Speech-to-text

MULTI-MODAL SYSTEMS WITH THE OPENAI API

James Chapman
Curriculum Manager, DataCamp

datacamp

Coming up...

Course goals

- OpenAI's audio models
- Text moderation
- Case study: Customer support chatbot

Recap...

```
from openai import OpenAI

# Create the OpenAI client
client = OpenAI(api_key="<OPENAI_API_TOKEN>")

# Create a request to the Chat Completions endpoint
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[{"role": "user",
               "content": "What is the OpenAI API?"}]
)
```

- **No API key required**—it's already set up for you

Recap...

```
# Extract the content from the response
print(response.choices[0].message.content)
```

```
The OpenAI API is a cloud-based service provided by OpenAI that allows developers
to integrate advanced AI models into their applications.
```

- OpenAI API goes beyond text

OpenAI's audio models

Speech-to-text capabilities:

- Transcribe audio
- Translate non-English audio
- Supports `mp3`, `mp4`, `mpeg`, `mpga`, `m4a`, `wav`, and `webm` (25 MB limit)

Use cases:

- Meeting transcripts
- Video captions
- Processing customer calls

Loading audio files

Example: transcribe `meeting_recording.mp3`

```
audio_file = open("meeting_recording.mp3", "rb")
```

If the file is located in a *different directory*

```
audio_file = open("path/to/file/meeting_recording.mp3", "rb")
```

Creating the transcription

- Audio endpoint

```
audio_file= open("meeting_recording.mp3", "rb")

response = client.audio.transcriptions.create(
    model="whisper-1",
    file=audio_file
)

print(response)
```

```
Transcription(text="Welcome everyone to the June product monthly. We'll get started in...)
```

[1] https://platform.openai.com/docs/guides/speech-to-text

The transcript

```
print(response.text)
```

```
Welcome everyone to the June product monthly. We'll get started in just a minute.
Alright, let's get started. Today's agenda will start with a spotlight from Chris
on the new mobile user onboarding flow, then we'll review how we're tracking on
our quarterly targets, and finally, we'll finish with another spotlight from Katie
who will discuss the upcoming branding updates...
```

Transcribing non-English audio

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.

Transcribing workflow:

1. `open()` audio file
2. Send a transcription request
3. Extract the text

Creating translations

```
audio_file = open("non_english_audio.m4a", "rb")

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

print(response.text)
```

```
The search volume for keywords like A I has increased rapidly since the launch of Cha GTP.
```

Transcription performance

- Performance can vary wildly, depending on:
  - Audio quality
  - Audio language
  - Model's knowledge of the subject matter

Let's practice!

MULTI-MODAL SYSTEMS WITH THE OPENAI API

Text-to-speech (TTS)

MULTI-MODAL SYSTEMS WITH THE OPENAI API

James Chapman
Curriculum Manager, DataCamp

datacamp

Text-to-speech

- Internet browsers, mobile apps, accessibility
- Text → realistic human speech
- Improve accessibility

Text-to-speech with OpenAI

- **Audio** endpoint → `.speech.create()`

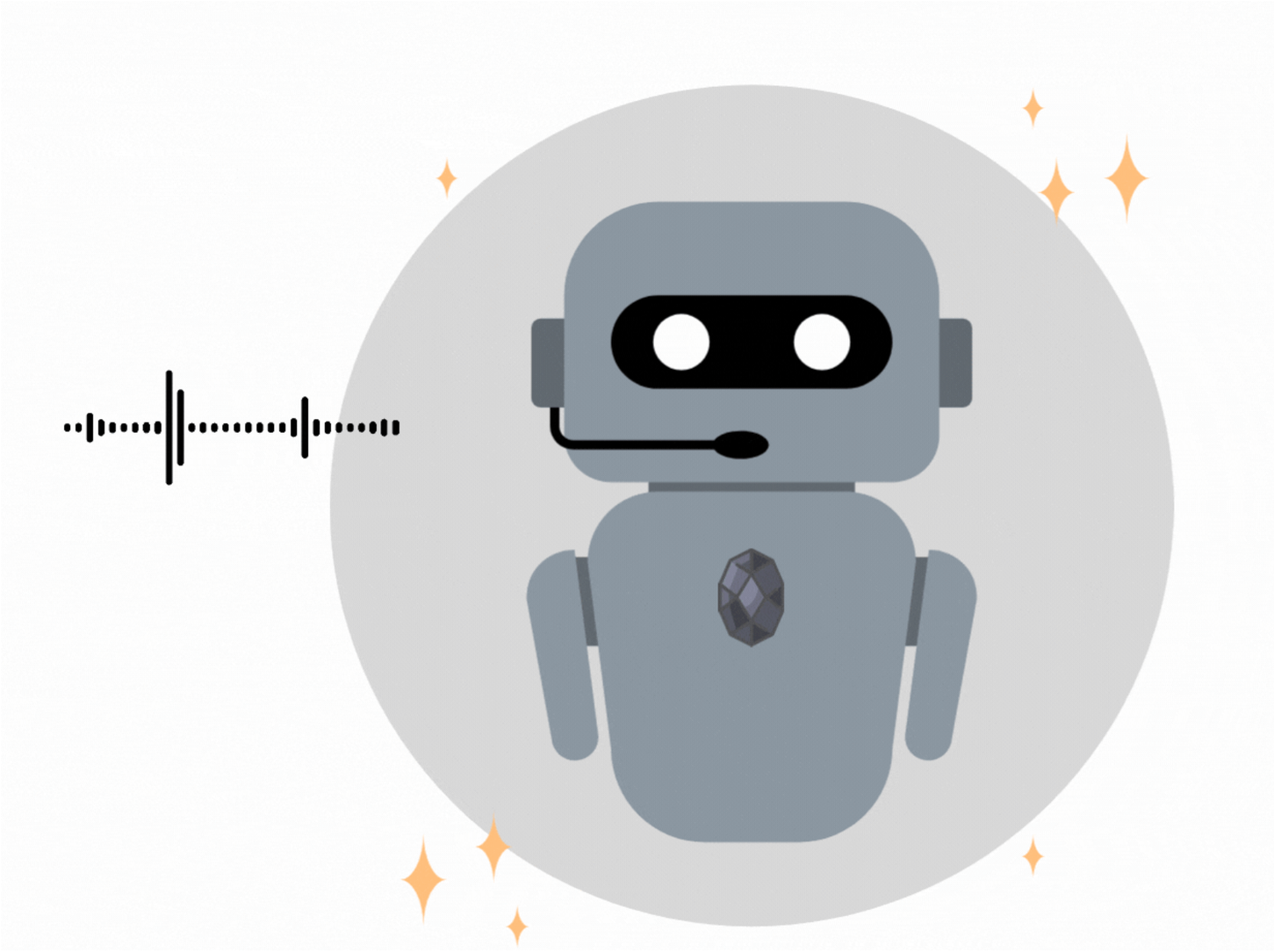
```
response = client.audio.speech.create(
    model="gpt-4o-mini-tts",
    voice="onyx",
    input="Creating human-like speech is now possible with just a few lines of code.
    Pretty neat, right?"
)

response.stream_to_file("output.mp3")
```

- `response_format`: `"mp3"`, `"opus"`, `"aac"`, `"flac"`, `"wav"`, and `"pcm"`

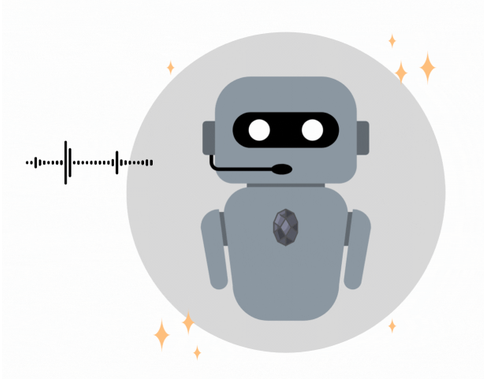
[1] https://www.openai.fm/

Onyx



OpenAI TTS

- 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.

Let's practice!

MULTI-MODAL SYSTEMS WITH THE OPENAI API

Content moderation

MULTI-MODAL SYSTEMS WITH THE OPENAI API

James Chapman
Curriculum Manager, DataCamp

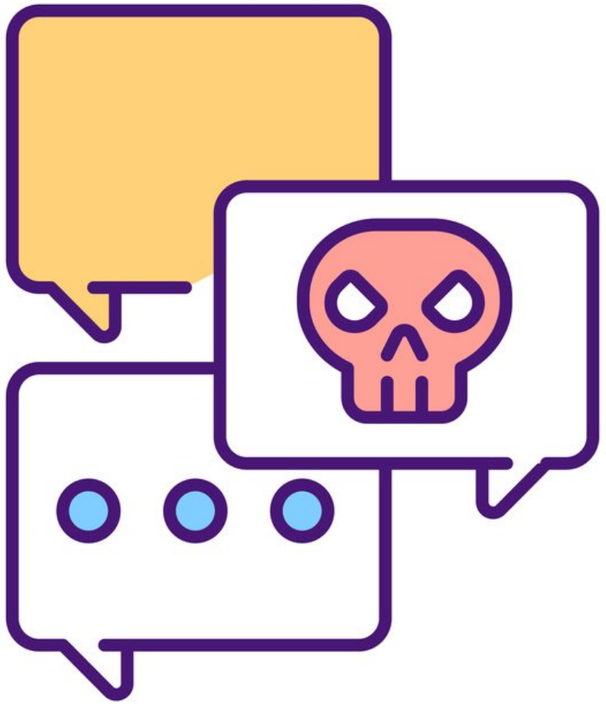
datacamp

Moderation

- Identifying inappropriate content

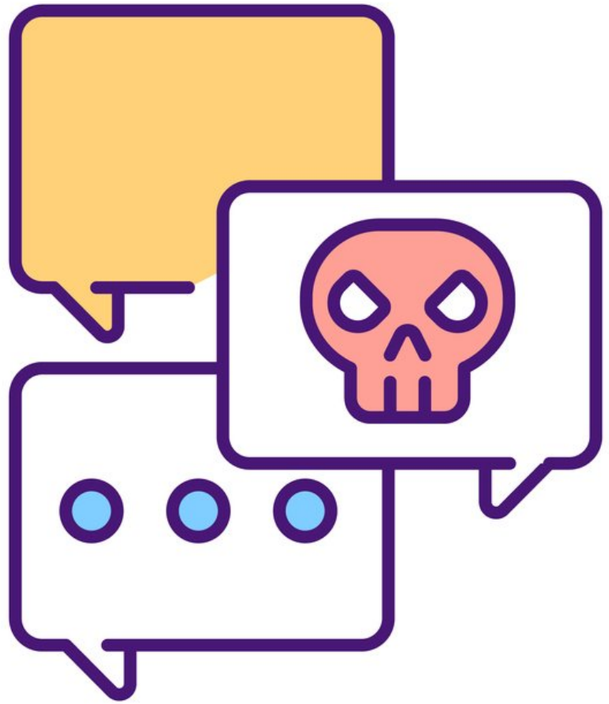
Traditionally,

- Moderators flag content *by-hand*
  - Time-consuming
- **Keyword pattern matching**
  - Lacks *nuance* and understanding of *context*



Violation categories

- Identify violations of terms or use
- Differentiate violation type by category
  - Violence
  - Hate speech



[1] https://openai.com/policies/usage-policies [2] https://platform.openai.com/docs/guides/moderation/overview

Creating a moderations request

```
from openai import OpenAI

client = OpenAI(api_key="ENTER API KEY")

response = client.moderations.create(
  model="text-moderation-latest",
  input="I could kill for a hamburger."
)
```

Interpreting the results

- `categories`
  - `true` / `false` indicator of category violation
- `category_scores`
  - Confidence of a violation
- `flagged`
  - `true` / `false` indicator of a violation

```
print(response.model_dump())
```

```
{'id': 'modr-8S80XeaVqvs4mmbufDBP9gTAmEGXP',
 'model': 'text-moderation-006',
 'results': [{'categories': {'harassment': False,
                             'harassment_threatening': False,
                             'hate': False,
                             'hate_threatening': False,
                             ...},
              'category_scores': {'harassment': 2.775943e-05,
                                  'harassment_threatening': 1.3526056e-06,
                                  'hate': 2.733528e-07,
                                  'hate_threatening': 4.930576e-08,
                                  ...},
              'flagged': False}]}
```

Interpreting the category scores

```
print(response.results[0].category_scores)
```

```
CategoryScores(harassment=2.775944e-05,
               harassment_threatening=1.3526056e-06,
               hate=2.733528e-07,
               hate_threatening=4.930576e-08,
               ...,
               violence=0.050085403025150 3,
               ...)
```

- Larger numbers → greater certainty of violation
- Numbers $\neq$ probabilities

Interpreting the category scores

```
print(response.results[0].category_scores)
```

```
CategoryScores(harassment=2.775944e-05,
               harassment_threatening=1.3526056e-06,
               hate=2.733528e-07,
               hate_threatening=4.930576e-08,
               ...,
               violence=0.050085403025150 3,
               ...)
```

- Larger numbers → greater certainty of violation
- Numbers $\neq$ probabilities

Considerations for implementing moderation

```
CategoryScores(harassment=2.775943e-05,
               harassment_threatening=1.3526056e-06,
               hate=2.733528e-07,
               hate_threatening=4.930576e-08,
               ...,
               violence=0.050085403025150 3,
               ...)
```

- Tune thresholds for each use case
- Stricter thresholds may result in fewer *false negatives*
- More lenient thresholds may result in fewer *false positives*

Let's practice!

MULTI-MODAL SYSTEMS WITH THE OPENAI API