Search for articles...

Best practices for prompt engineering with the OpenAI API

How to give clear and effective instructions to OpenAI models

Updated over 6 months ago

Table of contents

How prompt engineering works

Due to the way OpenAI models are trained, there are specific prompt formats that work particularly well and lead to more useful model outputs.

The official prompt engineering guide by OpenAI is usually the best place to start for prompting tips.

Below we present a number of prompt formats we find work well, but feel free to explore different formats, which may fit your task better.

Rules of Thumb and Examples

Note: the "*{text input here}*" is a placeholder for actual text/context

1. Use the latest model

For best results, we generally recommend using the latest, most capable models. Newer models tend to be easier to prompt engineer.

2. Put instructions at the beginning of the prompt and use ### or """ to separate the instruction and context

Less effective ❌:

```
Summarize the text below as a bullet point list of the most important points.

{text input here}
```

Better ✅:

```
Summarize the text below as a bullet point list of the most important points.

Text: """
{text input here}
"""
```

3. Be specific, descriptive and as detailed as possible about the desired context, outcome, length, format, style, etc

Be specific about the context, outcome, length, format, style, etc

Less effective ❌:

```
Write a poem about OpenAI.
```

Better ✅:

```
Write a short inspiring poem about OpenAI, focusing on the recent DALL-E product launch (DA
```

4. Articulate the desired output format through examples

Less effective ❌:

```
Extract the entities mentioned in the text below. Extract the following 4 entity types: cor

Text: {text}
```

Show, and tell - the models respond better when shown specific format requirements. This also makes it easier to programmatically parse out multiple outputs reliably.

Better ✅:

```
Extract the important entities mentioned in the text below. First extract all company names

Desired format:
Company names: <comma_separated_list_of_company_names>
People names: -||-
Specific topics: -||-
General themes: -||-

Text: {text}
```

5. Start with zero-shot, then few-shot, neither of them worked, then fine-tune

✅ Zero-shot

```
Extract keywords from the below text.

Text: {text}

Keywords:
```

✅ Few-shot - provide a couple of examples

```
Extract keywords from the corresponding texts below.
```

```
Text 1: Stripe provides APIs that web developers can use to integrate payment processing ir
Keywords 1: Stripe, payment processing, APIs, web developers, websites, mobile applications
##
Text 2: OpenAI has trained cutting-edge language models that are very good at understanding
Keywords 2: OpenAI, language models, text processing, API.
##
Text 3: {text}
Keywords 3:
```

✅ Fine-tune: see fine-tune best practices here.

6. Reduce “fluffy” and imprecise descriptions

Less effective ❌:

```
The description for this product should be fairly short, a few sentences only, and not too
```

Better ✅:

```
Use a 3 to 5 sentence paragraph to describe this product.
```

7. Instead of just saying what not to do, say what to do instead

Less effective ❌:

```
The following is a conversation between an Agent and a Customer. DO NOT ASK USERNAME OR PAS

Customer: I can’t log in to my account.
Agent:
```

Better ✅:

```
The following is a conversation between an Agent and a Customer. The agent will attempt to

Customer: I can't log in to my account.
Agent:
```

8. Code Generation Specific - Use "leading words" to nudge the model toward a particular pattern

Less effective ❌:

```
# Write a simple python function that
# 1. Ask me for a number in mile
# 2. It converts miles to kilometers
```

In this code example below, adding "*import*" hints to the model that it should start writing in Python. (Similarly "SELECT" is a good hint for the start of a SQL statement.)

Better ✅:

```
# Write a simple python function that
# 1. Ask me for a number in mile
# 2. It converts miles to kilometers

import
```

9. Use the Generate Anything feature

Developers can use the 'Generate Anything' feature to describe a task or expected natural language output and receive a tailored prompt.

Learn more about using the 'Generate Anything' feature.

Parameters

Generally, we find that `model` and `temperature` are the most commonly used parameters to alter the model output.

1. `model` - Higher performance models are generally more expensive and may have higher latency.
2. `temperature` - A measure of how often the model outputs a less likely token. The higher the `temperature`, the more random (and usually creative) the output. This, however, is not the same as "truthfulness". For most factual use cases such as data extraction, and truthful Q&A, the `temperature` of 0 is best.
3. `max_tokens` (**maximum length**) - Does not control the length of the output, but a hard cutoff limit for token generation. Ideally you won't hit this limit often, as your model will stop either when it thinks it's finished, or when it hits a stop sequence you defined.
4. `stop` **(stop sequences)** - A set of characters (tokens) that, when generated, will cause the text generation to stop.

For other parameter descriptions see the API reference.

Related Articles

- Controlling the length of OpenAI model responses
- Doing math with OpenAI models
- Function Calling in the OpenAI API
- Using OpenAI o-series models and GPT-4o models on ChatGPT
- Prompt engineering best practices for ChatGPT

Did this answer your question?