Test to Speech Models:  
Text to speech

==============

Learn how to turn text into lifelike spoken audio.

The Audio API provides a [`speech`](/docs/api-reference/audio/createSpeech) endpoint based on our [GPT-4o mini TTS (text-to-speech) model](/docs/models/gpt-4o-mini-tts). It comes with 11 built-in voices and can be used to:

\* Narrate a written blog post

\* Produce spoken audio in multiple languages

\* Give realtime audio output using streaming

Here's an example of the `alloy` voice:

Our [usage policies](https://openai.com/policies/usage-policies) require you to provide a clear disclosure to end users that the TTS voice they are hearing is AI-generated and not a human voice.

Quickstart

----------

The `speech` endpoint takes three key inputs:

1. The [model](/docs/api-reference/audio/createSpeech#audio-createspeech-model) you're using

2. The [text](/docs/api-reference/audio/createSpeech#audio-createspeech-input) to be turned into audio

3. The [voice](/docs/api-reference/audio/createSpeech#audio-createspeech-voice) that will speak the output

Here's a simple request example:

Generate spoken audio from input text

```javascript

import fs from "fs";

import path from "path";

import OpenAI from "openai";

const openai = new OpenAI();

const speechFile = path.resolve("./speech.mp3");

const mp3 = await openai.audio.speech.create({

model: "gpt-4o-mini-tts",

voice: "coral",

input: "Today is a wonderful day to build something people love!",

instructions: "Speak in a cheerful and positive tone.",

});

const buffer = Buffer.from(await mp3.arrayBuffer());

await fs.promises.writeFile(speechFile, buffer);

```

```python

from pathlib import Path

from openai import OpenAI

client = OpenAI()

speech\_file\_path = Path(\_\_file\_\_).parent / "speech.mp3"

with client.audio.speech.with\_streaming\_response.create(

model="gpt-4o-mini-tts",

voice="coral",

input="Today is a wonderful day to build something people love!",

instructions="Speak in a cheerful and positive tone.",

) as response:

response.stream\_to\_file(speech\_file\_path)

```

```bash

curl https://api.openai.com/v1/audio/speech \

-H "Authorization: Bearer $OPENAI\_API\_KEY" \

-H "Content-Type: application/json" \

-d '{

"model": "gpt-4o-mini-tts",

"input": "Today is a wonderful day to build something people love!",

"voice": "coral",

"instructions": "Speak in a cheerful and positive tone."

}' \

--output speech.mp3

```

By default, the endpoint outputs an MP3 of the spoken audio, but you can configure it to output any [supported format](#supported-output-formats).

### Text-to-speech models

For intelligent realtime applications, use the `gpt-4o-mini-tts` model, our newest and most reliable text-to-speech model. You can prompt the model to control aspects of speech, including:

\* Accent

\* Emotional range

\* Intonation

\* Impressions

\* Speed of speech

\* Tone

\* Whispering

Our other text-to-speech models are `tts-1` and `tts-1-hd`. The `tts-1` model provides lower latency, but at a lower quality than the `tts-1-hd` model.

### Voice options

The TTS endpoint provides 11 built‑in voices to control how speech is rendered from text. \*\*Hear and play with these voices in [OpenAI.fm](https://openai.fm), our interactive demo for trying the latest text-to-speech model in the OpenAI API\*\*. Voices are currently optimized for English.

\* `alloy`

\* `ash`

\* `ballad`

\* `coral`

\* `echo`

\* `fable`

\* `onyx`

\* `nova`

\* `sage`

\* `shimmer`

If you're using the [Realtime API](/docs/guides/realtime), note that the set of available voices is slightly different—see the [realtime model capabilities guide](/docs/guides/realtime-model-capabilities#voice-options) for current realtime voices.

### Streaming realtime audio

The Speech API provides support for realtime audio streaming using [chunk transfer encoding](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Transfer-Encoding). This means the audio can be played before the full file is generated and made accessible.

Stream spoken audio from input text directly to your speakers

```javascript

import OpenAI from "openai";

import { playAudio } from "openai/helpers/audio";

const openai = new OpenAI();

const response = await openai.audio.speech.create({

model: "gpt-4o-mini-tts",

voice: "coral",

input: "Today is a wonderful day to build something people love!",

instructions: "Speak in a cheerful and positive tone.",

response\_format: "wav",

});

await playAudio(response);

```

```python

import asyncio

from openai import AsyncOpenAI

from openai.helpers import LocalAudioPlayer

openai = AsyncOpenAI()

async def main() -> None:

async with openai.audio.speech.with\_streaming\_response.create(

model="gpt-4o-mini-tts",

voice="coral",

input="Today is a wonderful day to build something people love!",

instructions="Speak in a cheerful and positive tone.",

response\_format="pcm",

) as response:

await LocalAudioPlayer().play(response)

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.run(main())

```

```bash

curl https://api.openai.com/v1/audio/speech \

-H "Authorization: Bearer $OPENAI\_API\_KEY" \

-H "Content-Type: application/json" \

-d '{

"model": "gpt-4o-mini-tts",

"input": "Today is a wonderful day to build something people love!",

"voice": "coral",

"instructions": "Speak in a cheerful and positive tone.",

"response\_format": "wav"

}' | ffplay -i -

```

For the fastest response times, we recommend using `wav` or `pcm` as the response format.

Supported output formats

------------------------

The default response format is `mp3`, but other formats like `opus` and `wav` are available.

\* \*\*MP3\*\*: The default response format for general use cases.

\* \*\*Opus\*\*: For internet streaming and communication, low latency.

\* \*\*AAC\*\*: For digital audio compression, preferred by YouTube, Android, iOS.

\* \*\*FLAC\*\*: For lossless audio compression, favored by audio enthusiasts for archiving.

\* \*\*WAV\*\*: Uncompressed WAV audio, suitable for low-latency applications to avoid decoding overhead.

\* \*\*PCM\*\*: Similar to WAV but contains the raw samples in 24kHz (16-bit signed, low-endian), without the header.

Supported languages

-------------------

The TTS model generally follows the Whisper model in terms of language support. Whisper [supports the following languages](https://github.com/openai/whisper#available-models-and-languages) and performs well, despite voices being optimized for English:

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.

You can generate spoken audio in these languages by providing input text in the language of your choice.

Customization and ownership

---------------------------

### Custom voices

We do not support custom voices or creating a copy of your own voice.

### Who owns the output?

As with all outputs from our API, the person who created them owns the output. You are still required to inform end users that they are hearing audio generated by AI and not a real person talking to them.

Was this page useful?  
  
Speech to Text:  
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

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

)

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",

});

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."

)

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](#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,

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

{

"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)

```

```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-4o",

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-4o",

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.

Was this page useful?  
  
Audio and Speech:  
Audio and speech

================

Explore audio and speech features in the OpenAI API.

The OpenAI API provides a range of audio capabilities. If you know what you want to build, find your use case below to get started. If you're not sure where to start, read this page as an overview.

Build with audio

----------------

[

![Build voice agents](https://cdn.openai.com/API/docs/images/voice-agents-rounded.png)

Build voice agents

Build interactive voice-driven applications.

](/docs/guides/voice-agents)[

![Transcribe audio](https://cdn.openai.com/API/docs/images/stt-rounded.png)

Transcribe audio

Convert speech to text instantly and accurately.

](/docs/guides/speech-to-text)[

![Speak text](https://cdn.openai.com/API/docs/images/tts-rounded.png)

Speak text

Turn text into natural-sounding speech in real time.

](/docs/guides/text-to-speech)

A tour of audio use cases

-------------------------

LLMs can process audio by using sound as input, creating sound as output, or both. OpenAI has several API endpoints that help you build audio applications or voice agents.

### Voice agents

Voice agents understand audio to handle tasks and respond back in natural language. There are two main ways to approach voice agents: either with speech-to-speech models and the [Realtime API](/docs/guides/realtime), or by chaining together a speech-to-text model, a text language model to process the request, and a text-to-speech model to respond. Speech-to-speech is lower latency and more natural, but chaining together a voice agent is a reliable way to extend a text-based agent into a voice agent. If you are already using the [Agents SDK](/docs/guides/agents), you can [extend your existing agents with voice capabilities](https://openai.github.io/openai-agents-python/voice/quickstart/) using the chained approach.

### Streaming audio

Process audio in real time to build voice agents and other low-latency applications, including transcription use cases. You can stream audio in and out of a model with the [Realtime API](/docs/guides/realtime). Our advanced speech models provide automatic speech recognition for improved accuracy, low-latency interactions, and multilingual support.

### Text to speech

For turning text into speech, use the [Audio API](/docs/api-reference/audio/) `audio/speech` endpoint. Models compatible with this endpoint are `gpt-4o-mini-tts`, `tts-1`, and `tts-1-hd`. With `gpt-4o-mini-tts`, you can ask the model to speak a certain way or with a certain tone of voice.

### Speech to text

For speech to text, use the [Audio API](/docs/api-reference/audio/) `audio/transcriptions` endpoint. Models compatible with this endpoint are `gpt-4o-transcribe`, `gpt-4o-mini-transcribe`, and `whisper-1`. With streaming, you can continuously pass in audio and get a continuous stream of text back.

Choosing the right API

----------------------

There are multiple APIs for transcribing or generating audio:

|API|Supported modalities|Streaming support|

|---|---|---|

|Realtime API|Audio and text inputs and outputs|Audio streaming in and out|

|Chat Completions API|Audio and text inputs and outputs|Audio streaming out|

|Transcription API|Audio inputs|Audio streaming out|

|Speech API|Text inputs and audio outputs|Audio streaming out|

### General use APIs vs. specialized APIs

The main distinction is general use APIs vs. specialized APIs. With the Realtime and Chat Completions APIs, you can use our latest models' native audio understanding and generation capabilities and combine them with other features like function calling. These APIs can be used for a wide range of use cases, and you can select the model you want to use.

On the other hand, the Transcription, Translation and Speech APIs are specialized to work with specific models and only meant for one purpose.

### Talking with a model vs. controlling the script

Another way to select the right API is asking yourself how much control you need. To design conversational interactions, where the model thinks and responds in speech, use the Realtime or Chat Completions API, depending if you need low-latency or not.

You won't know exactly what the model will say ahead of time, as it will generate audio responses directly, but the conversation will feel natural.

For more control and predictability, you can use the Speech-to-text / LLM / Text-to-speech pattern, so you know exactly what the model will say and can control the response. Please note that with this method, there will be added latency.

This is what the Audio APIs are for: pair an LLM with the `audio/transcriptions` and `audio/speech` endpoints to take spoken user input, process and generate a text response, and then convert that to speech that the user can hear.

### Recommendations

\* If you need [real-time interactions](/docs/guides/realtime-conversations) or [transcription](/docs/guides/realtime-transcription), use the Realtime API.

\* If realtime is not a requirement but you're looking to build a [voice agent](/docs/guides/voice-agents) or an audio-based application that requires features such as [function calling](/docs/guides/function-calling), use the Chat Completions API.

\* For use cases with one specific purpose, use the Transcription, Translation, or Speech APIs.

Add audio to your existing application

--------------------------------------

Models such as GPT-4o or GPT-4o mini are natively multimodal, meaning they can understand and generate multiple modalities as input and output.

If you already have a text-based LLM application with the [Chat Completions endpoint](/docs/api-reference/chat/), you may want to add audio capabilities. For example, if your chat application supports text input, you can add audio input and output—just include `audio` in the `modalities` array and use an audio model, like `gpt-4o-audio-preview`.

Audio is not yet supported in the [Responses API](/docs/api-reference/chat/completions/responses).

Audio output from model

Create a human-like audio response to a prompt

```javascript

import { writeFileSync } from "node:fs";

import OpenAI from "openai";

const openai = new OpenAI();

// Generate an audio response to the given prompt

const response = await openai.chat.completions.create({

model: "gpt-4o-audio-preview",

modalities: ["text", "audio"],

audio: { voice: "alloy", format: "wav" },

messages: [

{

role: "user",

content: "Is a golden retriever a good family dog?"

}

],

store: true,

});

// Inspect returned data

console.log(response.choices[0]);

// Write audio data to a file

writeFileSync(

"dog.wav",

Buffer.from(response.choices[0].message.audio.data, 'base64'),

{ encoding: "utf-8" }

);

```

```python

import base64

from openai import OpenAI

client = OpenAI()

completion = client.chat.completions.create(

model="gpt-4o-audio-preview",

modalities=["text", "audio"],

audio={"voice": "alloy", "format": "wav"},

messages=[

{

"role": "user",

"content": "Is a golden retriever a good family dog?"

}

]

)

print(completion.choices[0])

wav\_bytes = base64.b64decode(completion.choices[0].message.audio.data)

with open("dog.wav", "wb") as f:

f.write(wav\_bytes)

```

```bash

curl "https://api.openai.com/v1/chat/completions" \

-H "Content-Type: application/json" \

-H "Authorization: Bearer $OPENAI\_API\_KEY" \

-d '{

"model": "gpt-4o-audio-preview",

"modalities": ["text", "audio"],

"audio": { "voice": "alloy", "format": "wav" },

"messages": [

{

"role": "user",

"content": "Is a golden retriever a good family dog?"

}

]

}'

```

Audio input to model

Use audio inputs for prompting a model

```javascript

import OpenAI from "openai";

const openai = new OpenAI();

// Fetch an audio file and convert it to a base64 string

const url = "https://cdn.openai.com/API/docs/audio/alloy.wav";

const audioResponse = await fetch(url);

const buffer = await audioResponse.arrayBuffer();

const base64str = Buffer.from(buffer).toString("base64");

const response = await openai.chat.completions.create({

model: "gpt-4o-audio-preview",

modalities: ["text", "audio"],

audio: { voice: "alloy", format: "wav" },

messages: [

{

role: "user",

content: [

{ type: "text", text: "What is in this recording?" },

{ type: "input\_audio", input\_audio: { data: base64str, format: "wav" }}

]

}

],

store: true,

});

console.log(response.choices[0]);

```

```python

import base64

import requests

from openai import OpenAI

client = OpenAI()

# Fetch the audio file and convert it to a base64 encoded string

url = "https://cdn.openai.com/API/docs/audio/alloy.wav"

response = requests.get(url)

response.raise\_for\_status()

wav\_data = response.content

encoded\_string = base64.b64encode(wav\_data).decode('utf-8')

completion = client.chat.completions.create(

model="gpt-4o-audio-preview",

modalities=["text", "audio"],

audio={"voice": "alloy", "format": "wav"},

messages=[

{

"role": "user",

"content": [

{

"type": "text",

"text": "What is in this recording?"

},

{

"type": "input\_audio",

"input\_audio": {

"data": encoded\_string,

"format": "wav"

}

}

]

},

]

)

print(completion.choices[0].message)

```

```bash

curl "https://api.openai.com/v1/chat/completions" \

-H "Content-Type: application/json" \

-H "Authorization: Bearer $OPENAI\_API\_KEY" \

-d '{

"model": "gpt-4o-audio-preview",

"modalities": ["text", "audio"],

"audio": { "voice": "alloy", "format": "wav" },

"messages": [

{

"role": "user",

"content": [

{ "type": "text", "text": "What is in this recording?" },

{

"type": "input\_audio",

"input\_audio": {

"data": "<base64 bytes here>",

"format": "wav"

}

}

]

}

]

}'

```

Was this page useful?