시하였습니다.