Prompt Engineering with OpenAI

Prompt engineering is the process of **designing and optimizing prompts** to better utilize LLMs. Providing relevant, specific, unambiguous, and well-structured prompts can help the model better understand the context and generate more accurate responses.

Here are the few pointers for prompt engineering.

- **Setting context**: You can give a brief introduction, a summary, or a definition of the topic before asking the model to generate content.
- **Adjusting model parameters**: You can increase the temperature to make the model more adventurous.
- **Selecting the right prompt**: Choose a prompt that is **clear, specific, and informative**. The prompt should communicate the desired **output format, length, style, and tone**.
- **Evaluating and iterating**: Test the model's output on **different prompts and parameters and compare the results**. Modify the prompt or the parameters to improve the output and repeat the process until you are satisfied.
- **Using natural language:** Avoid using jargon, slang, abbreviations, or ambiguous terms that might confuse the model or lead to unwanted outputs. For example, instead of saying "Write a haiku about AI", you can say "Write a short poem with three lines, where the first line has five syllables, the second line has seven syllables, and the third line has five syllables. The poem should be about artificial intelligence."
- **Providing feedback:** Give the model some feedback on its output, either positive or negative, to help it learn from its mistakes and improve its performance. You can also use reinforcement learning techniques, such as rewards or penalties, to shape the model's behaviour. For example, you can say "**Good job, this is a valid summary**. However, you missed one important point: …" or "**This is not a good summary**, because it is too long and contains irrelevant details. Please try again and focus on the main points."

Techniques to Optimize the Prompt:

Use section markers

A specific technique for formatting instructions is to split the instructions at the beginning or end of the prompt and have the user content contained within --- or ### blocks. These tags allow the model to more clearly differentiate between instructions and content. For example:

```
Translate the text into French
```

What's the weather going to be like today?

Primary, supporting, and grounding content

a) Primary content refers to content that is the **subject of the query**, such a sentence to translate or an article to summarize. This content is often included at the beginning or end of the prompt (as an instruction and differentiated by --- blocks), with instructions explaining what to do with it.

For example, say we have a long article that we want to summarize. We could put it in a --- block in the prompt, then end with the instruction.

<insert full article here, as primary content>

Summarize this article and identify three takeaways in a bulleted list

b) Supporting content is content that may alter the response, but isn't the focus or subject of the prompt. Examples of supporting content include things like names, preferences, future date to include in the response, and so on. Providing supporting content allows the model to respond more completely, accurately, and be more likely to include the desired information.

For example, given a very long promotional email, the model is able to extract key information. If you then add supporting content to the prompt specifying something specific you're looking for, the model can provide a more useful response. In this case the email is the primary content, with the specifics of what you're interested in as the supporting content

---

<insert full email here, as primary content>

<the next line is the supporting content>
Topics I'm very interested in: AI, webinar dates, submission deadlines

Extract the key points from the above email, and put them in a bulleted list:

c) Grounding content allows the model to provide reliable answers by providing content for the model to draw answer from. Grounding content could be an **essay** or **article** that you then ask questions about, a company **FAQ** document, or information that is **more recent** than the data the model was trained on. If you need more reliable and current responses, or you need to reference unpublished or specific information, grounding content is highly recommended.

Grounding content differs from primary content as it's the source of information to answer the prompt query, instead of the content being operated on for things like summarization or translation. For example, when provided an unpublished research paper on the history of AI, it can then answer questions using that grounding content.

```
---
<insert unpublished paper on the history of AI here, as grounding content>
---
Where and when did the field of AI start?
```

This grounding data allows the model to give more accurate and informed answers that may not be part of the dataset it was trained on.

Cues

Cues are **leading words** for the model to build upon, and often help shape the response in the right direction. They often are used with instructions, but don't always. Cues are particularly helpful if prompting the model for code generation. Current Azure OpenAI models can generate some interesting code snippets, however code generation will be covered in more depth in a future module.

Example1, if you're wanting help creating a SQL query, provide instructions of what you need along with the beginning of the query:

```
Write a join query to get customer names with purchases in the past 30 days between tables named orders and customer on customer ID.
SELECT
```

Example2: Help with python code, by giving code comments about the desired app and including import as a leading word at the end of the prompt, or similar in your desired language.

Example3: Given a large collection of customer reviews in a prompt, and ending with:

```
[
 {
   "Product Reviewed by": "Manas",
   "Rating": "1.0 out of 5 stars",
   "Review": "Only 5 months use my earphone not working and useless"
 },
 {
   "Product Reviewed by": "Venkataramanan B.V",
   "Rating": "1.0 out of 5 stars",
   "Review": "In just 5 months, nothing is heard in both the ear buds"
 },
 {
   "Product Reviewed by": "Very good. It's good for big breed dogs till the age of 6 months or 7 then they need a more bigger one than this.",
   "Rating": "5.0 out of 5 stars",
   "Review": "It's really nice. I ordered it almost 1 year back and it's still good and working well"
 },
 {
   "Product Reviewed by": "Placeholder",
   "Rating": "5.0 out of 5 stars",
   "Review": "एक ईमानदार रिव्यू ये product शानदार हे मजबूत हे बैलेंस है में उसको यूज कर रहा हू साउंड क्वालिटी बड़ी अच्छी हे वैल्यू फॉर मनी शोक से लीजिए\n\nएक ईमानदार रिव्यू।। नाइस बैलेंस साउन्ड ,मजबूत वायर ,\n\nमें उसे यूज कर रहा हू शोक से लीजिए वैल्यू फॉर मनी"
 },
]

Most common complaints:


Non-english reviews summary:
-
```

The model then knows to complete the statements based off the context provided in the reviews.

Provide Context

By providing context to the AI model, it allows the model to better understand what you are asking for or what it should know to provide the best answer. Context can be provided in several ways.

a) Request output composition

Specifying the structure of your output can have a large impact on your results.

This could include something like asking the model to

i) cite their sources

ii) write the response as an email,

iii) format the response as a SQL query

iv) classify sentiment into a specific structure,

v) and so on.

vi)

For example:

Write a table in **markdown** with 6 animals in it, with their genus and species

Response

```
| Animal          | Genus          | Species        |
| ---------------- |:--------------:|:--------------:|
| Red Panda       | Ailurus        | fulgens        |
| African Elephant | Loxodonta      | africana       |
| Snow Leopard    | Panthera       | uncia          |
| Arctic Fox      | Vulpes         | lagopus        |
| Chimpanzee      | Pan            | troglodytes    |
| Siberian Tiger  | Panthera       | tigris altaica |
```

This technique can be used with custom formats, such as a **JSON structure:**

```
Put two fictional characters into JSON of the following format
{
  firstNameFictional:
  jobFictional:
}
```

Here's an example of how you can put two fictional characters into JSON format:

```
{
  "firstNameFictional": "Henry",
  "jobFictional": "Wizard"
},
{
  "firstNameFictional": "Frank",
  "jobFictional": "Hero"
}
```

Few shot learning

Using a user defined example conversation is what is called few shot learning, which provides the model examples of how it should respond to a given query. These examples serve to train the model how to respond.

For example, by providing the model a couple prompts and the expected response, it continues in the same pattern without instructing it what to do:

```
User: That was an awesome experience
Assistant: positive
User: I won't do that again
Assistant: negative
User: That was not worth my time
Assistant: negative
User: You can't miss this
Assistant:
```

If the model is provided with just **You can't miss** this with no additional context from few shot learning, the response isn't likely to be useful.

In practical terms, conversation history and few shot learning are sent to the model in the same way; each user message and assistant response is a discrete message in the message object. The ChatCompletion endpoint is optimized to include message history, regardless of if this message history is provided as few shot learning, or actual conversation history.

```
var ChatCompletionsOptions = new ChatCompletionsOptions()
{
   Messages =
   {
      new ChatMessage(ChatRole.System, "You are a helpful assistant."),
      new ChatMessage(ChatRole.User, "That was an awesome experience"),
      new ChatMessage(ChatRole.Assistant, "positive"),
      new ChatMessage(ChatRole.User, "I won't do that again"),
      new ChatMessage(ChatRole.Assistant, "negative"),
      new ChatMessage(ChatRole.User, "That was not worth my time"),
      new ChatMessage(ChatRole.Assistant, "negative"),
      new ChatMessage(ChatRole.User, "You can't miss this")
   }
};
```

While not ideal, similar results can be achieved with the Completion endpoint by including the conversation exchange within the single prompt, much like we saw with system messages. If you do need to include multiple messages in a single prompt with Completion, you can make it clear to the model by putting each message on a new line and labeling each line with the role. Formatting this prompt would be similar to how the conversation was presented at the top of this section.

Break down a complex task

Another technique for improved interaction is to divide complex prompts into multiple queries. This allows the model to better understand each individual part, and can improve the overall accuracy. Dividing your prompts also allows you to include the response from a previous prompt in a future prompt, and using that information in addition to the capabilities of the model to generate interesting responses.

For example, you could ask the model **Doug can ride down the zip line in 30 seconds, and takes 5 minutes to climb back up to the top. How many times can Doug ride the zip line in 17 minutes?.** The result is likely 3, which if Doug starts at the top of the zip line is incorrect.

A more informative answer could come from asking it multiple questions, about the round trip time to get back to the top of the zip line, and how to account for the fact that Doug starts at the top. Breaking down this problem reveals that Doug can, in fact, ride the zip line four times.

Chain of thought

One useful method to help you break down your task effectively is to ask the model to explain its chain of thought.

Asking a model to respond with the step by step process by which it determined the response is a helpful way to understand how the model is interpreting the prompt. By doing so, you can see where the model made an incorrect logical turn and better understand how to change your prompt to avoid the error. This technique can include asking it to cite its sources, like Bing chat does (which uses a GPT-4 generation model), and giving reasoning for why it determined its answer.

The chain of thought prompting technique is best used to help you iterate and improve on your prompts to get the highest quality answer from the model.

For example, asking the model What sport is easiest to learn but hardest to master? results in response with an answer, and a small explanation of why. However, when prompted with What sport is easiest to learn but hardest to master? Give a step by step approach of your thoughts, ending in your answer, the response is a complete explanation of how it arrived at its answer.

Sure, here's a step-by-step approach to answering the question:

1. Identify sports that are popular and have a large following.
2. Consider the basic mechanics of each sport and how easy they are to learn.
3. Think about the various techniques and skills required to play each sport at a high level, and how difficult they are to master.
4. Compare the relative ease of learning each sport to the difficulty of mastering it.
5. Based on the above analysis, determine which sport is easiest to learn but hardest to master.

Using this approach, I would say that the sport of golf is easiest to learn but hardest to master. Here's why:

1. Golf is a popular sport with a large following.
2. The basic mechanics of hitting a golf ball are relatively simple and can be learned quickly with some practice.
3. However, achieving consistency and mastering the various shots and techniques required to play at a high level can take years of practice and dedication.
4. Compared to other sports, the difference between a beginner and an intermediate golfer is relatively small, while the difference between an intermediate and an advanced golfer is much larger.
5. Therefore, I would say that golf is the sport that is easiest to learn but hardest to master.

Knowing the steps of reason the model used to form its response allows us to identify where we need to split the prompt, provide more information, or clarify instructions.