Text generation models
    Quickstart
        Python
        Javascript - NodeJS
        Curl
    Prompt engineering
    FAQ
        Which model should I use?
        How should I set the temperature parameter?
        Is fine-tuning available for the latest models?
        Do you store the data that is passed into the API?
        How can I make my application more safe?
        Should I use ChatGPT or the API?

Text generation models

OpenAI's text generation models (often called generative pre-trained transformers or large language models) have been trained to understand natural language, code, and images. The models provide text outputs in response to their inputs. The text inputs to these models are also referred to as "prompts". Designing a prompt is essentially how you “program” a large language model model, usually by providing instructions or some examples of how to successfully complete a task.

Using OpenAI's text generation models, you can build applications to:

- Draft documents
- Write computer code
- Answer questions about a knowledge base
- Analyze texts
- Give software a natural language interface
- Tutor in a range of subjects
- Translate languages
- Simulate characters for games

Prompt examples

Explore prompt examples for inspiration

To use one of these models via the OpenAI API, you’ll send a request to the Chat Completions API containing the inputs and your API key, and receive a response containing the model’s output.

You can experiment with various models in the chat playground. If you’re not sure which model to use then try `gpt-4o` if you need high intelligence or `gpt-4o-mini` if you need the fastest speed and lowest cost.

Quickstart

Chat models take a list of messages as input and return a model-generated message as output. Although the chat format is designed to make multi-turn conversations easy, it's just as useful for single-turn tasks without any conversation.

An example Chat Completions API call looks like the following:

Python

```python
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "What is a LLM?"}
  ]

)
```

Javascript - NodeJS

```javascript
import OpenAI from "openai";

const openai = new OpenAI();

async function main() {
  const completion = await openai.chat.completions.create({
    messages: [
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "What is a LLM?"}
      ],
    model: "gpt-4o-mini",
  });

  console.log(completion.choices[0]);
}
main();
```

Curl

```bash
curl https://api.openai.com/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "gpt-4o-mini",
    "messages": [
      {
        "role": "system",
        "content": "You are a helpful assistant."
      },
      {
        "role": "user",
        "content": "What is a LLM?"
      }
    ]
  }'
```

To learn more, you can view the Chat Completions guide.

Prompt engineering

An awareness of the best practices for working with OpenAI models can make a significant difference in application performance. The failure modes that each exhibit and the ways of working around or correcting those failure modes are not always intuitive. There is an entire field related to working with language models which has come to be known as "prompt engineering", but as the field has progressed its scope has outgrown merely engineering the prompt into engineering systems that use model queries as components.

To learn more, read our guide on prompt engineering which covers methods to improve model reasoning, reduce the likelihood of model hallucinations, and more.

You can also find many useful resources including code samples in the OpenAI Cookbook.

FAQ

Which model should I use?

We generally recommend that you default to using either `gpt-4o` or `gpt-4o-mini`.

If your use case requires high intelligence or reasoning about images as well as text, we recommend you evaluate both `gpt-4o` and `gpt-4-turbo` (although they have very similar intelligence, note that `gpt-4o` is both faster and cheaper).

If your use case requires the fastest speed and lowest cost, we recommend `gpt-4o-mini` since it is optimized for these aspects.

We recommend using `gpt-4o-mini` where you would have previously used `gpt-3.5-turbo` as it is cheaper with higher intelligence, has a larger context window (up to 128,000 tokens compared to 4,096 tokens for `gpt-3.5-turbo`), and is multimodal.

You can experiment in the playground to investigate which models provide the best price performance trade-off for your usage. A common design pattern is to use several distinct query types which are each dispatched to the model appropriate to handle them.

How should I set the temperature parameter?

Lower values for temperature result in more consistent outputs (e.g. 0.2), while higher values generate more diverse and creative results (e.g. 1.0). Select a temperature value based on the desired trade-off between coherence and creativity for your specific application. The temperature can range is from 0 to 2.

Is fine-tuning available for the latest models?

See the fine-tuning guide for the latest information on which models are available for fine-tuning and how to get started.

Do you store the data that is passed into the API?

As of March 1st, 2023, we retain your API data for 30 days but no longer use your data sent via the API to improve our models. Learn more in our data usage policy. Some endpoints offer zero retention.

How can I make my application more safe?

If you want to add a moderation layer to the outputs of the Chat API, you can follow our moderation guide to prevent content that violates OpenAI’s usage policies from being shown. We also encourage you to read our safety guide for more information on how to build safer systems.

Should I use ChatGPT or the API?

ChatGPT offers a chat interface for our models and a range of built-in features such as integrated browsing, code execution, plugins, and more. By contrast, using OpenAI's API provides more flexibility but requires that you write code or send the requests to our models programmatically.