Speech to text

Learn how to turn audio into text

Introduction

The Audio API provides two speech to text endpoints, `transcriptions` and `translations`, based on our state-of-the-art open source large-v2 Whisper model. They 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. We currently support multiple input and output file formats.

Transcribe audio — python

```python
from openai import OpenAI
client = OpenAI()

audio_file= open("/path/to/file/audio.mp3", "rb")
transcription = client.audio.transcriptions.create(
  model="whisper-1",
  file=audio_file
)
print(transcription.text)
```

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 — python

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

The API Reference includes the full list of available parameters.

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.

Translate audio — python

```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)
```

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 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 (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.

Timestamps

By default, the Whisper API will output a transcript of the provided audio in text. The `timestamp_granularities[]` parameter 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 — python

```python
from openai import OpenAI
client = OpenAI()

audio_file = open("speech.mp3", "rb")
transcript = client.audio.transcriptions.create(
  file=audio_file,
  model="whisper-1",
  response_format="verbose_json",
  timestamp_granularities=["word"]
)
```

```
11
12 print(transcript.words)
```

Longer inputs

By default, the Whisper 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 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 to improve the quality of the transcripts generated by the Whisper API. The model will try to match the style of the prompt, so it will be more likely to use capitalization and punctuation if the prompt does too. However, the current prompting system is much more limited than our other language models and only provides limited control over the generated audio. Here are some examples of how prompting can help in different scenarios:

1. Prompts can be very helpful for correcting specific words or acronyms that the model may misrecognize 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, you can prompt the model with the transcript of the preceding segment. This will make the transcript more accurate, as the model will use the relevant information from the previous audio. The model will only consider the final

224 tokens of the prompt and ignore anything earlier. For multilingual inputs, Whisper uses a custom tokenizer. For English only inputs, it uses the standard GPT-2 tokenizer which are both accessible through the open source Whisper Python package.

3. Sometimes the model might skip punctuation in the transcript. You can avoid this by using 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, you can 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.

Improving reliability

As we explored in the prompting section, one of the most common challenges faced when using Whisper is the model often does not recognize uncommon words or acronyms. To address this, we have highlighted different techniques which 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.

Since it wasn't trained using instruction-following techniques, Whisper operates more like a base GPT model. It's important to keep in mind that Whisper only considers the first 244 tokens of the prompt.

Prompt parameter — python

```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, OrbitalLin
)
print(transcription.text)
```

While it will increase reliability, this technique is limited to only 244 characters so your list of SKUs would need to be relatively small in order 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 — python

```python
system_prompt = "You are a helpful assistant for the company ZyntriQix. Your task is to

def generate_corrected_transcript(temperature, system_prompt, audio_file):
    response = client.chat.completions.create(
        model="gpt-4-turbo",
        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 can see that GPT-4 manages to correct many misspellings in the transcript. Due to its larger context window, this method might be more scalable than using Whisper's prompt parameter and is more reliable since GPT-4 can be instructed and guided in ways that aren't possible with Whisper given the lack of instruction following.
