Speech to text

Learn how to turn audio into text.

The Audio API provides two speech to text endpoints:

* `transcriptions`
* `translations`

Historically, both endpoints have been backed by our open source [Whisper model](https://openai.com/blog/whisper/) (`whisper-1`). The `transcriptions` endpoint now also supports higher quality model snapshots, with limited parameter support:

* `gpt-4o-mini-transcribe`
* `gpt-4o-transcribe`

All endpoints can be used to:

* Transcribe audio into whatever language the audio is in.
* Translate and transcribe the audio into English.

File uploads are currently limited to 25 MB, and the following input file types are supported: `mp3`, `mp4`, `mpeg`, `mpga`, `m4a`, `wav`, and `webm`.

Quickstart

Transcriptions
1

The transcriptions API takes as input the audio file you want to transcribe and the desired output file format for the transcription of the audio. All models support the same set of input formats. On output, `whisper-1` supports a range of formats (`json`, `text`, `srt`, `verbose_json`, `vtt`); the newer `gpt-4o-mini-transcribe` and `gpt-4o-transcribe` snapshots currently only support `json` or plain `text` responses.

Transcribe audio

```javascript
import fs from "fs";
import OpenAI from "openai";

const openai = new OpenAI();

const transcription = await openai.audio.transcriptions.create({
  file: fs.createReadStream("/path/to/file/audio.mp3"),
  model: "gpt-4o-transcribe",
});

console.log(transcription.text);
```

```python
from openai import OpenAI

client = OpenAI()
audio_file= open("/path/to/file/audio.mp3", "rb")

transcription = client.audio.transcriptions.create(
    model="gpt-4o-transcribe",
    file=audio_file
)
2
```

```
print(transcription.text)
```

```bash
curl --request POST \
  --url https://api.openai.com/v1/audio/transcriptions \
  --header "Authorization: Bearer $OPENAI_API_KEY" \
  --header 'Content-Type: multipart/form-data' \
  --form file=@/path/to/file/audio.mp3 \
  --form model=gpt-4o-transcribe
```

By default, the response type will be json with the raw text included.

{ "text": "Imagine the wildest idea that you've ever had, and you're curious about how it might scale to something that's a 100, a 1,000 times bigger. .... }

The Audio API also allows you to set additional parameters in a request. For example, if you want to set the `response_format` as `text`, your request would look like the following:

Additional options

```javascript
import fs from "fs";
import OpenAI from "openai";

const openai = new OpenAI();

const transcription = await openai.audio.transcriptions.create({
  file: fs.createReadStream("/path/to/file/speech.mp3"),
  model: "gpt-4o-transcribe",
  response_format: "text",
3
```

```
});

console.log(transcription.text);
```

```python
from openai import OpenAI

client = OpenAI()
audio_file = open("/path/to/file/speech.mp3", "rb")

transcription = client.audio.transcriptions.create(
    model="gpt-4o-transcribe",
    file=audio_file,
    response_format="text"
)

print(transcription.text)
```

```bash
curl --request POST \
  --url https://api.openai.com/v1/audio/transcriptions \
  --header "Authorization: Bearer $OPENAI_API_KEY" \
  --header 'Content-Type: multipart/form-data' \
  --form file=@/path/to/file/speech.mp3 \
  --form model=gpt-4o-transcribe \
  --form response_format=text
```

The [API Reference](/docs/api-reference/audio) includes the full list of available parameters.

The newer `gpt-4o-mini-transcribe` and `gpt-4o-transcribe` models currently

have a limited parameter surface: they only support `json` or `text` response formats. Other parameters, such as `timestamp_granularities`, require `verbose_json` output and are therefore only available when using `whisper-1`.

Translations

The translations API takes as input the audio file in any of the supported languages and transcribes, if necessary, the audio into English. This differs from our Transcriptions endpoint since the output is not in the original input language and is instead translated to English text. This endpoint supports only the `whisper-1` model.

Translate audio

```javascript
import fs from "fs";
import OpenAI from "openai";

const openai = new OpenAI();

const translation = await openai.audio.translations.create({
  file: fs.createReadStream("/path/to/file/german.mp3"),
  model: "whisper-1",
});

console.log(translation.text);
```

```python
from openai import OpenAI

client = OpenAI()
audio_file = open("/path/to/file/german.mp3", "rb")
```

```
translation = client.audio.translations.create(
    model="whisper-1",
    file=audio_file,
)

print(translation.text)
```

```bash
curl --request POST \
  --url https://api.openai.com/v1/audio/translations \
  --header "Authorization: Bearer $OPENAI_API_KEY" \
  --header 'Content-Type: multipart/form-data' \
  --form file=@/path/to/file/german.mp3 \
  --form model=whisper-1 \
```

In this case, the inputted audio was german and the outputted text looks like:

Hello, my name is Wolfgang and I come from Germany. Where are you heading today?

We only support translation into English at this time.

Supported languages

We currently [support the following languages](https://github.com/openai/whisper#available-models-and-languages) through both the `transcriptions` and `translations` endpoint:

Afrikaans, Arabic, Armenian, Azerbaijani, Belarusian, Bosnian, Bulgarian, Catalan, Chinese, Croatian, Czech, Danish, Dutch, English, Estonian, Finnish, French, Galician, German, Greek, Hebrew, Hindi, Hungarian, Icelandic, Indonesian, Italian, Japanese, Kannada, Kazakh, Korean, Latvian, Lithuanian, Macedonian, Malay,

Marathi, Maori, Nepali, Norwegian, Persian, Polish, Portuguese, Romanian, Russian, Serbian, Slovak, Slovenian, Spanish, Swahili, Swedish, Tagalog, Tamil, Thai, Turkish, Ukrainian, Urdu, Vietnamese, and Welsh.

While the underlying model was trained on 98 languages, we only list the languages that exceeded <50% [word error rate](https://en.wikipedia.org/wiki/Word_error_rate) (WER) which is an industry standard benchmark for speech to text model accuracy. The model will return results for languages not listed above but the quality will be low.

We support some ISO 639-1 and 639-3 language codes for GPT-4o based models. For language codes we don't have, try prompting for specific languages (i.e., "Output in English").

Timestamps

By default, the Transcriptions API will output a transcript of the provided audio in text. The [`timestamp_granularities[]` parameter](/docs/api-reference/audio/createTranscription#audio-createtranscription-timestamp_granularities) enables a more structured and timestamped json output format, with timestamps at the segment, word level, or both. This enables word-level precision for transcripts and video edits, which allows for the removal of specific frames tied to individual words.

Timestamp options

```javascript
import fs from "fs";
import OpenAI from "openai";

const openai = new OpenAI();

const transcription = await openai.audio.transcriptions.create({
  file: fs.createReadStream("audio.mp3"),
```

```
  model: "whisper-1",
  response_format: "verbose_json",
  timestamp_granularities: ["word"]
});

console.log(transcription.words);
```

```python
from openai import OpenAI

client = OpenAI()
audio_file = open("/path/to/file/speech.mp3", "rb")

transcription = client.audio.transcriptions.create(
  file=audio_file,
  model="whisper-1",
  response_format="verbose_json",
  timestamp_granularities=["word"]
)

print(transcription.words)
```

```bash
curl https://api.openai.com/v1/audio/transcriptions \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: multipart/form-data" \
  -F file="@/path/to/file/audio.mp3" \
  -F "timestamp_granularities[]=word" \
  -F model="whisper-1" \
  -F response_format="verbose_json"
```

The `timestamp_granularities[]` parameter is only supported for `whisper-1`.

Longer inputs

By default, the Transcriptions API only supports files that are less than 25 MB. If you have an audio file that is longer than that, you will need to break it up into chunks of 25 MB's or less or used a compressed audio format. To get the best performance, we suggest that you avoid breaking the audio up mid-sentence as this may cause some context to be lost.

One way to handle this is to use the [PyDub open source Python package](https://github.com/jiaaro/pydub) to split the audio:

```python
from pydub import AudioSegment

song = AudioSegment.from_mp3("good_morning.mp3")

# PyDub handles time in milliseconds
ten_minutes = 10 * 60 * 1000

first_10_minutes = song[:ten_minutes]

first_10_minutes.export("good_morning_10.mp3", format="mp3")
```

OpenAI makes no guarantees about the usability or security of 3rd party software like PyDub.

Prompting

You can use a [prompt](/docs/api-

reference/audio/createTranscription#audio/createTranscription-prompt) to improve the quality of the transcripts generated by the Transcriptions API.

Prompting

```javascript
import fs from "fs";
import OpenAI from "openai";

const openai = new OpenAI();

const transcription = await openai.audio.transcriptions.create({
  file: fs.createReadStream("/path/to/file/speech.mp3"),
  model: "gpt-4o-transcribe",
  response_format: "text",
  prompt:"The following conversation is a lecture about the recent developments around OpenAI, GPT-4.5 and the future of AI.",
});

console.log(transcription.text);
```

```python
from openai import OpenAI

client = OpenAI()
audio_file = open("/path/to/file/speech.mp3", "rb")

transcription = client.audio.transcriptions.create(
  model="gpt-4o-transcribe",
  file=audio_file,
  response_format="text",
  prompt="The following conversation is a lecture about the recent developments around OpenAI, GPT-4.5 and the future of AI."
10
```

)

print(transcription.text)
```

```bash
curl --request POST \
  --url https://api.openai.com/v1/audio/transcriptions \
  --header "Authorization: Bearer $OPENAI_API_KEY" \
  --header 'Content-Type: multipart/form-data' \
  --form file=@/path/to/file/speech.mp3 \
  --form model=gpt-4o-transcribe \
  --form prompt="The following conversation is a lecture about the recent developments around OpenAI, GPT-4.5 and the future of AI."
```

For `gpt-4o-transcribe` and `gpt-4o-mini-transcribe`, you can use the `prompt` parameter to improve the quality of the transcription by giving the model additional context similarly to how you would prompt other GPT-4o models.

Here are some examples of how prompting can help in different scenarios:

1. Prompts can help correct specific words or acronyms that the model misrecognizes in the audio. For example, the following prompt improves the transcription of the words DALL·E and GPT-3, which were previously written as "GDP 3" and "DALI": "The transcript is about OpenAI which makes technology like DALL·E, GPT-3, and ChatGPT with the hope of one day building an AGI system that benefits all of humanity."
2. To preserve the context of a file that was split into segments, prompt the model with the transcript of the preceding segment. The model uses relevant information from the previous audio, improving transcription accuracy. The `whisper-1` model only considers the final 224 tokens of the prompt and ignores anything earlier. For multilingual inputs, Whisper uses a custom
```

tokenizer. For English-only inputs, it uses the standard GPT-2 tokenizer. Find both tokenizers in the open source [Whisper Python package](https://github.com/openai/whisper/blob/main/whisper/tokenizer.py#L361).
3. Sometimes the model skips punctuation in the transcript. To prevent this, use a simple prompt that includes punctuation: "Hello, welcome to my lecture."
4. The model may also leave out common filler words in the audio. If you want to keep the filler words in your transcript, use a prompt that contains them: "Umm, let me think like, hmm... Okay, here's what I'm, like, thinking."
5. Some languages can be written in different ways, such as simplified or traditional Chinese. The model might not always use the writing style that you want for your transcript by default. You can improve this by using a prompt in your preferred writing style.

For `whisper-1`, the model tries to match the style of the prompt, so it's more likely to use capitalization and punctuation if the prompt does too. However, the current prompting system is more limited than our other language models and provides limited control over the generated text.

You can find more examples on improving your `whisper-1` transcriptions in the [improving reliability](/docs/guides/speech-to-text#improving-reliability) section.

Streaming transcriptions

There are two ways you can stream your transcription depending on your use case and whether you are trying to transcribe an already completed audio recording or handle an ongoing stream of audio and use OpenAI for turn detection.

Streaming the transcription of a completed audio recording

If you have an already completed audio recording, either because it's an audio

file or you are using your own turn detection (like push-to-talk), you can use our Transcription API with `stream=True` to receive a stream of [transcript events](/docs/api-reference/audio/transcript-text-delta-event) as soon as the model is done transcribing that part of the audio.

Stream transcriptions

```javascript
import fs from "fs";
import OpenAI from "openai";

const openai = new OpenAI();

const stream = await openai.audio.transcriptions.create({
  file: fs.createReadStream("/path/to/file/speech.mp3"),
  model: "gpt-4o-mini-transcribe",
  response_format: "text",
  stream: true,
});

for await (const event of stream) {
  console.log(event);
}
```

```python
from openai import OpenAI

client = OpenAI()
audio_file = open("/path/to/file/speech.mp3", "rb")

stream = client.audio.transcriptions.create(
  model="gpt-4o-mini-transcribe",
  file=audio_file,
13
```

```
  response_format="text",
  stream=True
)

for event in stream:
  print(event)
```

```bash
curl --request POST \
  --url https://api.openai.com/v1/audio/transcriptions \
  --header "Authorization: Bearer $OPENAI_API_KEY" \
  --header 'Content-Type: multipart/form-data' \
  --form file=@example.wav \
  --form model=whisper-1 \
  --form stream=True
```

You will receive a stream of `transcript.text.delta` events as soon as the model is done transcribing that part of the audio, followed by a `transcript.text.done` event when the transcription is complete that includes the full transcript.

Additionally, you can use the `include[]` parameter to include `logprobs` in the response to get the log probabilities of the tokens in the transcription. These can be helpful to determine how confident the model is in the transcription of that particular part of the transcript.

Streamed transcription is not supported in `whisper-1`.

Streaming the transcription of an ongoing audio recording

In the Realtime API, you can stream the transcription of an ongoing audio recording. To start a streaming session with the Realtime API, create a WebSocket connection with the following URL:

```text
wss://api.openai.com/v1/realtime?intent=transcription
```

Below is an example payload for setting up a transcription session:

```json
{
  "type": "transcription_session.update",
  "input_audio_format": "pcm16",
  "input_audio_transcription": {
    "model": "gpt-4o-transcribe",
    "prompt": "",
    "language": ""
  },
  "turn_detection": {
    "type": "server_vad",
    "threshold": 0.5,
    "prefix_padding_ms": 300,
    "silence_duration_ms": 500,
  },
  "input_audio_noise_reduction": {
    "type": "near_field"
  },
  "include": [
    "item.input_audio_transcription.logprobs"
  ]
}
```

To stream audio data to the API, append audio buffers:

```json
15
```

```
{
  "type": "input_audio_buffer.append",
  "audio": "Base64EncodedAudioData"
}
```

When in VAD mode, the API will respond with `input_audio_buffer.committed` every time a chunk of speech has been detected. Use `input_audio_buffer.committed.item_id` and `input_audio_buffer.committed.previous_item_id` to enforce the ordering.

The API responds with transcription events indicating speech start, stop, and completed transcriptions.

The primary resource used by the streaming ASR API is the `TranscriptionSession`:

```json
{
  "object": "realtime.transcription_session",
  "id": "string",
  "input_audio_format": "pcm16",
  "input_audio_transcription": [{
    "model": "whisper-1" | "gpt-4o-transcribe" | "gpt-4o-mini-transcribe",
    "prompt": "string",
    "language": "string"
  }],
  "turn_detection": {
    "type": "server_vad",
    "threshold": "float",
    "prefix_padding_ms": "integer",
    "silence_duration_ms": "integer",
  } | null,
  "input_audio_noise_reduction": {
```

```
    "type": "near_field" | "far_field"
  },
  "include": ["string"]
}
```

Authenticate directly through the WebSocket connection using your API key or an ephemeral token obtained from:

```text
POST /v1/realtime/transcription_sessions
```

This endpoint returns an ephemeral token (`client_secret`) to securely authenticate WebSocket connections.

Improving reliability

One of the most common challenges faced when using Whisper is the model often does not recognize uncommon words or acronyms. Here are some different techniques to improve the reliability of Whisper in these cases:

Using the prompt parameter

The first method involves using the optional prompt parameter to pass a dictionary of the correct spellings.

Because it wasn't trained with instruction-following techniques, Whisper operates more like a base GPT model. Keep in mind that Whisper only considers the first 224 tokens of the prompt.

Prompt parameter

```javascript
import fs from "fs";
import OpenAI from "openai";

const openai = new OpenAI();

const transcription = await openai.audio.transcriptions.create({
  file: fs.createReadStream("/path/to/file/speech.mp3"),
  model: "whisper-1",
  response_format: "text",
  prompt:"ZyntriQix, Digique Plus, CynapseFive, VortiQore V8, EchoNix Array,
OrbitalLink Seven, DigiFractal Matrix, PULSE, RAPT, B.R.I.C.K., Q.U.A.R.T.Z.,
F.L.I.N.T.",
});

console.log(transcription.text);
```

```python
from openai import OpenAI

client = OpenAI()
audio_file = open("/path/to/file/speech.mp3", "rb")

transcription = client.audio.transcriptions.create(
  model="whisper-1",
  file=audio_file,
  response_format="text",
  prompt="ZyntriQix, Digique Plus, CynapseFive, VortiQore V8, EchoNix Array,
OrbitalLink Seven, DigiFractal Matrix, PULSE, RAPT, B.R.I.C.K., Q.U.A.R.T.Z.,
F.L.I.N.T."
)

print(transcription.text)
18
```

```

```bash
curl --request POST \
  --url https://api.openai.com/v1/audio/transcriptions \
  --header "Authorization: Bearer $OPENAI_API_KEY" \
  --header 'Content-Type: multipart/form-data' \
  --form file=@/path/to/file/speech.mp3 \
  --form model=whisper-1 \
  --form prompt="ZyntriQix, Digique Plus, CynapseFive, VortiQore V8, EchoNix Array, OrbitalLink Seven, DigiFractal Matrix, PULSE, RAPT, B.R.I.C.K., Q.U.A.R.T.Z., F.L.I.N.T."
```

While it increases reliability, this technique is limited to 224 tokens, so your list of SKUs needs to be relatively small for this to be a scalable solution.

Post-processing with GPT-4

The second method involves a post-processing step using GPT-4 or GPT-3.5-Turbo.

We start by providing instructions for GPT-4 through the `system_prompt` variable. Similar to what we did with the prompt parameter earlier, we can define our company and product names.

Post-processing

```javascript
const systemPrompt = `
You are a helpful assistant for the company ZyntriQix. Your task is to correct any spelling discrepancies in the transcribed text. Make sure that the names of the following products are spelled correctly: ZyntriQix, Digique Plus, CynapseFive, VortiQore V8, EchoNix Array,
```
```

OrbitalLink Seven, DigiFractal Matrix, PULSE, RAPT, B.R.I.C.K.,
Q.U.A.R.T.Z., F.L.I.N.T. Only add necessary punctuation such as
periods, commas, and capitalization, and use only the context provided.
`;

```
const transcript = await transcribe(audioFile);
const completion = await openai.chat.completions.create({
model: "gpt-4.1",
temperature: temperature,
messages: [
  {
    role: "system",
    content: systemPrompt
  },
  {
    role: "user",
    content: transcript
  }
],
store: true,
});

console.log(completion.choices[0].message.content);
```

```python
system_prompt = """
You are a helpful assistant for the company ZyntriQix. Your task is to correct any spelling discrepancies in the transcribed text. Make sure that the names of the following products are spelled correctly: ZyntriQix, Digique Plus, CynapseFive, VortiQore V8, EchoNix Array, OrbitalLink Seven, DigiFractal Matrix, PULSE, RAPT, B.R.I.C.K., Q.U.A.R.T.Z., F.L.I.N.T. Only add necessary punctuation such as periods, commas, and capitalization, and use only the context provided.

```

```
"""

def generate_corrected_transcript(temperature, system_prompt, audio_file):
  response = client.chat.completions.create(
      model="gpt-4.1",
      temperature=temperature,
      messages=[
          {
              "role": "system",
              "content": system_prompt
          },
          {
              "role": "user",
              "content": transcribe(audio_file, "")
          }
      ]
  )
  return completion.choices[0].message.content
corrected_text = generate_corrected_transcript(
  0, system_prompt, fake_company_filepath
)
```

If you try this on your own audio file, you'll see that GPT-4 corrects many misspellings in the transcript. Due to its larger context window, this method might be more scalable than using Whisper's prompt parameter. It's also more reliable, as GPT-4 can be instructed and guided in ways that aren't possible with Whisper due to its lack of instruction following.