Intro to Prompt Engineering: Tips and Tricks

Introduction

Prompt engineering is a relatively new discipline that involves developing and optimizing prompts to use language models for various applications and research topics efficiently. It helps to understand the capabilities and limitations of LLMs better and is essential for many NLP tasks. We will provide practical examples to demonstrate the difference between good and bad prompts, helping you to understand the nuances of prompt engineering better.

By the end of this lesson, you will have a solid foundation in the knowledge and strategies needed to create powerful prompts that enable LLMs to deliver accurate, contextually relevant, and insightful responses.

Role Prompting

Role prompting involves asking the LLM to assume a specific role or identity before performing a given task, such as acting as a copywriter. This can help guide the model's response by providing a context or perspective for the task. To work with role prompts, you could iteratively:

1. Specify the role in your prompt, e.g., "As a copywriter, create some attention-grabbing taglines for AWS services."
2. Use the prompt to generate an output from an LLM.
3. Analyze the generated response and, if necessary, refine the prompt for better results.

Examples:

In this example, the LLM is asked to act as a futuristic robot band conductor and suggest a song title related to the given theme and year. (A reminder to set your OpenAI API key in your environment variables using the `OPENAI_API_KEY` key)

Remember to install the required packages with the following command: `pip install langchain==0.0.208 deeplake openai tiktoken`.

```
from langchain import PromptTemplate, LLMChain
from langchain.llms import OpenAI


# Before executing the following code, make sure to have
# your OpenAI key saved in the “OPENAI_API_KEY” environment variable.
# Initialize LLM
llm = OpenAI(model_name="text-davinci-003", temperature=0)


template = """
```

```
As a futuristic robot band conductor, I need you to help me come up with a song
title.

What's a cool song title for a song about {theme} in the year {year}?

"""

prompt = PromptTemplate(

    input_variables=["theme", "year"],

    template=template,

)


# Create the LLMChain for the prompt

llm = OpenAI(model_name="text-davinci-003", temperature=0)


# Input data for the prompt

input_data = {"theme": "interstellar travel", "year": "3030"}


# Create LLMChain

chain = LLMChain(llm=llm, prompt=prompt)


# Run the LLMChain to get the AI-generated song title

response = chain.run(input_data)


print("Theme: interstellar travel")

print("Year: 3030")

print("AI-generated song title:", response)
```

The sample code.

```
Theme: interstellar travel

Year: 3030

AI-generated song title:

"Journey to the Stars: 3030"
```

The output.
This is a good prompt for several reasons:

- **Clear instructions:** The prompt is phrased as a clear request for help in generating a song title, and it specifies the context: "As a futuristic robot band conductor." This helps the LLM understand that the desired output should be a song title related to a futuristic scenario.

- **Specificity:** The prompt asks for a song title that relates to a specific theme and a specific year, "{theme} in the year {year}." This provides enough context for the LLM to generate a relevant and creative output. The prompt can be adapted for different themes and years by using input variables, making it versatile and reusable.
- **Open-ended creativity:** The prompt allows for open-ended creativity, as it doesn't limit the LLM to a particular format or style for the song title. The LLM can generate a diverse range of song titles based on the given theme and year.
- **Focus on the task:** The prompt is focused solely on generating a song title, making it easier for the LLM to provide a suitable output without getting sidetracked by unrelated topics.

These elements help the LLM understand the user's intention and generate a suitable response.

Few Shot Prompting

Few Shot Prompting In the next example, the LLM is asked to provide the emotion associated with a given color based on a few examples of color-emotion pairs:

```
from langchain import PromptTemplate, FewShotPromptTemplate, LLMChain
from langchain.llms import OpenAI


# Initialize LLM
llm = OpenAI(model_name="text-davinci-003", temperature=0)


examples = [
    {"color": "red", "emotion": "passion"},
    {"color": "blue", "emotion": "serenity"},
    {"color": "green", "emotion": "tranquility"},
]


example_formatter_template = """
Color: {color}
Emotion: {emotion}\n
"""
example_prompt = PromptTemplate(
    input_variables=["color", "emotion"],
    template=example_formatter_template,
)
```

```
few_shot_prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    prefix="Here are some examples of colors and the emotions associated with them:\n\n",
    suffix="\n\nNow, given a new color, identify the emotion associated with it:\n\nColor: {input}\nEmotion:",
    input_variables=["input"],
    example_separator="\n",
)


formatted_prompt = few_shot_prompt.format(input="purple")


# Create the LLMChain for the prompt
chain = LLMChain(llm=llm, prompt=PromptTemplate(template=formatted_prompt, input_variables=[]))


# Run the LLMChain to get the AI-generated emotion associated with the input color
response = chain.run({})


print("Color: purple")
print("Emotion:", response)
```

The sample code.

```
Color: purple
Emotion:  creativity
```

The output.
This prompt provides **clear instructions** and few-shot examples to help the model understand the task.

Bad Prompt Practices

Now, let's see some examples of prompting that are generally considered bad.
Here's an example of a too-vague prompt that **provides little context** or guidance for the model to generate a meaningful response.

```
from langchain import PromptTemplate


template = "Tell me something about {topic}."
```

```
prompt = PromptTemplate(
    input_variables=["topic"],
    template=template,
)
prompt.format(topic="dogs")
```

The sample code.

```
'Tell me something about dogs.’
```

The output.

Chain Prompting

Chain Prompting refers to the practice of chaining consecutive prompts, where the output of a previous prompt becomes the input of the successive prompt.
To use chain prompting with LangChain, you could:

- Extract relevant information from the generated response.
- Use the extracted information to create a new prompt that builds upon the previous response.
- Repeat steps as needed until the desired output is achieved.

`PromptTemplate` class makes constructing prompts with dynamic inputs easier. This is useful when creating a prompt chain that depends on previous answers.

```
from langchain import PromptTemplate, LLMChain
from langchain.llms import OpenAI


# Initialize LLM
llm = OpenAI(model_name="text-davinci-003", temperature=0)


# Prompt 1
template_question = """What is the name of the famous scientist who developed the theory of general relativity?
Answer: """
prompt_question = PromptTemplate(template=template_question, input_variables=[])


# Prompt 2
template_fact = """Provide a brief description of {scientist}'s theory of general relativity.
Answer: """
prompt_fact = PromptTemplate(input_variables=["scientist"], template=template_fact)
```

```
# Create the LLMChain for the first prompt
chain_question = LLMChain(llm=llm, prompt=prompt_question)

# Run the LLMChain for the first prompt with an empty dictionary
response_question = chain_question.run({})

# Extract the scientist's name from the response
scientist = response_question.strip()

# Create the LLMChain for the second prompt
chain_fact = LLMChain(llm=llm, prompt=prompt_fact)

# Input data for the second prompt
input_data = {"scientist": scientist}

# Run the LLMChain for the second prompt
response_fact = chain_fact.run(input_data)

print("Scientist:", scientist)
print("Fact:", response_fact)
```

The sample code.

```
Scientist: Albert Einstein
Fact:
Albert Einstein's theory of general relativity is a theory of gravitation that states
that the gravitational force between two objects is a result of the curvature of
spacetime caused by the presence of mass and energy. It explains the phenomenon of
gravity as a result of the warping of space and time by matter and energy.
```

The output.
This prompt may generate a less informative or focused response than the previous example due to its more open-ended nature.

Bad Prompt Example:

```
from langchain import PromptTemplate, LLMChain
from langchain.llms import OpenAI

# Initialize LLM
```

```
llm = OpenAI(model_name="text-davinci-003", temperature=0)


# Prompt 1
template_question = """What is the name of the famous scientist who developed the
theory of general relativity?
Answer: """
prompt_question = PromptTemplate(template=template_question, input_variables=[])


# Prompt 2
template_fact = """Tell me something interesting about {scientist}.
Answer: """
prompt_fact = PromptTemplate(input_variables=["scientist"], template=template_fact)


# Create the LLMChain for the first prompt
chain_question = LLMChain(llm=llm, prompt=prompt_question)


# Run the LLMChain for the first prompt with an empty dictionary
response_question = chain_question.run({})


# Extract the scientist's name from the response
scientist = response_question.strip()


# Create the LLMChain for the second prompt
chain_fact = LLMChain(llm=llm, prompt=prompt_fact)


# Input data for the second prompt
input_data = {"scientist": scientist}


# Run the LLMChain for the second prompt
response_fact = chain_fact.run(input_data)


print("Scientist:", scientist)
print("Fact:", response_fact)
```

The sample code.

```
Scientist: Albert Einstein

Fact:  Albert Einstein was a vegetarian and an advocate for animal rights. He was
also a pacifist and a socialist, and he was a strong supporter of the civil rights
movement. He was also a passionate violinist and a lover of sailing.
```

The output.
This prompt may generate a less informative or focused response than the previous example due to its more open-ended nature.

An example of the unclear prompt:

```
from langchain import PromptTemplate, LLMChain

from langchain.llms import OpenAI


# Initialize LLM

llm = OpenAI(model_name="text-davinci-003", temperature=0)


# Prompt 1

template_question = """What are some musical genres?

Answer: """

prompt_question = PromptTemplate(template=template_question, input_variables=[])


# Prompt 2

template_fact = """Tell me something about {genre1}, {genre2}, and {genre3} without
giving any specific details.

Answer: """

prompt_fact = PromptTemplate(input_variables=["genre1", "genre2", "genre3"],
template=template_fact)


# Create the LLMChain for the first prompt

chain_question = LLMChain(llm=llm, prompt=prompt_question)


# Run the LLMChain for the first prompt with an empty dictionary

response_question = chain_question.run({})


# Assign three hardcoded genres

genre1, genre2, genre3 = "jazz", "pop", "rock"


# Create the LLMChain for the second prompt
```

```
chain_fact = LLMChain(llm=llm, prompt=prompt_fact)


# Input data for the second prompt
input_data = {"genre1": genre1, "genre2": genre2, "genre3": genre3}


# Run the LLMChain for the second prompt
response_fact = chain_fact.run(input_data)


print("Genres:", genre1, genre2, genre3)
print("Fact:", response_fact)
```

The sample code.

```
Genres: jazz pop rock
Fact:
Jazz, pop, and rock are all genres of popular music that have been around for
decades. They all have distinct sounds and styles, and have influenced each other in
various ways. Jazz is often characterized by improvisation, complex harmonies, and
syncopated rhythms. Pop music is typically more accessible and often features catchy
melodies and hooks. Rock music is often characterized by distorted guitars, heavy
drums, and powerful vocals.
```

The output.

In this example, the second prompt is constructed poorly. It asks to "tell me something about {genre1}, {genre2}, and {genre3} without giving any specific details." This prompt is unclear, as it asks for information about the genres but also states not to provide specific details. This makes it difficult for the LLM to generate a coherent and informative response. As a result, the LLM may provide a less informative or confusing answer.

The first prompt asks for "some musical genres" **without specifying any criteria or context**, and the second prompt asks why the given genres are "unique" **without providing any guidance** on what aspects of uniqueness to focus on, such as their historical origins, stylistic features, or cultural significance.

Chain of Thought Prompting

Chain of Thought Prompting (CoT) is a technique developed to encourage large language models to explain their reasoning process, leading to more accurate results. By providing few-shot exemplars demonstrating the reasoning process, the LLM is guided to explain its reasoning when answering the prompt. This approach has been found effective in improving results on tasks like arithmetic, common sense, and symbolic reasoning.

In the context of LangChain, CoT can be beneficial for several reasons. First, it can help break down complex tasks by assisting the LLM in decomposing a complex task into simpler steps, making it easier to understand and solve the problem. This is particularly

useful for calculations, logic, or multi-step reasoning tasks. Second, CoT can guide the model through related prompts, helping generate more coherent and contextually relevant outputs. This can lead to more accurate and useful responses in tasks that require a deep understanding of the problem or domain.
There are some limitations to consider when using CoT. One limitation is that it has been found to yield performance gains only when used with models of approximately 100 billion parameters or larger; smaller models tend to produce illogical chains of thought, which can lead to worse accuracy than standard prompting. Another limitation is that CoT may not be equally effective for all tasks. It has been shown to be most effective for tasks involving arithmetic, common sense, and symbolic reasoning. For other types of tasks, the benefits of using CoT might be less pronounced or even counterproductive.

Tips for Effective Prompt Engineering

- **Be specific** with your prompt: Provide enough context and detail to guide the LLM toward the desired output.
- **Force conciseness** when needed.
- **Encourage the model to explain its reasoning**: This can lead to more accurate results, especially for complex tasks.

Keep in mind that prompt engineering is an iterative process, and it may require several refinements to obtain the best possible answer. As LLMs become more integrated into products and services, the ability to create effective prompts will be an important skill to have.
A well-structured prompt example:

```
from langchain import FewShotPromptTemplate, PromptTemplate, LLMChain
from langchain.llms import OpenAI


# Initialize LLM
llm = OpenAI(model_name="text-davinci-003", temperature=0)


examples = [
    {
        "query": "What's the secret to happiness?",
        "answer": "Finding balance in life and learning to enjoy the small moments."
    }, {
        "query": "How can I become more productive?",
        "answer": "Try prioritizing tasks, setting goals, and maintaining a healthy work-life balance."
    }
]
```

```
example_template = """
User: {query}
AI: {answer}
"""

example_prompt = PromptTemplate(
    input_variables=["query", "answer"],
    template=example_template
)

prefix = """The following are excerpts from conversations with an AI
life coach. The assistant provides insightful and practical advice to the users'
questions. Here are some
examples:
"""

suffix = """
User: {query}
AI: """

few_shot_prompt_template = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    prefix=prefix,
    suffix=suffix,
    input_variables=["query"],
    example_separator="\n\n"
)

# Create the LLMChain for the few-shot prompt template
chain = LLMChain(llm=llm, prompt=few_shot_prompt_template)

# Define the user query
```

```
user_query = "What are some tips for improving communication skills?"


# Run the LLMChain for the user query
response = chain.run({"query": user_query})


print("User Query:", user_query)
print("AI Response:", response)
```

The sample code.

```
User Query: What are some tips for improving communication skills?
AI Response:  Practice active listening, be mindful of your body language, and be
open to constructive feedback.
```

The output.
This prompt:

- **Provides a clear context in the prefix**: The prompt states that the AI is a life coach providing insightful and practical advice. This context helps guide the AI's responses and ensures they align with the intended purpose.
- **Uses examples** that demonstrate the AI's role and **the type of responses** it generates: By providing relevant examples, the AI can better understand the style and tone of the responses it should produce. These examples serve as a reference for the AI to generate similar responses that are consistent with the given context.
- Separates examples and the actual query: This **allows the AI to understand the format it should follow**, ensuring a clear distinction between example conversations and the user's input. This separation helps the AI to focus on the current query and respond accordingly.
- Includes a clear suffix that indicates where the user's input goes and where the AI should provide its response: The suffix acts as a cue for the AI, showing where the user's query ends and the AI's response should begin. This structure helps maintain **a clear and consistent format** for the generated responses.

By using this well-structured prompt, the AI can understand its role, the context, and the expected response format, leading to more accurate and useful outputs.

Conclusion

This lesson explored various techniques for creating more effective prompts for large language models. By understanding and applying these tips and tricks, you'll be better equipped to craft powerful prompts that enable LLMs to deliver accurate, contextually relevant, and insightful responses. Always remember that prompt engineering is an iterative process that may require refinement to obtain the best possible results.
In conclusion, prompt engineering is a powerful technique that can help to optimize language models for various applications and research topics. By creating good prompts, we can guide the model to deliver accurate, contextually relevant, and insightful responses. Role prompting and chain prompting are two techniques that can

be used to create good prompts, and we have provided practical examples of each. On the other hand, we have also demonstrated bad prompt examples that don't provide enough context or guidance for the model to generate a meaningful response. By following the tips and tricks presented in this post, you can develop a solid foundation in prompt engineering and use language models for various tasks more effectively.