

How to count tokens with Tiktoken



Ted Sanders

Dec 15, 2022

 Open in Github

`tiktoken` is a fast open-source tokenizer by OpenAI.

Given a text string (e.g., `"tiktoken is great!"`) and an encoding (e.g., `"cl100k_base"`), a tokenizer can split the text string into a list of tokens (e.g., `["t", "ik", "token", " is", " great", "!"]`).

Splitting text strings into tokens is useful because GPT models see text in the form of tokens. Knowing how many tokens are in a text string can tell you (a) whether the string is too long for a text model to process and (b) how much an OpenAI API call costs (as usage is priced by token).

Encodings

Encodings specify how text is converted into tokens. Different models use different encodings.

`tiktoken` supports three encodings used by OpenAI models:

Encoding name	OpenAI models
<code>cl100k_base</code>	<code>gpt-4</code> , <code>gpt-3.5-turbo</code> , <code>text-embedding-ada-002</code> , <code>text-embedding-3-small</code> , <code>text-embedding-3-large</code>
<code>p50k_base</code>	Codex models, <code>text-davinci-002</code> , <code>text-davinci-003</code>
<code>r50k_base</code> (or <code>gpt2</code>)	GPT-3 models like <code>davinci</code>

You can retrieve the encoding for a model using `tiktoken.encoding_for_model()` as follows:

```
encoding = tiktoken.encoding_for_model('gpt-3.5-turbo')
```



Note that `p50k_base` overlaps substantially with `r50k_base`, and for non-code applications, they will usually give the same tokens.

Tokenizer libraries by language

For `cl100k_base` and `p50k_base` encodings:

- Python: [tiktoken](#)
- .NET / C#: [SharpToken](#), [TiktokenSharp](#)
- Java: [jtokkit](#)
- Golang: [tiktoken-go](#)
- Rust: [tiktoken-rs](#)

For `r50k_base` (`gpt2`) encodings, tokenizers are available in many languages.

- Python: [tiktoken](#) (or alternatively [GPT2TokenizerFast](#))
- JavaScript: [gpt-3-encoder](#)
- .NET / C#: [GPT Tokenizer](#)
- Java: [gpt2-tokenizer-java](#)
- PHP: [GPT-3-Encoder-PHP](#)
- Golang: [tiktoken-go](#)
- Rust: [tiktoken-rs](#)

(OpenAI makes no endorsements or guarantees of third-party libraries.)

How strings are typically tokenized

In English, tokens commonly range in length from one character to one word (e.g., "t" or "great"), though in some languages tokens can be shorter than one character or longer than one word. Spaces are usually grouped with the starts of words (e.g., " is" instead of "is" or " " + "is"). You can quickly check how a string is tokenized at the [OpenAI Tokenizer](#), or the third-party [Tiktokenizer](#) webapp.

0. Install tiktoken

If needed, install tiktoken with pip :

```
%pip install --upgrade tiktoken  
%pip install --upgrade openai
```



1. Import tiktoken

```
import tiktoken
```



2. Load an encoding

Use `tiktoken.get_encoding()` to load an encoding by name.

The first time this runs, it will require an internet connection to download. Later runs won't need an internet connection.

```
encoding = tiktoken.get_encoding("cl100k_base")
```



Use `tiktoken.encoding_for_model()` to automatically load the correct encoding for a given model name.

```
encoding = tiktoken.encoding_for_model("gpt-3.5-turbo")
```



3. Turn text into tokens with `encoding.encode()`

The `.encode()` method converts a text string into a list of token integers.

```
encoding.encode("tiktoken is great!")
```

```
[83, 1609, 5963, 374, 2294, 0]
```

Count tokens by counting the length of the list returned by `.encode()`.

```
def num_tokens_from_string(string: str, encoding_name: str) -> int:
    """Returns the number of tokens in a text string."""
    encoding = tiktoken.get_encoding(encoding_name)
    num_tokens = len(encoding.encode(string))
    return num_tokens
```

```
num_tokens_from_string("tiktoken is great!", "cl100k_base")
```

```
6
```

4. Turn tokens into text with `encoding.decode()`

`.decode()` converts a list of token integers to a string.

```
encoding.decode([83, 1609, 5963, 374, 2294, 0])
```

```
'tiktoken is great!'
```

Warning: although `.decode()` can be applied to single tokens, beware that it can be lossy for tokens that aren't on utf-8 boundaries.

For single tokens, `.decode_single_token_bytes()` safely converts a single integer token to the bytes it represents.

```
[encoding.decode_single_token_bytes(token) for token in [83, 1609, 5963, 374, 2294, 0]]
```

```
[b't', b'ik', b'token', b' is', b' great', b'!']
```

(The `b` in front of the strings indicates that the strings are byte strings.)

5. Comparing encodings

Different encodings vary in how they split words, group spaces, and handle non-English characters. Using the methods above, we can compare different encodings on a few example strings.

```
def compare_encodings(example_string: str) -> None:
    """Prints a comparison of three string encodings."""
    # print the example string
    print(f'\nExample string: "{example_string}"')
    # for each encoding, print the # of tokens, the token integers, and the token bytes
    for encoding_name in ["r50k_base", "p50k_base", "cl100k_base"]:
        encoding = tiktoken.get_encoding(encoding_name)
        token_integers = encoding.encode(example_string)
        num_tokens = len(token_integers)
        token_bytes = [encoding.decode_single_token_bytes(token) for token in token_integers]
        print()
        print(f"{encoding_name}: {num_tokens} tokens")
        print(f"token integers: {token_integers}")
        print(f"token bytes: {token_bytes}")
```

```
compare_encodings("antidisestablishmentarianism")
```

```
Example string: "antidisestablishmentarianism"
```

```
r50k_base: 5 tokens
token integers: [415, 29207, 44390, 3699, 1042]
token bytes: [b'ant', b'idis', b'establishment', b'arian', b'ism']
```

```
p50k_base: 5 tokens
token integers: [415, 29207, 44390, 3699, 1042]
token bytes: [b'ant', b'idis', b'establishment', b'arian', b'ism']
```

```
cl100k_base: 6 tokens
```

```
token integers: [519, 85342, 34500, 479, 8997, 2191]
token bytes: [b'ant', b'idis', b'establish', b'ment', b'arian', b'ism']
```

```
compare_encodings("2 + 2 = 4")
```

Example string: "2 + 2 = 4"

```
r50k_base: 5 tokens
token integers: [17, 1343, 362, 796, 604]
token bytes: [b'2', b' +', b' 2', b' =', b' 4']
```

```
p50k_base: 5 tokens
token integers: [17, 1343, 362, 796, 604]
token bytes: [b'2', b' +', b' 2', b' =', b' 4']
```

```
cl100k_base: 7 tokens
token integers: [17, 489, 220, 17, 284, 220, 19]
token bytes: [b'2', b' +', b' ', b'2', b' =', b' ', b'4']
```

```
compare_encodings("お誕生日おめでとう")
```

Example string: "お誕生日おめでとう"

```
r50k_base: 14 tokens
token integers: [2515, 232, 45739, 243, 37955, 33768, 98, 2515, 232, 1792, 223, 30640, 1792, 223]
token bytes: [b'\xe3\x81', b'\x8a', b'\xe8\xaa', b'\x95', b'\xe7\x94\x9f', b'\xe6\x97', b'\xa5']
```

```
p50k_base: 14 tokens
token integers: [2515, 232, 45739, 243, 37955, 33768, 98, 2515, 232, 1792, 223, 30640, 1792, 223]
token bytes: [b'\xe3\x81', b'\x8a', b'\xe8\xaa', b'\x95', b'\xe7\x94\x9f', b'\xe6\x97', b'\xa5']
```

```
cl100k_base: 9 tokens
token integers: [33334, 45918, 243, 21990, 9080, 33334, 62004, 16556, 78699]
token bytes: [b'\xe3\x81\x8a', b'\xe8\xaa', b'\x95', b'\xe7\x94\x9f', b'\xe6\x97\xa5', b'\xa5']
```

6. Counting tokens for chat completions API calls

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

Below is an example function for counting tokens for messages passed to `gpt-3.5-turbo` or `gpt-4`.

Note that the exact way that tokens are counted from messages may change from model to model. Consider the counts from the function below an estimate, not a timeless guarantee.

In particular, requests that use the optional functions input will consume extra tokens on top of the estimates calculated below.

```
def num_tokens_from_messages(messages, model="gpt-3.5-turbo-0613"):
    """Return the number of tokens used by a list of messages."""
    try:
        encoding = tiktoken.encoding_for_model(model)
    except KeyError:
        print("Warning: model not found. Using cl100k_base encoding.")
        encoding = tiktoken.get_encoding("cl100k_base")
    if model in {
        "gpt-3.5-turbo-0613",
        "gpt-3.5-turbo-16k-0613",
        "gpt-4-0314",
        "gpt-4-32k-0314",
        "gpt-4-0613",
        "gpt-4-32k-0613",
    }:
        tokens_per_message = 3
        tokens_per_name = 1
    elif model == "gpt-3.5-turbo-0301":
        tokens_per_message = 4 # every message follows <|start|>{role/name}\n{content}<|enc
        tokens_per_name = -1 # if there's a name, the role is omitted
    elif "gpt-3.5-turbo" in model:
        print("Warning: gpt-3.5-turbo may update over time. Returning num tokens assuming gp
        return num_tokens_from_messages(messages, model="gpt-3.5-turbo-0613")
    elif "gpt-4" in model:
        print("Warning: gpt-4 may update over time. Returning num tokens assuming gpt-4-0613
        return num_tokens_from_messages(messages, model="gpt-4-0613")
    else:
        raise NotImplementedError(
            f"num_tokens_from_messages() is not implemented for model {model}. See https:/
        )
    num_tokens = 0
    for message in messages:
        num_tokens += tokens_per_message
        for key, value in message.items():
            num_tokens += len(encoding.encode(value))
            if key == "name":
                num_tokens += tokens_per_name
    num_tokens += 3 # every reply is primed with <|start|>assistant<|message|>
    return num_tokens
```

let's verify the function above matches the OpenAI API response

```

from openai import OpenAI
import os

client = OpenAI(api_key=os.environ.get("OPENAI_API_KEY", "<your OpenAI API key if not set as

example_messages = [
    {
        "role": "system",
        "content": "You are a helpful, pattern-following assistant that translates corporate
    },
    {
        "role": "system",
        "name": "example_user",
        "content": "New synergies will help drive top-line growth.",
    },
    {
        "role": "system",
        "name": "example_assistant",
        "content": "Things working well together will increase revenue.",
    },
    {
        "role": "system",
        "name": "example_user",
        "content": "Let's circle back when we have more bandwidth to touch base on opportuni
    },
    {
        "role": "system",
        "name": "example_assistant",
        "content": "Let's talk later when we're less busy about how to do better.",
    },
    {
        "role": "user",
        "content": "This late pivot means we don't have time to boil the ocean for the clie
    },
]

for model in [
    "gpt-3.5-turbo-0301",
    "gpt-3.5-turbo-0613",
    "gpt-3.5-turbo",
    "gpt-4-0314",
    "gpt-4-0613",
    "gpt-4",
]:
    print(model)
    # example token count from the function defined above
    print(f"{num_tokens_from_messages(example_messages, model)} prompt tokens counted by num
    # example token count from the OpenAI API
    response = client.chat.completions.create(model=model,
        messages=example_messages,
        temperature=0,
        max_tokens=1)
    print(f"{response.usage.prompt_tokens} prompt tokens counted by the OpenAI API.')
    print()

```

gpt-3.5-turbo-0301

127 prompt tokens counted by num_tokens_from_messages()

127 prompt tokens counted by the OpenAI API.

gpt-3.5-turbo-0613

129 prompt tokens counted by num_tokens_from_messages().

129 prompt tokens counted by the OpenAI API.

gpt-3.5-turbo

Warning: gpt-3.5-turbo may update over time. Returning num tokens assuming gpt-3.5-turbo-0613.

129 prompt tokens counted by num_tokens_from_messages().

129 prompt tokens counted by the OpenAI API.

gpt-4-0314

129 prompt tokens counted by num_tokens_from_messages().

129 prompt tokens counted by the OpenAI API.

gpt-4-0613

129 prompt tokens counted by num_tokens_from_messages().

129 prompt tokens counted by the OpenAI API.

gpt-4

Warning: gpt-4 may update over time. Returning num tokens assuming gpt-4-0613.

129 prompt tokens counted by num_tokens_from_messages().

129 prompt tokens counted by the OpenAI API.