Chat completions Beta

Looking for ChatGPT? Head to chat.openai.com.

Using the OpenAI Chat API, you can build your own applications with `gpt-3.5-turbo` and `gpt-4` to do things like:

- Draft an email or other piece of writing
- Write Python code
- Answer questions about a set of documents
- Create conversational agents
- Give your software a natural language interface
- Tutor in a range of subjects
- Translate languages
- Simulate characters for video games and much more

This guide explains how to make an API call for chat-based language models and shares tips for getting good results. You can also experiment with the new chat format in the OpenAI Playground.

Introduction

Chat models take a series 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 conversations (such as those previously served by instruction following models like `text-davinci-003`).

An example API call looks as follows:

```
# Note: you need to be using OpenAI Python v0.27.0 for the code b     t
import openai

openai.ChatCompletion.create(
```

```
    model="gpt-3.5-turbo",
    messages=[
          {"role": "system", "content": "You are a helpful assistant."},
          {"role": "user", "content": "Who won the world series in 2020?"
          {"role": "assistant", "content": "The Los Angeles Dodgers won tl
          {"role": "user", "content": "Where was it played?"}
      ]
)
```

The main input is the messages parameter. Messages must be an array of message objects, where each object has a role (either "system", "user", or "assistant") and content (the content of the message). Conversations can be as short as 1 message or fill many pages.

user and assistant messages.

The system message helps set the behavior of the assistant. In the example above, the assistant was instructed with "You are a helpful assistant."

> gpt-3.5-turbo-0301 does not always pay strong attention to system messages. Future models will be trained to pay stronger attention to system messages.

The user messages help instruct the assistant. They can be generated by the end users of an application, or set by a developer as an instruction.

The assistant messages help store prior responses. They can also be written by a developer to help give examples of desired behavior.

Including the conversation history helps when user instructions refer to prior messages. In the example above, the user's final question of "Where was it played?" only makes sense in the context of the prior messages about the World Series of 2020. Because the models have no memory of past requests, all relevant information must be supplied via the conversation. If a conversation cannot fit within the model's token limit, it will need to be shortened in some way.

Response format

An example API response looks as follows:

```
{
 'id': 'chatcmpl-6p9XYPYSTTRi0xEviKjjilqrWU2Ve',
 'object': 'chat.completion',
 'created': 1677649420,
 'model': 'gpt-3.5-turbo',
 'usage': {'prompt_tokens': 56, 'completion_tokens': 31, 'total_tokens'
 'choices': [
  {
   'message': {
    'role': 'assistant',
    'content': 'The 2020 World Series was played in Arlington, Texas a
   'finish_reason': 'stop',
   'index': 0
  }
 ]
}
```

In Python, the assistant's reply can be extracted with `response['choices'][0]['message']['content']`.

Every response will include a `finish_reason`. The possible values for `finish_reason` are:

- `stop`: API returned complete model output
- `length`: Incomplete model output due to `max_tokens` parameter or token limit
- `content_filter`: Omitted content due to a flag from our content filters
- `null`: API response still in progress or incomplete

Managing tokens

Language models read 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.

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 (4096 tokens for `gpt-3.5-turbo-0301`)

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.

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` use tokens in the same way as other models, 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 4096 tokens for `gpt-3.5-turbo`), 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 too 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.

Instructing chat models

Best practices for instructing models may change from model version to version. The advice that follows applies to gpt-3.5-turbo-0301 and may not apply to future models.

Many conversations begin with a system message to gently instruct the assistant. For example, here is one of the system messages used for ChatGPT:

```
You are ChatGPT, a large language model trained by OpenAI. Answer as
concisely as possible. Knowledge cutoff: {knowledge_cutoff} Current date:
{current_date}
```

In general, `gpt-3.5-turbo-0301` does not pay strong attention to the system message, and therefore important instructions are often better placed in a user message.

If the model isn't generating the output you want, feel free to iterate and experiment with potential improvements. You can try approaches like:

- Make your instruction more explicit
- Specify the format you want the answer in
- Ask the model to think step by step or debate pros and cons before settling on an answer

For more prompt engineering ideas, read the OpenAI Cookbook guide on techniques to improve reliability.

Beyond the system message, the temperature and max tokens are two of many options developers have to influence the output of the chat models. For temperature, higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. In the case of max tokens, if you want to limit a response to a certain length, max tokens can be set to an arbitrary number. This may cause issues for example if you set the max tokens value to 5 since the output will be cut-off and the result will not make sense to users.

Chat vs Completions

Because `gpt-3.5-turbo` performs at a similar capability to `text-davinci-003` but at 10% the price per token, we recommend `gpt-3.5-turbo` for most use cases.

For many developers, the transition is as simple as rewriting and retesting a prompt.

For example, if you translated English to French with the following completions prompt:

```
Translate the following English text to French: "{text}"
```

An equivalent chat conversation could look like:

```
[
  {"role": "system", "content": "You are a helpful assistant that transla
  {"role": "user", "content": 'Translate the following English text to Fr
]
```

Or even just the user message:

```
[
  {"role": "user", "content": 'Translate the following English text to Fr
]
```

FAQ

Is fine-tuning available for `gpt-3.5-turbo`?

No. As of Mar 1, 2023, you can only fine-tune base GPT-3 models. See the fine-tuning guide for more details on how to use fine-tuned models.

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.

Adding a moderation layer

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.