Advanced usage

Copy page

Use advanced techniques for reproducibility and parameter tuning.

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.

Reproducible outputs

Chat Completions are non-deterministic by default (which means model outputs may differ from request to request). That being said, we offer some control towards deterministic outputs by giving you access to the seed parameter and the system_fingerprint response field.

To receive (mostly) deterministic outputs across API calls, you can:

- Set the seed parameter to any integer of your choice and use the same value across requests you'd like deterministic outputs for.
- Ensure all other parameters (like `prompt` or `temperature`) are the exact same across requests.

Sometimes, determinism may be impacted due to necessary changes OpenAI makes to model configurations on our end. To help you keep track of these changes, we expose the system_fingerprint field. If this value is different, you may see different outputs due to changes we've made on our systems.

Deterministic outputs

Explore the new seed parameter in the OpenAI cookbook

Managing tokens

Language models read and write text in chunks called tokens. In English, a token can be as short as one character or as long as one word (e.g., `a` or `apple`), and in some languages tokens can be even shorter than one character or even longer than one word.

As a rough rule of thumb, 1 token is approximately 4 characters or 0.75 words for English text.

Check out our Tokenizer tool to test specific strings and see how they are translated into tokens.

For example, the string `"ChatGPT is great!"` is encoded into six tokens: `["Chat", "G", "PT", " is", " great", "!"]`.

The total number of tokens in an API call affects:

- How much your API call costs, as you pay per token
- How long your API call takes, as writing more tokens takes more time
- Whether your API call works at all, as total tokens must be below the model's maximum limit (4097 tokens for `gpt-3.5-turbo`)

Both input and output tokens count toward these quantities. For example, if your API call used 10 tokens in the message input and you received 20 tokens in the message output, you would be billed for 30 tokens. Note however that for some models the price per token is different for tokens in the input vs. the output (see the pricing page for more information).

To see how many tokens are used by an API call, check the `usage` field in the API response (e.g., `response['usage']['total_tokens']`).

Chat models like `gpt-3.5-turbo` and `gpt-4-turbo-preview` use tokens in the same way as the models available in the completions API, but because of their message-based formatting, it's more difficult to count how many tokens will be used by a conversation.

DEEP DIVE

Counting tokens for chat API calls

To see how many tokens are in a text string without making an API call, use OpenAI's tiktoken Python library. Example code can be found in the OpenAI Cookbook's guide on how to count tokens with tiktoken.

Each message passed to the API consumes the number of tokens in the content, role, and other fields, plus a few extra for behind-the-scenes formatting. This may change slightly in the future.

If a conversation has too many tokens to fit within a model's maximum limit (e.g., more than 4097 tokens for `gpt-3.5-turbo` or more than 128k tokens for `gpt-4o`), you will have to truncate, omit, or otherwise shrink your text until it fits. Beware that if a message is removed from the messages input, the model will lose all knowledge of it.

Note that very long conversations are more likely to receive incomplete replies. For example, a `gpt-3.5-turbo` conversation that is 4090 tokens long will have its reply cut off after just 6 tokens.

Parameter details

Frequency and presence penalties

The frequency and presence penalties found in the Chat Completions API and Legacy Completions API can be used to reduce the likelihood of sampling repetitive sequences of tokens.

DEEP DIVE

Penalties behind the scenes

Reasonable values for the penalty coefficients are around 0.1 to 1 if the aim is to just reduce repetitive samples somewhat. If the aim is to strongly suppress repetition, then one can increase the coefficients up to 2, but this can noticeably degrade the quality of samples. Negative values can be used to increase the likelihood of repetition.

Token log probabilities

The logprobs parameter found in the Chat Completions API and Legacy Completions API, when requested, provides the log probabilities of each output token, and a limited number of the most likely tokens at each token position alongside their log probabilities. This can be useful in some cases to assess the confidence of the model in its output, or to examine alternative responses the model might have given.

Other parameters

See the full API reference documentation to learn more.