Structuring an API call

DEVELOPING AI SYSTEMS WITH THE OPENAI API

Francesca Donadoni
Curriculum Manager, DataCamp

datacamp

Progress Snapshot

OpenAI Library

Progress Snapshot

OpenAI Library

API Call

Progress Snapshot

OpenAI Library

API Call

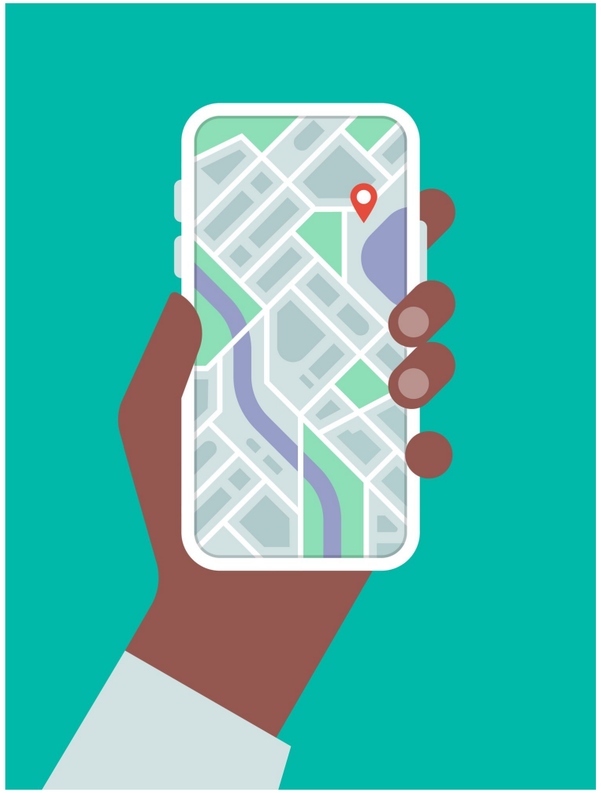
Response Message

Progress Snapshot

```
from openai import OpenAI
client = OpenAI(api_key="ENTER YOUR KEY HERE")
response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[
   {"role": "user",
   "content": "Who developed ChatGPT?"}
 ]
)
print(response.choices[0].message.content)
```

```
ChatGPT was developed by OpenAI, an artificial intelligence research lab.
```

Challenges of a production environment



Challenges of a production environment

- Error Handling
  - Displaying user-friendly error messages
  - Alternatives for when the service is unavailable
- Moderation and Safety
  - Control unwanted inputs
  - Minimizing the risk of data leaks
- Testing and Validation
  - Checking for responses that are out of topic
  - Testing for inconsistent behavior
- Communication with External Systems
  - Calling external functions and APIs
  - Optimizing response times

Components of an OpenAI API request

```
from openai import OpenAI

client = OpenAI(api_key="ENTER YOUR KEY HERE")

response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[
   {"role": "user",
   "content": "Please write down five trees with their scientific names in json format."}
 ],
  response_format={"type": "json_object"}
)
```

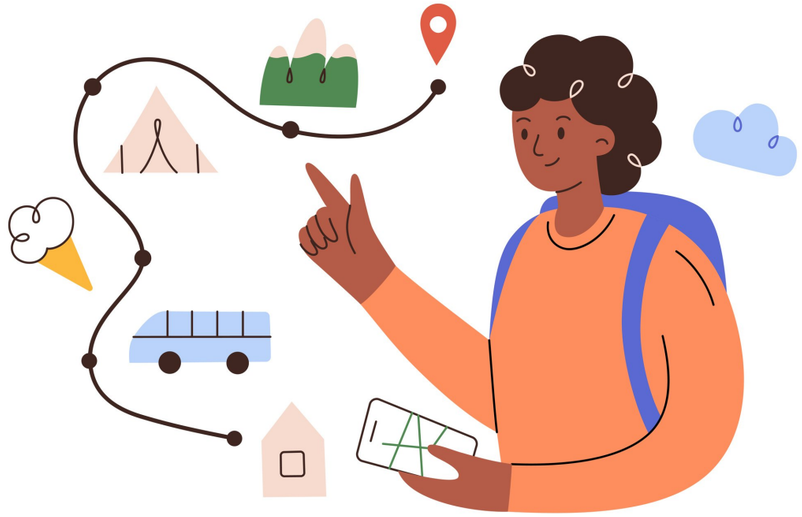
Components of an OpenAI API response

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

```
{
  "trees": [
    {"commonName": "Oak", "scientificName": "Quercus"},
    {"commonName": "Maple", "scientificName": "Acer"},
    {"commonName": "Pine", "scientificName": "Pinus"},
    {"commonName": "Birch", "scientificName": "Betula"},
    {"commonName": "Willow", "scientificName": "Salix"}
  ]
}
```

What's next

- Integration in production
- Calling external functions
- Best practices



Let's practice!

DEVELOPING AI SYSTEMS WITH THE OPENAI API

Handling errors

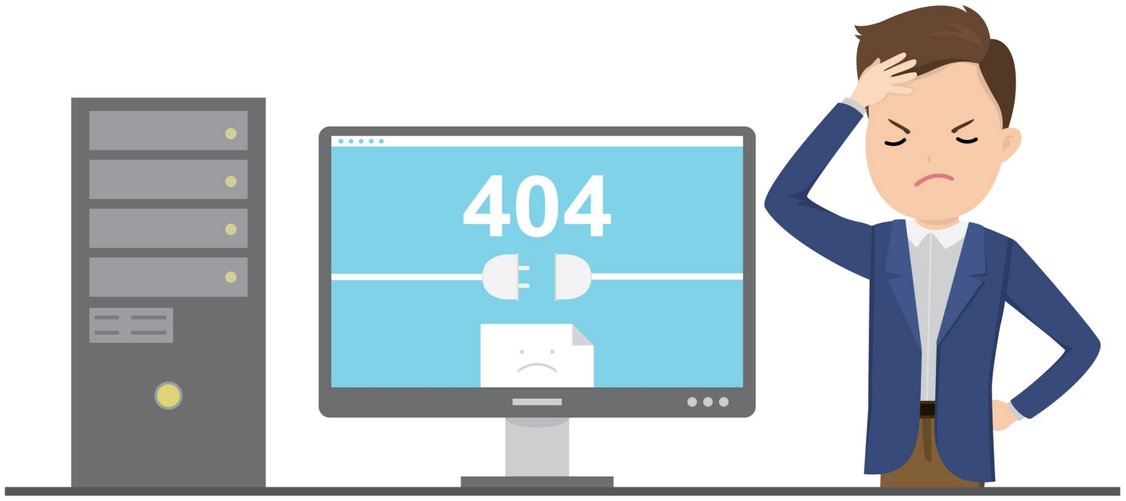
DEVELOPING AI SYSTEMS WITH THE OPENAI API

Francesca Donadoni
Curriculum Manager, DataCamp

datacamp

Errors in AI applications

- Simplifying the user experience
- Eliminating barriers



Errors in the OpenAI API library

```
response = client.chat.completions.create(
 model="text-davinci-001",
 messages=[
   {
    "role": "user",
    "content": "List two data science professions with related skills in json format."
   }
 ],
 response_format={"type": "json_object"}
)
```

```
NotFoundError: Error code: 404 - {'error': {'message': 'The model `text-davinci-001`
has been deprecated, learn more here: https://platform.openai.com/docs/deprecations'}}
```

Connection errors

- Generally due to connection issues on either the user's or the service's side
- Examples: `InternalServerError`, `APIConnectionError`, `APITimeoutError`
- Potential solution:
  - Checking your connection configuration
  - Reaching out to support if that fails

Resource limits errors

- Generally due limits on the frequency of requests or the amount of text passed

- Examples: `ConflictError` , `RateLimitError`

- Potential solution:
  - Checking limit restrictions
  - Ensure requests are within limits

Authentication errors

```
client = OpenAI(api_key="This is an Invalid Key")
response = client.chat.completions.create(
 model="gpt-4o-mini",
 messages=[
   {
    "role": "user",
    "content": "List two data science professions with related skills in json format."
   }
 ],
 response_format={"type": "json_object"}
)
```

```
AuthenticationError: Error code: 401 - {'error': {'message': 'Incorrect API key provided:
ThisIsNo*AKey. You can find your API key at https://platform.openai.com/account/api-keys.'}}
```

Bad request errors

```
response = client.chat.completions.create(
 model="gpt-4o-mini",
 messages=[
   {"role": "This is not a Valid Role", "content": "List two data science
   professions with related skills in json format."}
 ],
 response_format={"type": "json_object"}
)
```

```
BadRequestError: Error code: 400 - {'error': {'message': "'NotARole' is not one of
['system', 'assistant', 'user', 'function'] - 'messages.0.role'",
'type': 'invalid_request_error', 'param': None, 'code': None}}
```

Handling exceptions

```
try:
    response = client.chat.completions.create(
     model="gpt-4o-mini",
     messages=[{"role": "user",
      "content": "List five data science professions."}])
except openai.AuthenticationError as e:
  print(f"OpenAI API failed to authenticate: {e}")
  pass
except openai.RateLimitError as e:
  print(f"OpenAI API request exceeded rate limit: {e}")
  pass
except Exception as e:
  print(f"Unable to generate a response. Exception: {e}")
  pass
```

Let's practice!

DEVELOPING AI SYSTEMS WITH THE OPENAI API

Batching

DEVELOPING AI SYSTEMS WITH THE OPENAI API

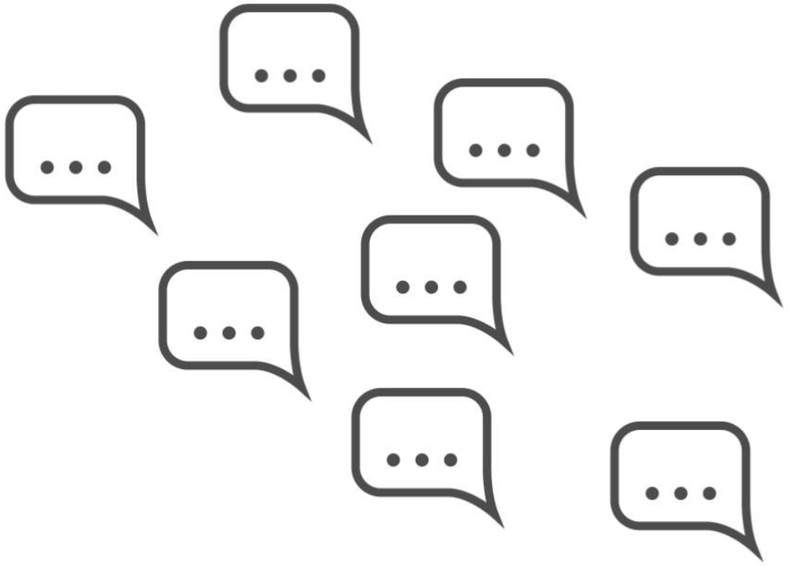
Francesca Donadoni
Curriculum Manager, DataCamp

datacamp

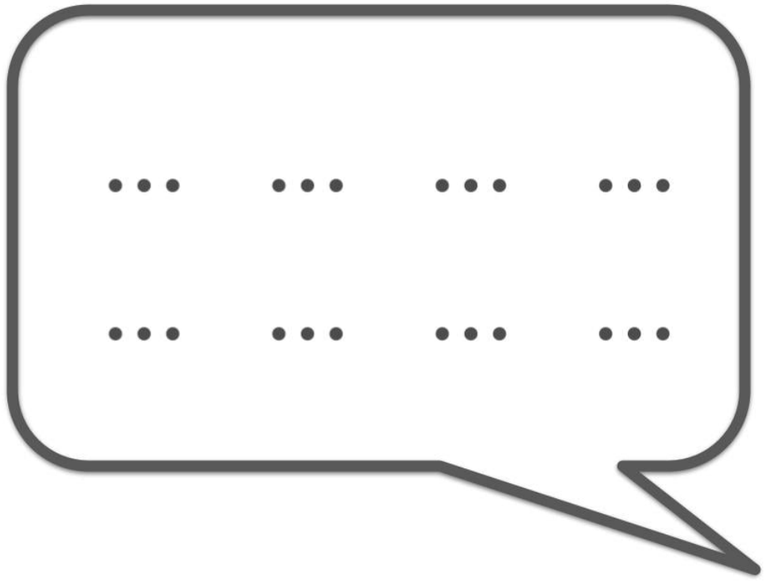
What are rate limits

How rate limits occur

- Too many requests



- Too much text in the request



Avoiding rate limits

- Retry
  - Short wait between requests
- Batching
  - Processing multiple messages in one request
- Reducing tokens
  - Quantifying and cutting down the number of tokens

Retrying

```
from tenacity import (
    retry,
    stop_after_attempt,
    wait_random_exponential
)
@retry(wait=wait_random_exponential(min=1, max=60), stop=stop_after_attempt(6))
```

Retrying

```
@retry(wait=wait_random_exponential(min=1, max=60), stop=stop_after_attempt(6))
def get_response(model, message):
    response = client.chat.completions.create(
      model=model,
      messages=[message],
      response_format={"type": "json_object"}
    )
    return response.choices[0].message.content
```

Batching

```
countries = ["United States", "Ireland", "India"]

message=[
    {
    "role": "system",
    "content": """You are given a series of countries and are asked to return the
    country and capital city. Provide each of the questions with an answer in the
    response as separate content.""",
    }]

[message.append({"role": "user", "content": i }) for i in countries]
```

Batching

```
response = client.chat.completions.create(
      model="gpt-4o-mini",
      messages=message
    )

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

```
United States: Washington D.C.
Ireland: Dublin
India: New Delhi
```

Reducing tokens

```
import tiktoken

encoding = tiktoken.encoding_for_model("gpt-4o-mini")
prompt = "Tokens can be full words, or groups of characters commonly grouped
          together: tokenization."

num_tokens = len(encoding.encode(prompt))
print("Number of tokens in prompt:", num_tokens)
```

```
Number of tokens in prompt: 17
```

Let's practice!

DEVELOPING AI SYSTEMS WITH THE OPENAI API