Search for articles...

Best practices for prompt engineering with OpenAI API

Best practices for prompt engineering with OpenAI API

How to give clear and effective instructions to GPT-3 and Codex

J   Written by Jessica Shieh
     Updated over a week ago

Table of contents   ⌄

💡 If you're just getting started with OpenAI API, we recommend reading the
Introduction and Quickstart tutorials first.

How prompt engineering works

Due to the way the instruction-following models are trained or the data they are
trained on, there are specific prompt formats that work particularly well and
align better with the tasks at hand. Below we present a number of prompt
formats we find work reliably well, but feel free to explore different formats, which
may fit your task best.

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. As of November 2022, the best options are the **"text-davinci-003"** <u>model</u> for text generation, and the **"code-davinci-002"** model for code generation.

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 laur
```

4. Articulate the desired output format through examples ([example 1](#), [example 2](#)).

Less effective ❌:

```
Extract the entities mentioned in the text below. Extract the following 4 entity type
```
```
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

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

Text: {text}
```
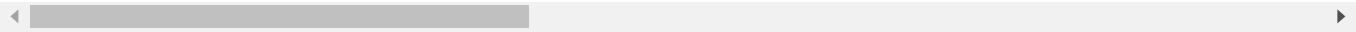
5. Start with zero-shot, then few-shot ([example](#)), neither of them worked, then fine-tune

✅ 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 process
Keywords 1: Stripe, payment processing, APIs, web developers, websites, mobile applic
##
Text 2: OpenAI has trained cutting-edge language models that are very good at underst
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 no
```

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 ❌:

```
ing is a conversation between an Agent and a Customer. DO NOT ASK USERNAME OR PASSWORD.

I can't log in to my account.
```

Better ✅:

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

```
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
```

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 more expensive and 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.

Additional Resources

If you're interested in additional resources, we recommend:

- Guides
  - Text completion - learn how to generate or edit text using our models
  - Code completion - explore prompt engineering for Codex
  - Fine-tuning - Learn how to train a custom model for your use case
  - Embeddings - learn how to search, classify, and compare text
  - Moderation
- OpenAI cookbook repo - contains example code and prompts for
  accomplishing common tasks with the API, including Question-answering
  with Embeddings
- Community Forum

Want a few flashcards to help the article stick better? 🧠

Generate Flashcards

🌳 Wisdolia

Did this answer your question?

😞 😐 😃

OpenAI

ChatGPT   API   DALL·E   Service Status