

# Text moderation

WORKING WITH THE OPENAI API



**James Chapman**

Curriculum Manager, DataCamp

# Going beyond text completions...

Completions → generate *new text output* using text prompt

Beyond completions:

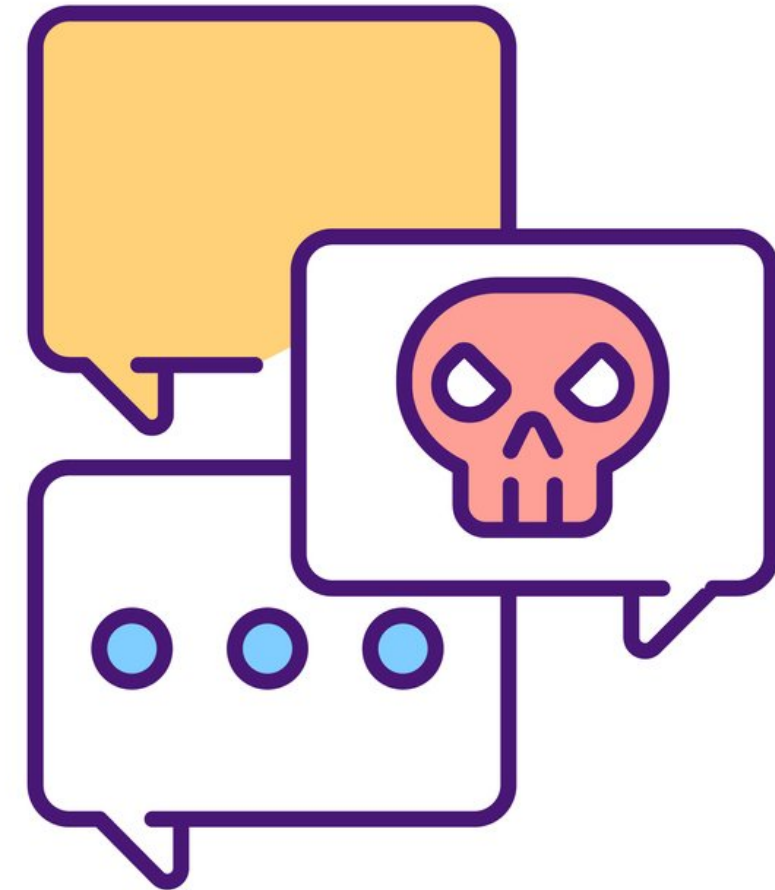
- Text moderation
- Audio transcription and translation
- Combining models together

# Text moderation

- Identifying inappropriate content

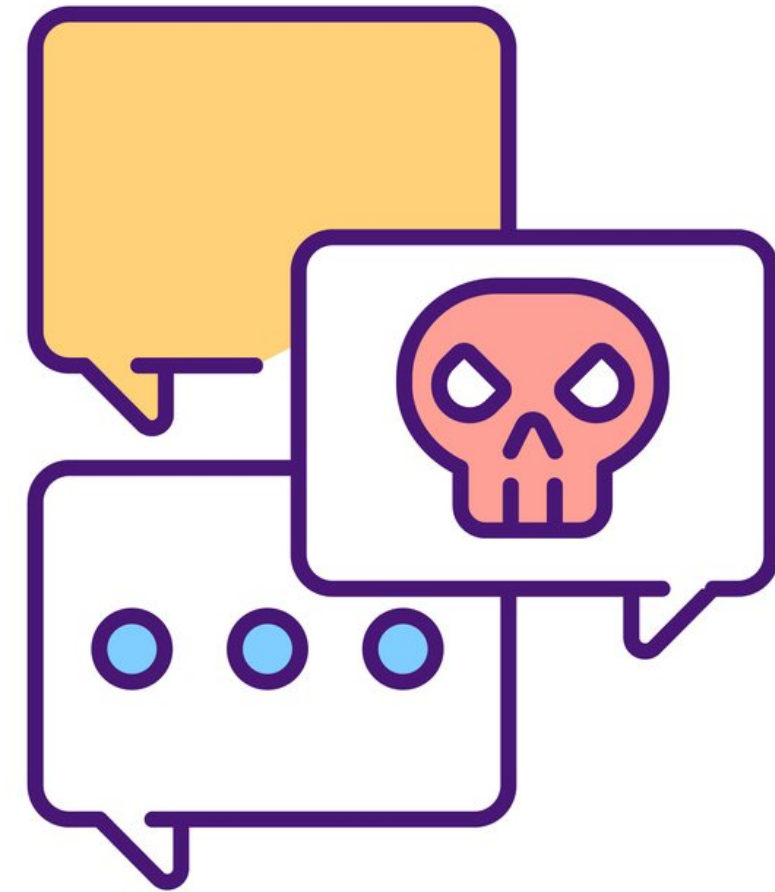
Traditionally,

- 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



<sup>1</sup> <https://openai.com/policies/usage-policies> <sup>2</sup> <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': 2.775940447463654e-05,  
                         'harassment_threatening': 1.3526056363843963e-06,  
                         'hate': 2.733528674525587e-07,  
                         'hate_threatening': 4.930571506633896e-08,  
                         ...},  
     'flagged': False}]  
}
```

# Interpreting the category scores

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

```
CategoryScores(harassment=2.775940447463654e-05,  
               harassment_threatening=1.3526056363843963e-06,  
               hate=2.733528674525587e-07,  
               hate_threatening=4.930571506633896e-08,  
               ...,  
               violence=0.0500854030251503,  
               ...)
```

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

# Considerations for implementing moderation

```
CategoryScores(harassment=2.775940447463654e-05,  
               harassment_threatening=1.3526056363843963e-06,  
               hate=2.733528674525587e-07,  
               hate_threatening=4.930571506633896e-08,  
               ...,  
               violence=0.0500854030251503,  
               ...)
```

- Determine appropriate 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!**  
WORKING WITH THE OPENAI API

# Speech-to-Text Transcription with Whisper

WORKING WITH THE OPENAI API



**James Chapman**  
Curriculum Manager, DataCamp

# OpenAI's Whisper

## Speech-to-text capabilities:

- Transcribe audio
- Translate and transcribe audio into *English*
- Supports `mp3` , `mp4` , `mpeg` , `mpga` , `m4a` , `wav` , and `webm` (25 MB limit)
- Meeting transcripts
- Video captions



# Loading audio files

**Example:** transcribe `meeting_recording_20230602.mp3`

```
audio_file = open("meeting_recording_20230602.mp3", "rb")
```

If the file is located in a *different directory*

```
audio_file = open("path/to/file/meeting_recording_20230602.mp3", "rb")
```

# Making a request

- Audio endpoint

```
audio_file= open("meeting_recording_20230602.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...)
```

# 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...
```

- Don't use sensitive or confidential recordings

# Transcribing non-English languages

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.

1. `open()` audio file
2. Make a transcriptions request to the Audio endpoint
3. Extract text from the response

**Let's practice!**  
WORKING WITH THE OPENAI API



# Speech Translation with Whisper

WORKING WITH THE OPENAI API



**James Chapman**  
Curriculum Manager, DataCamp

# Whisper's translation capabilities

- Translate and transcribe audio
- Currently limited to *English* transcripts
- Supports `mp3` , `mp4` , `mpeg` , `mpga` , `m4a` , `wav` , and `webm` (25 MB limit)



# Translating audio

```
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 ChatGPT.

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

# Bringing prompts into the mix

- Can provide `prompt` to the model (optional)
- Improve response quality by:
  - Providing an example of desired style
  - Provide context on transcript context

**Example:** Retaining filler words

```
prompt="Ok, ummm... this is what we should do, like, to uhhh... increase revenue."
```

**Example:** Provide context

```
prompt="A discussion on how to increase revenue."
```

# Adding in a prompt

```
audio_file = open("non_english_audio.m4a", "rb")
prompt = "The transcript is about AI trends and ChatGPT."

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

print(response.text)
```

The search volume for keywords like AI has increased rapidly since the launch of ChatGPT.

**Let's practice!**  
WORKING WITH THE OPENAI API

# Combining models

WORKING WITH THE OPENAI API



**James Chapman**

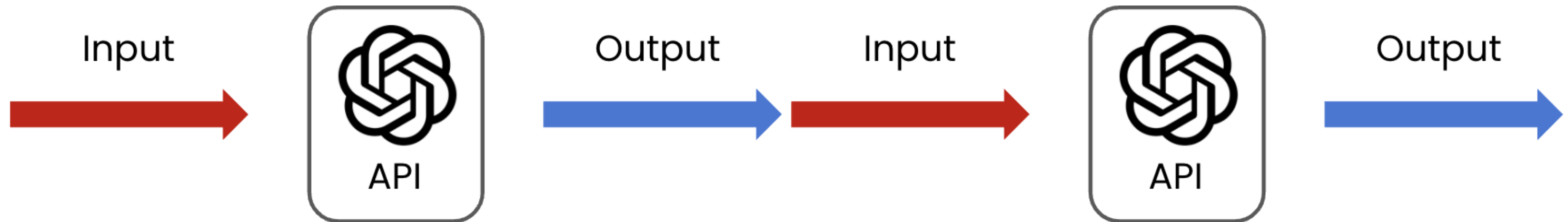
Curriculum Manager, DataCamp

# Combining models





# Combining models



- **Chaining:** Feeding output from one model into another
- Can use the *same model* multiple times
  - **Example:** validating original response
- Or *different models*
  - **Example:** summarizing meeting recordings

# Example: Extracting meeting attendees

```
audio_file = open("meeting_recording_20230608.mp4", "rb")

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

transcript = audio_response.text
prompt = "Extract the attendee names from the start of this meeting transcript: " + transcript

chat_response = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "user", "content": prompt}
    ]
)

print(chat_response.choices[0].message.content)
```

# Example: Extracting meeting attendees

The meeting attendees were Otis, Paul, Elaine, Nicola, Alan, and Imran.

Note:

- **No guarantees** on model performance
- Ensure that applications are **well-tested**
- Usage should be restricted to **non-sensitive data**

**Let's practice!**  
WORKING WITH THE OPENAI API

# Congratulations!

WORKING WITH THE OPENAI API

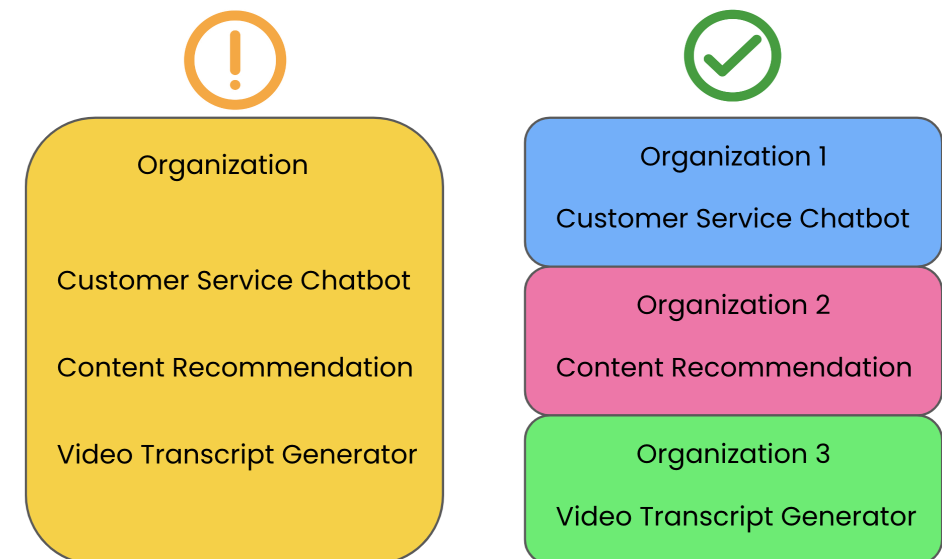
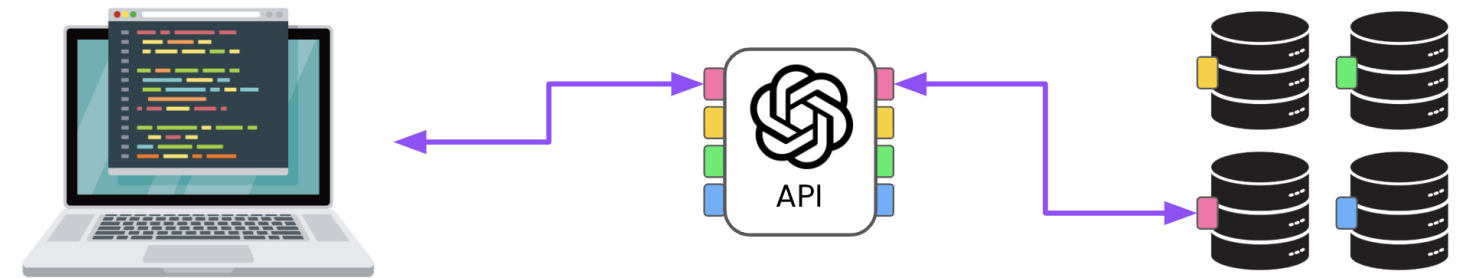


**James Chapman**

Curriculum Manager, DataCamp

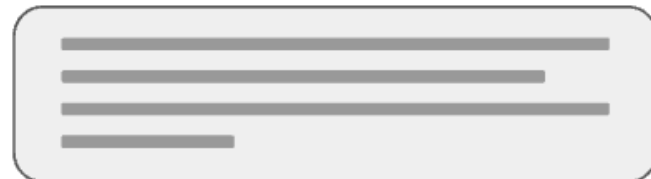
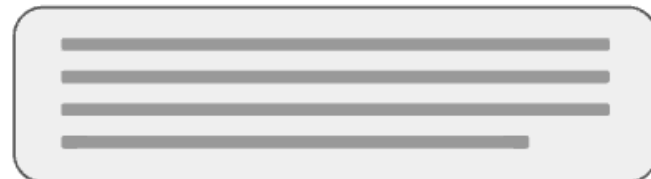
# Chapter 1

- What the OpenAI API is used for
- How to create an API key
- How to create requests to the Completions endpoint
- Creating and managing organizations



# Chapter 2

```
response = client.completions.create(  
    model="gpt-3.5-turbo-instruct",  
    prompt="When was OpenAI founded?",  
    max_tokens=20,  
    temperature=0.5  
)
```



- Solved different tasks with text completions, including:
  - Q&A
  - Text transformation
  - Content generation
  - Sentiment analysis
  - Categorization
- `max_tokens` and `temperature`
- Multi-turn conversation with Chat Completions

# Chapter 3

## Moderation

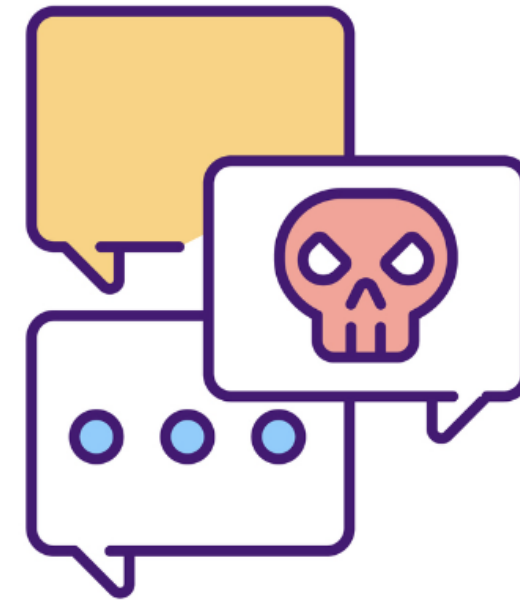
- Detect inappropriate content

## Audio

- Translate
- Transcribe

## Chaining

- Automating complex tasks





# What next?

AI application development:

- [ChatGPT Prompt Engineering for Developers](#)
- [Introduction to Embeddings with the OpenAI API](#)
- Developing LLM Applications with LangChain (coming soon!)
- Working with Hugging Face (coming soon!)

Apply your learning in projects:

- [Planning a Trip to Paris with the OpenAI API](#)
- [Enriching Stock Market Data using the OpenAI API](#)

**Let's practice!**  
WORKING WITH THE OPENAI API