Exploring GPT-3

An unofficial first look at the general-purpose language processing API from OpenAI

Steve Tingiris

Foreword by Bret Kinsella (Founder and CEO of Voicebot.ai)

Preface

What if this audiobook was written by artificial intelligence? Would you read it? I hope so because parts of it were. Yes, GPT-3 was used to create parts of this audiobook. It's a bit meta I know, a audiobook about GPT-3 written by GPT-3. But creating content is one of the many great uses for GPT-3. So why not? Also, for me, content generation was the use case that most piqued my interest. I wondered if GPT-3 could be used in a product I was working on to automate the generation of technical learning material.

You probably also have a specific reason why you're interested in GPT-3. Perhaps it's intellectual curiosity. Or maybe you have an idea that you think GPT-3 can enable. You've likely seen online demos of GPT-3 generating content, writing code, penning poetry, or something else, and you're wondering if GPT-3 could be used for an idea you have. If so, this audiobook was written specifically for you.

My goal for this audiobook is to provide a practical resource to help you get started with GPT-3, as quickly as possible, without any required technical background. That said, as I write this, GPT-3 is still in private beta. So, everyone is learning as they go. But the one thing I've learned for sure is that the possible applications for GPT-3 are vast and there is no way to know all of what's possible, let alone get it into a audiobook. So, I hope this audiobook makes getting started easy, but I also hope it's just the beginning of your journey *Exploring GPT-3*

Who this audiobook is for

This audiobook is for anyone with an interest in NLP or learning GPT-3 — with or without a technical background. Developers, product managers, entrepreneurs, and hobbyists who want to learn about NLP, AI, and GPT-3 will find this audiobook useful. Basic computer skills are all you need to get the most out of the audiobook. While experience with a modern programming language is helpful, it's not required. The code examples provided are beginner friendly and easy to follow, even if you're brand new to writing code.

What this audiobook covers

Chapter 1, *Introducing GPT-3 and the OpenAI API*, is a high-level introduction to GPT-3 and the OpenAI API.

Chapter 2, *GPT-3 Applications and Use Cases*, is an overview of core GPT-3 use cases: text generation, classification, and semantic search.

Chapter 3, *Working with the OpenAI Playground*, is a semi-deep dive into the OpenAI Playground and the developer portal.

Chapter 4, *Working with the OpenAI API*, is an introduction to calling the OpenAI API using Postman.

Chapter 5, *Calling the OpenAI API in Code*, is an introduction to using the OpenAI API with both Node.js/JavaScript and Python.

Chapter 6, *Content Filtering*, explains how to implement content filtering.

Chapter 7, *Generating and Transforming Text*, contains code and prompt examples for generating and transforming text.

Chapter 8, *Classifying and Categorizing Text*, takes a closer look at text classification and the OpenAI API Classification endpoint.

Chapter 9, *Building a GPT-3 Powered Question-Answering App*, explains how to build a functional GPT-3 powered web knowledge base.

Chapter 10, *Going Live with OpenAI-Powered Apps*, explains the OpenAI application review and approval process and discusses getting ready for a review.

To get the most out of this audiobook

All of the code examples in this audiobook were written using a web-based **Integrated Development Environment** (or **IDE**) from replit.com. A free replit.com account is sufficient to follow the examples. To use replit.com, all that is required is a modern web browser and a replit.com account. The code has also been tested on macOS using Visual Studio Code, although it should work with any code editor and properly configured operating system. Code examples are provided in both Node.js/JavaScript and Python. For Node.js, version 12.16.1 is used and for Python, version 3.8.2 is used.

All of the code examples will require an OpenAI API Key and access to the OpenAI API. You can request access to the OpenAI API by visiting open AI.com/API.

Download the example code files

You can download the example code files for this audiobook from GitHub at github.com/PacktPublishing/Exploring-GPT-3. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at https://github.com/PacktPublishing/. Check them out!

Chapter 1

Technical requirements

This chapter requires you to have access to the **OpenAI** **Application Programming Interface** (**API**). You can register for API access by visiting https://openai.com/.

Figures

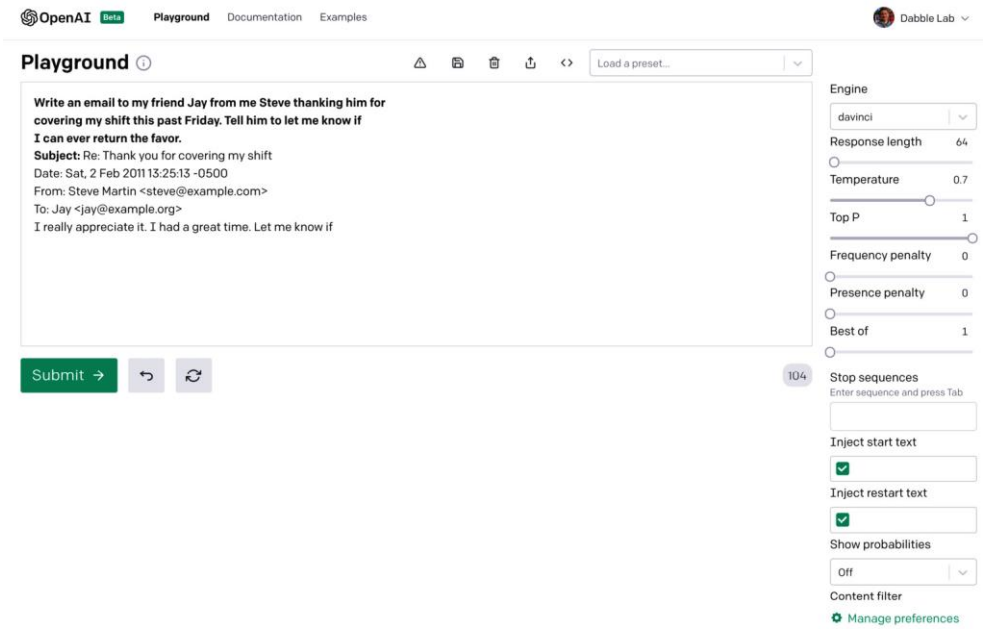

Figure 1.1 – Zero-shot prompt example

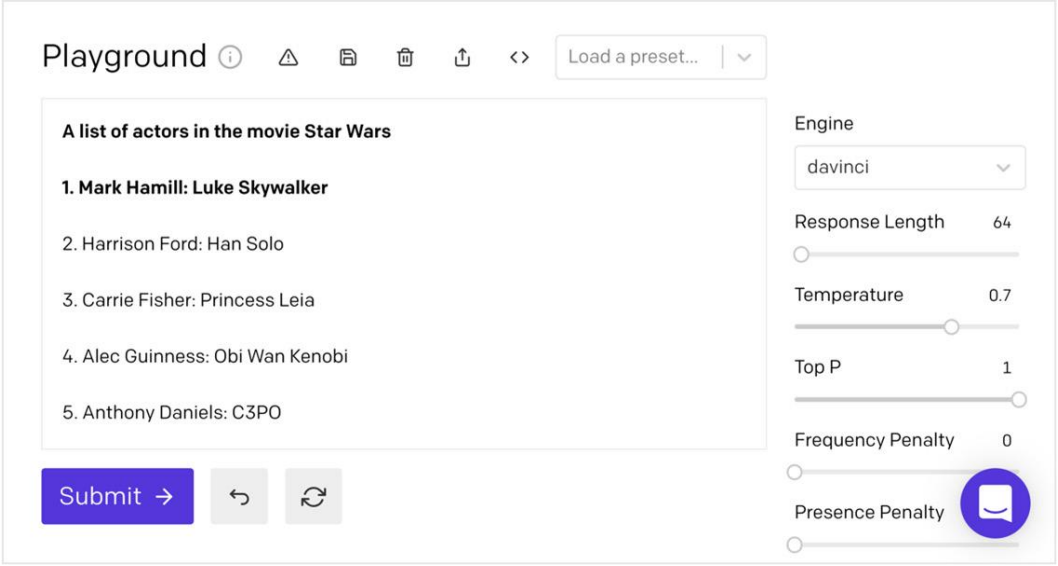

Figure 1.2 – One-shot prompt example

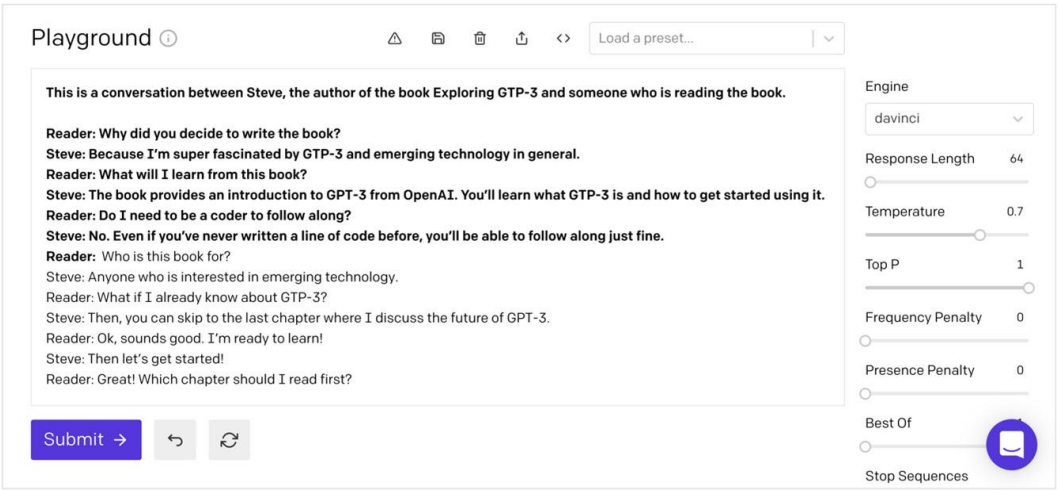

Figure 1.3 – Few-shot prompt example

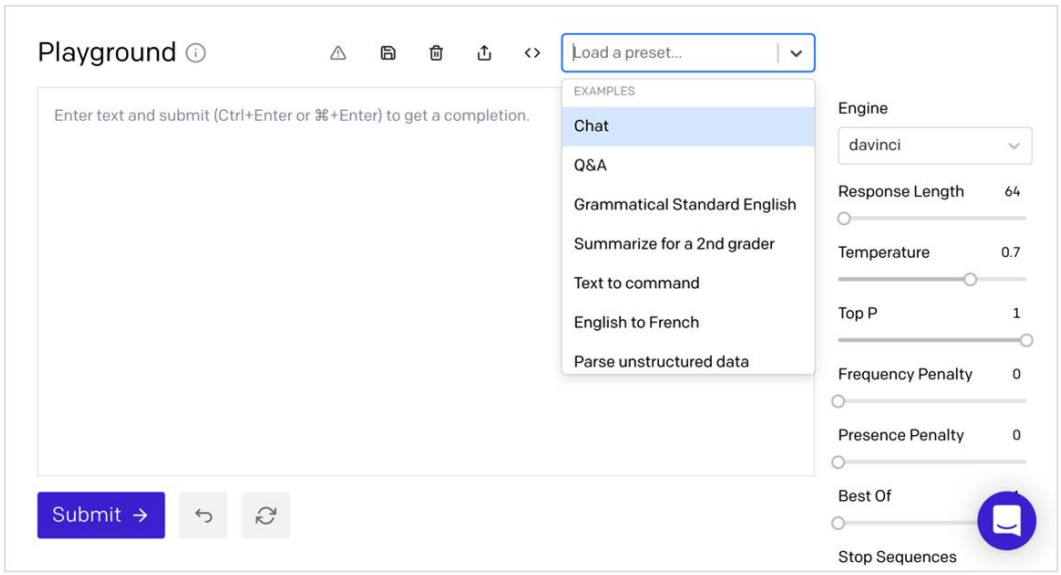

Figure 1.4 – Presets

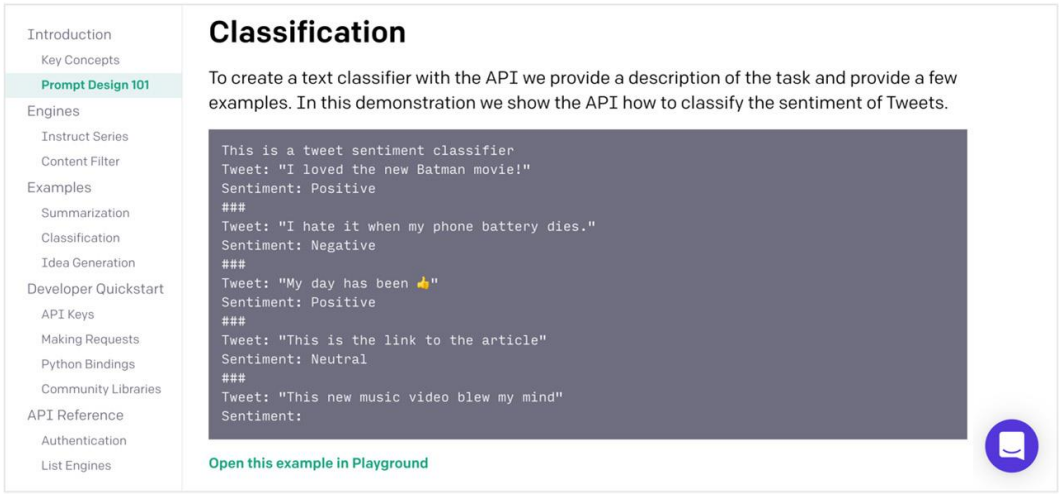

Figure 1.5 – OpenAI documentation provides prompt examples

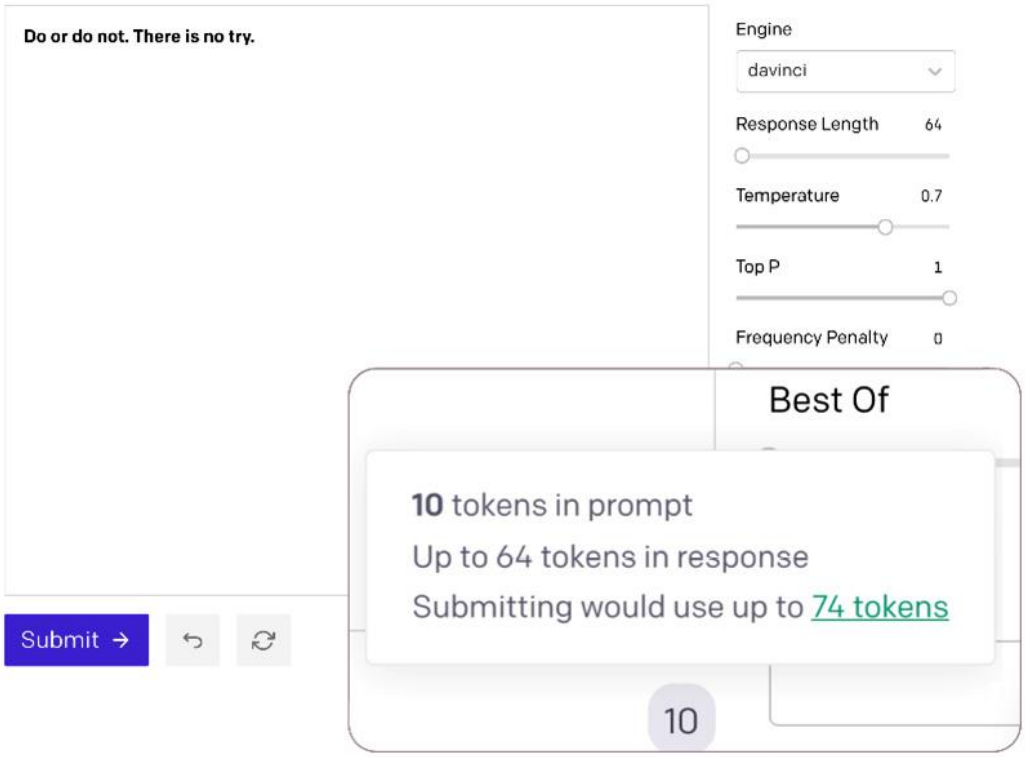

Figure 1.6 – Token count

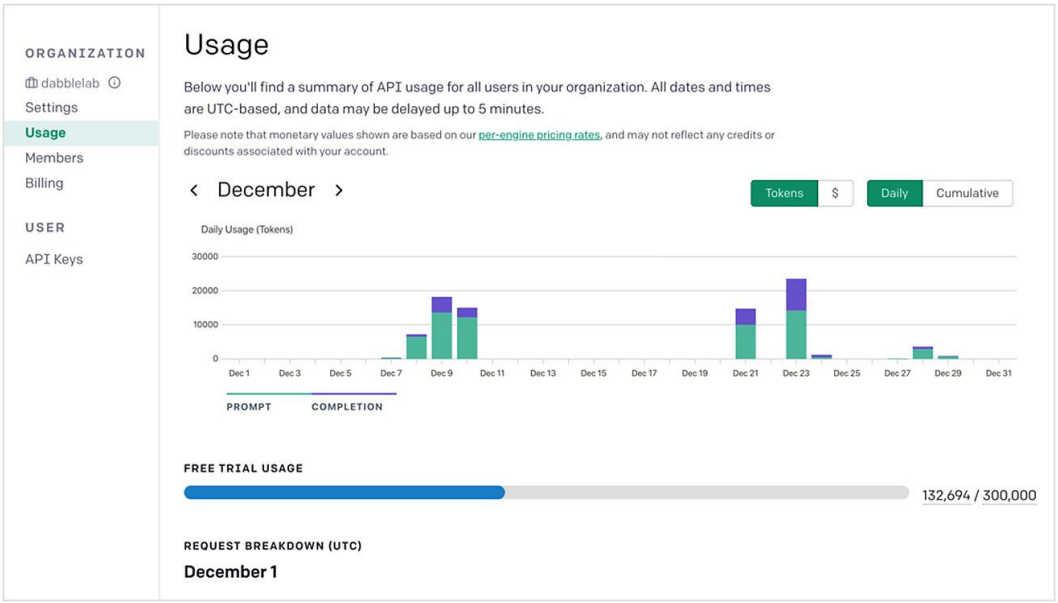

Figure 1.7 – Usage statistics

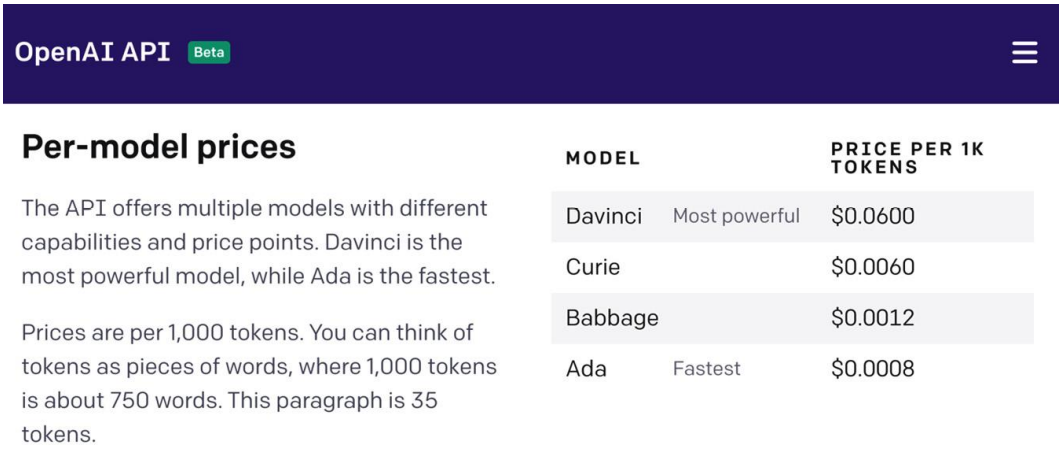

Figure 1.8 – Model pricing

Chapter 2

Technical requirements

This chapter requires you to have access to the OpenAI API. You can register for API access by visiting https://openai.com.

Figures

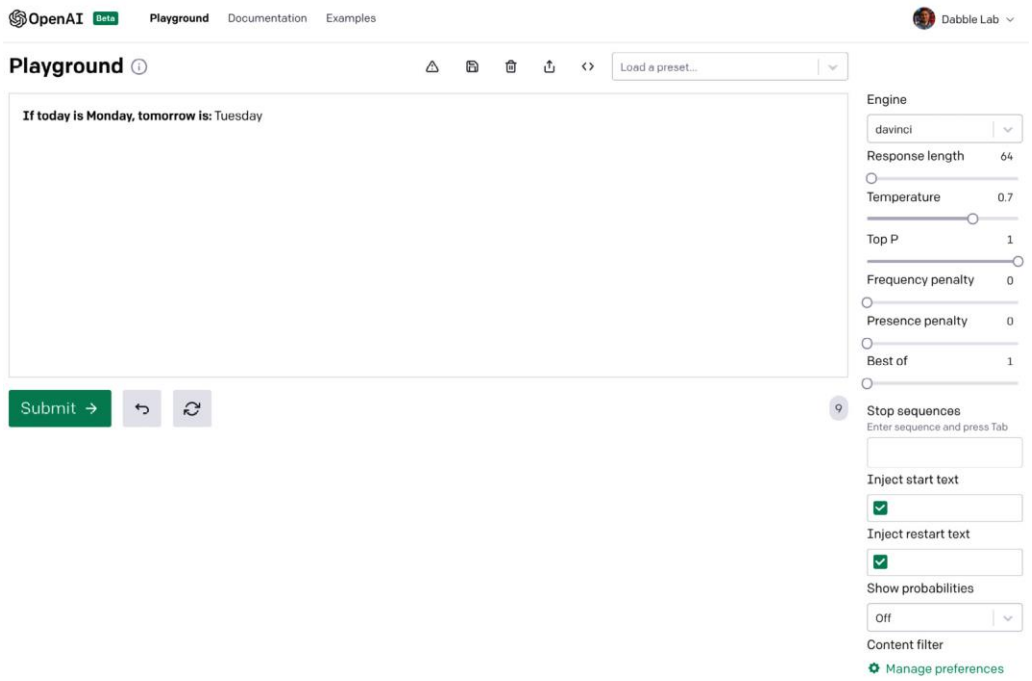

Figure 2.1 – Playground window

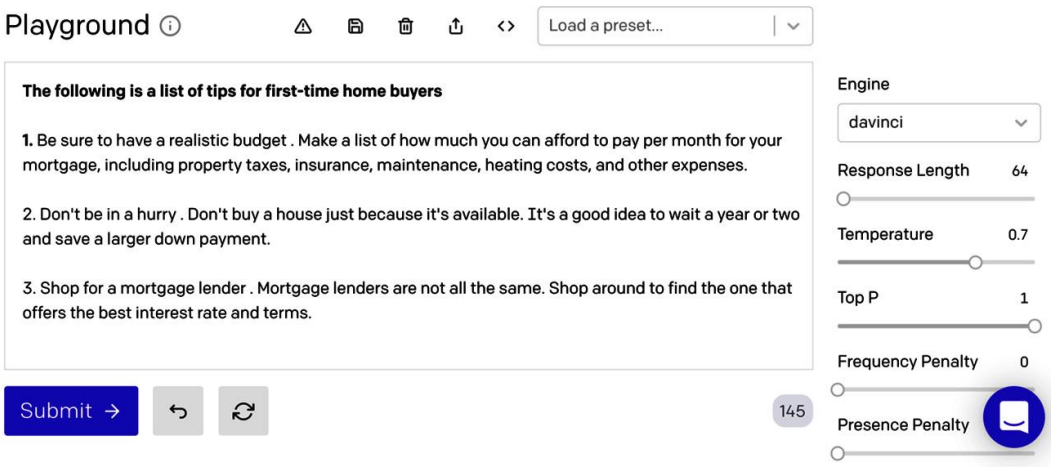

Figure 2.2 – Text generation example – tips for first-time home buyers

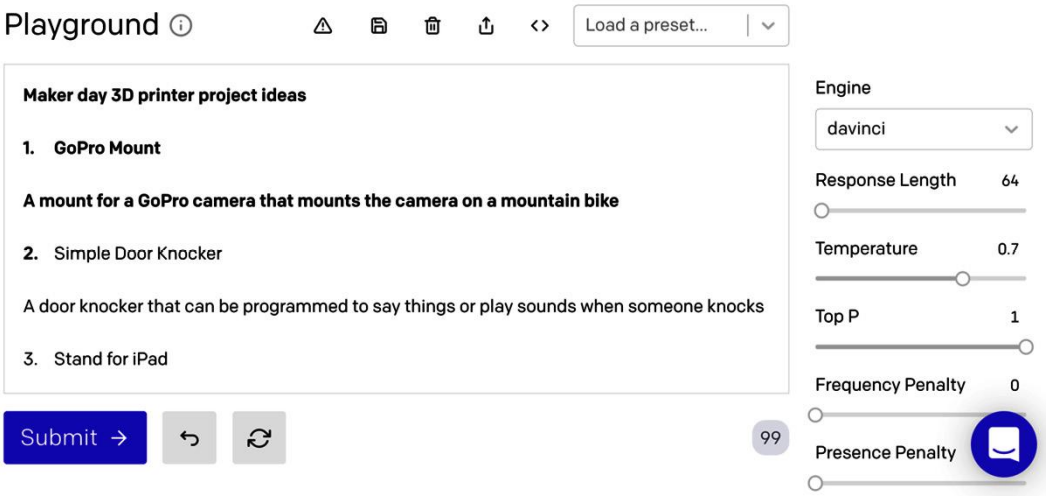

Figure 2.3 – Text generation example – 3D print project ideas

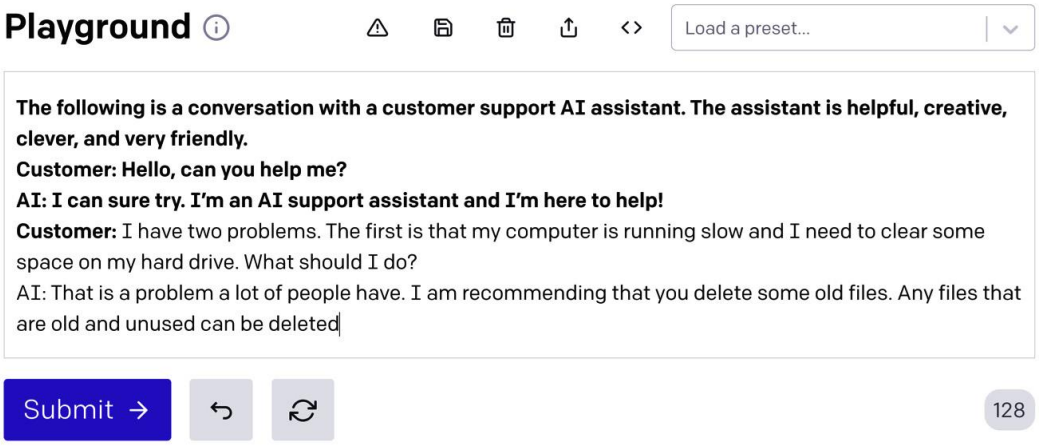

Figure 2.4 – Text generation example – customer support AI assistant

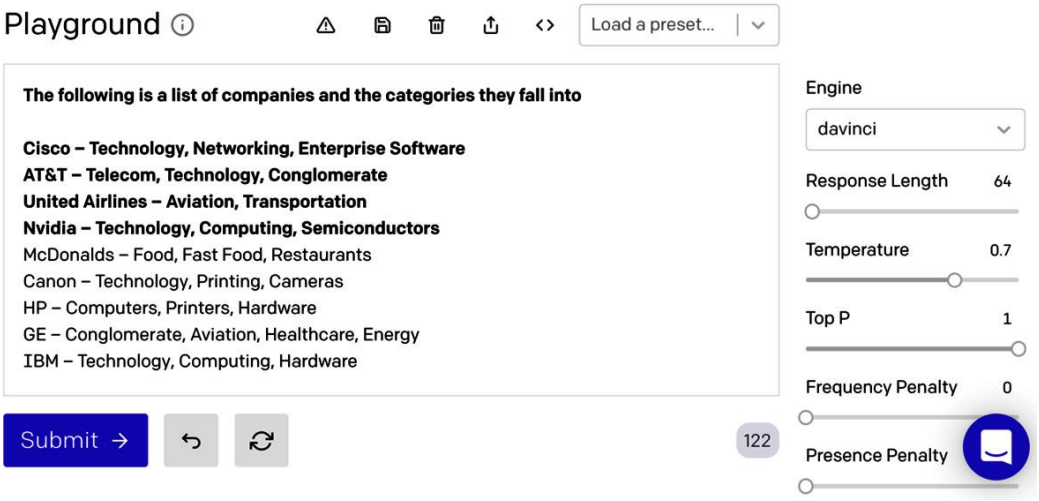

Figure 2.5 – Text generation example – list generation

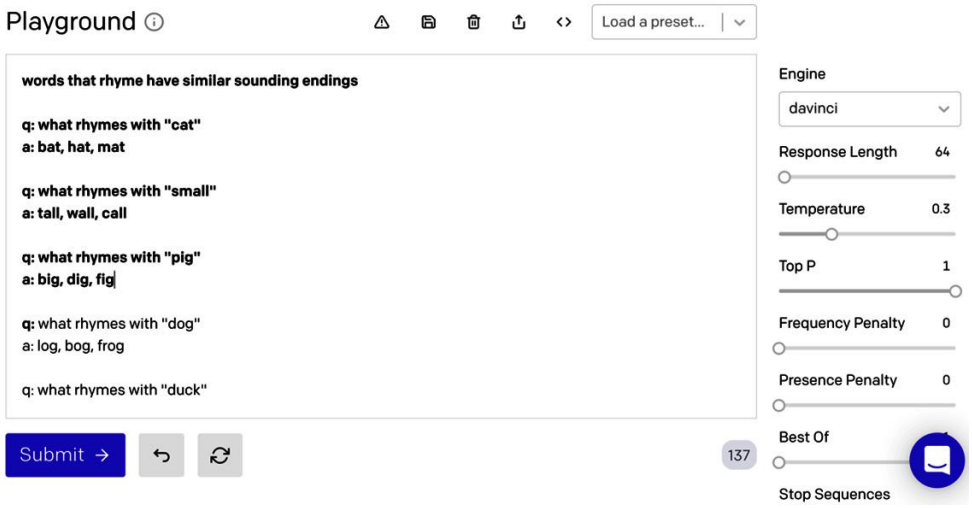

Figure 2.6 – Text generation example – quiz generation

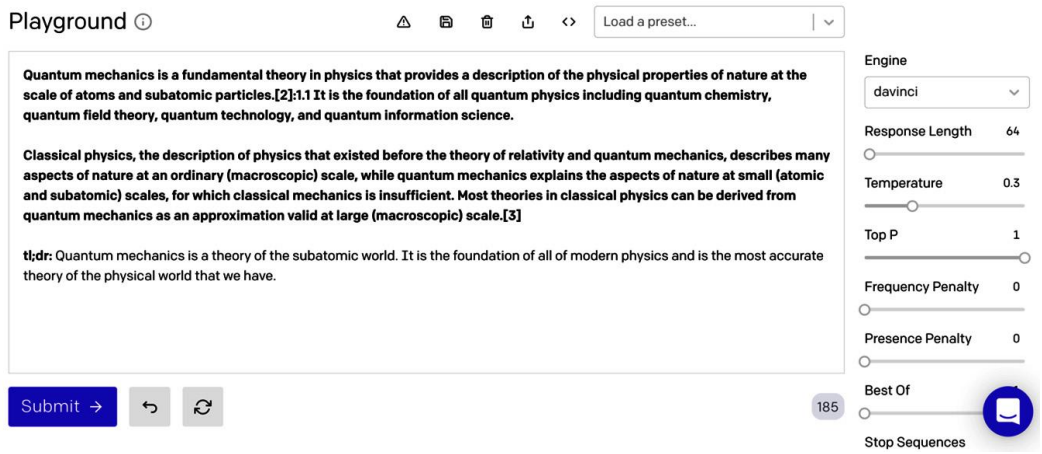

Figure 2.7 – Text generation example – tl;dr: summary

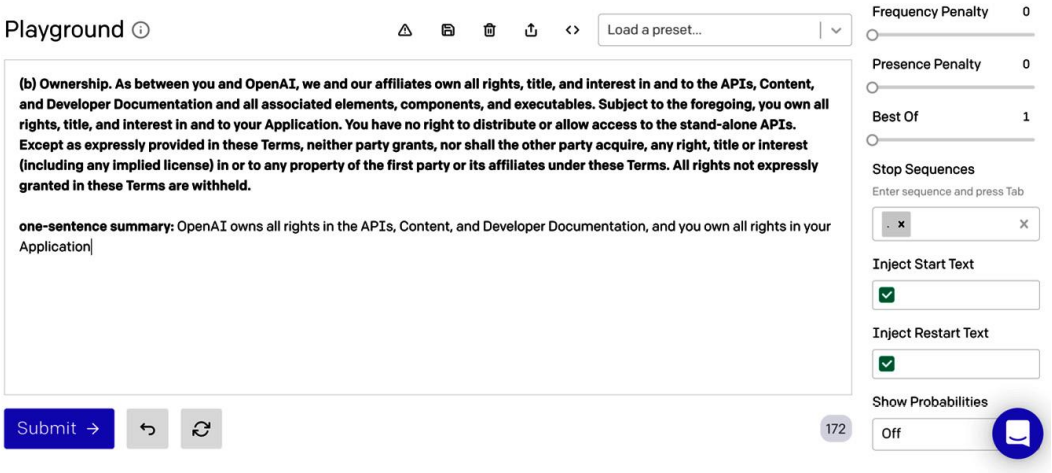

Figure 2.8 – Text generation example – one-sentence summary

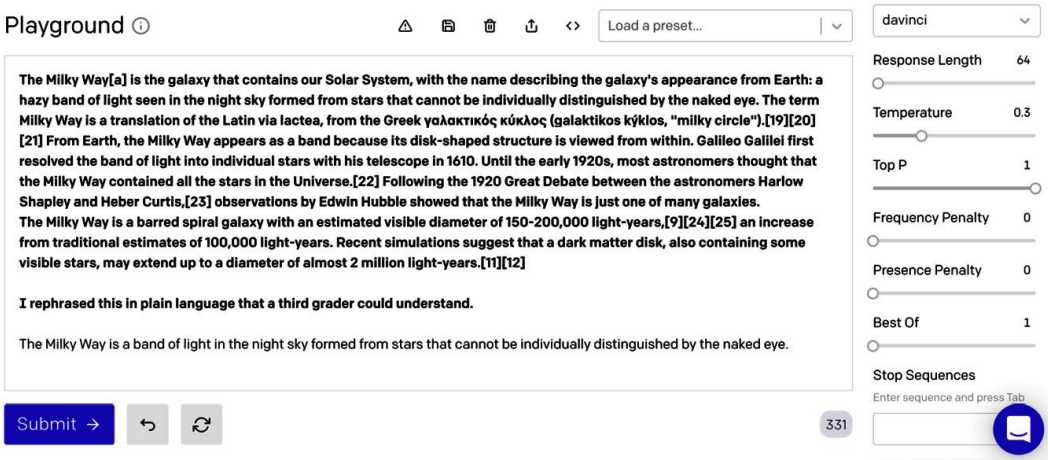

Figure 2.9 – Text generation example – grade-level summarization

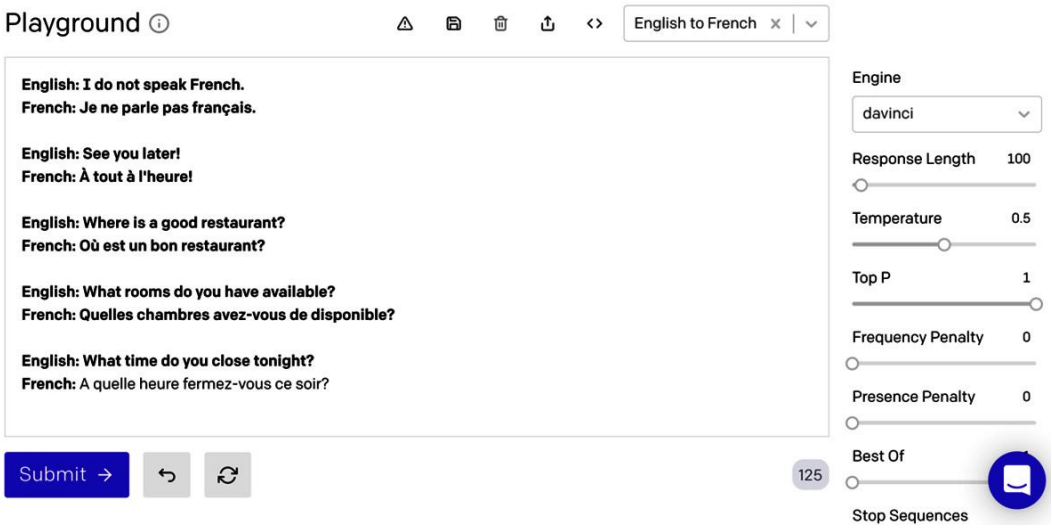

Figure 2.10 – Text generation example – translation from English to French

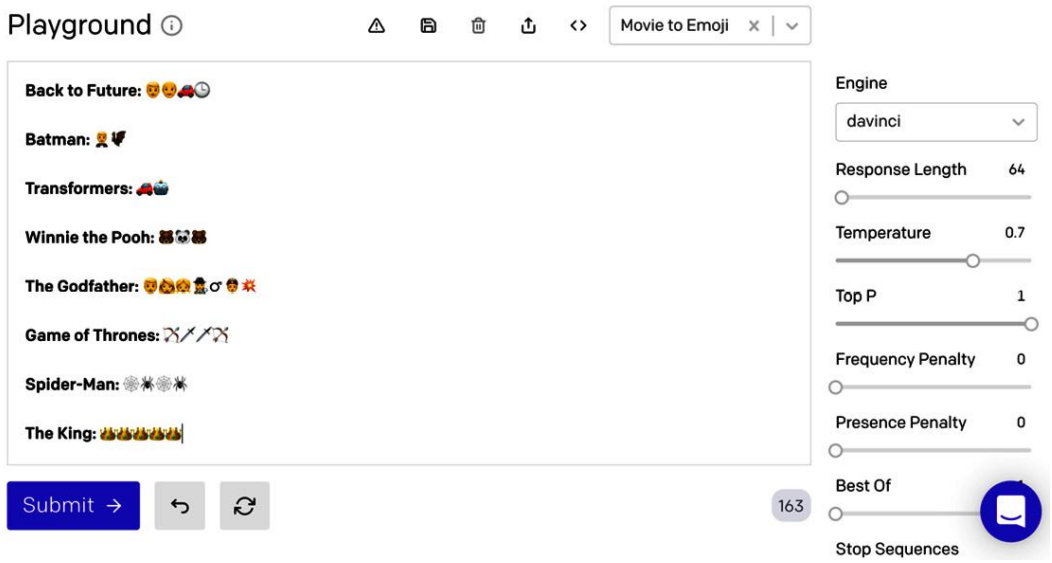

Figure 2.11 – Text generation example – conversion from text to emoji

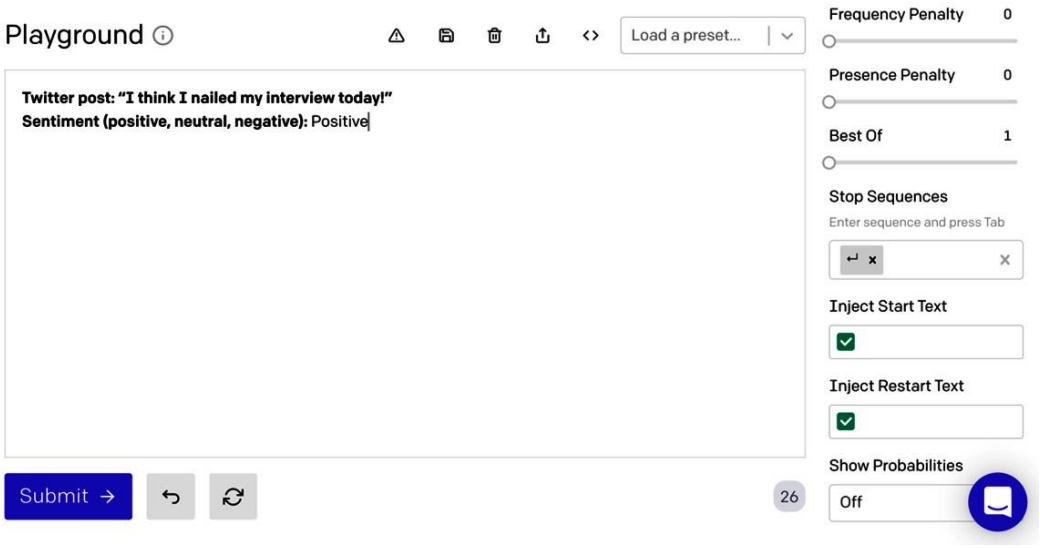

Figure 2.12 – Text generation example – zero-shot classification

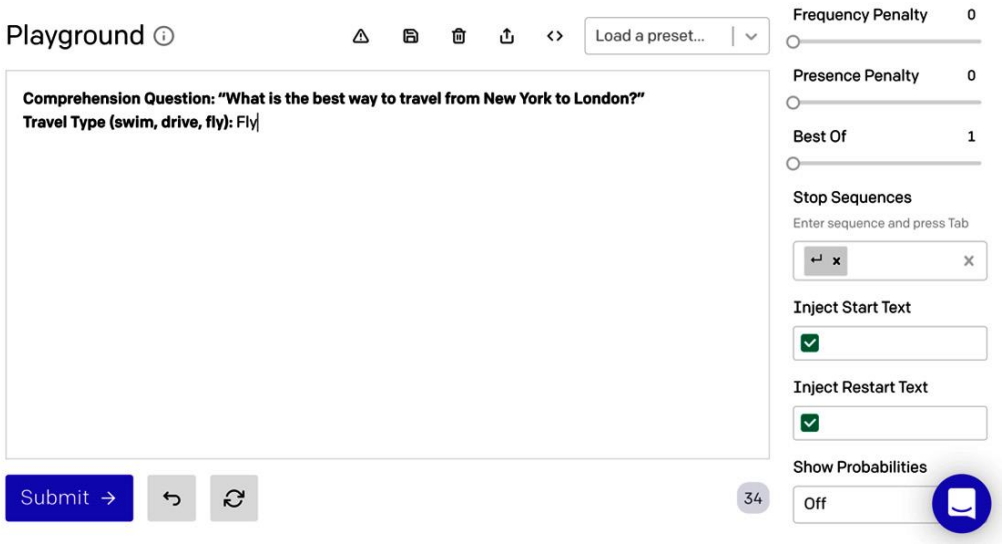

Figure 2.13 – Text generation example – zero-shot classification

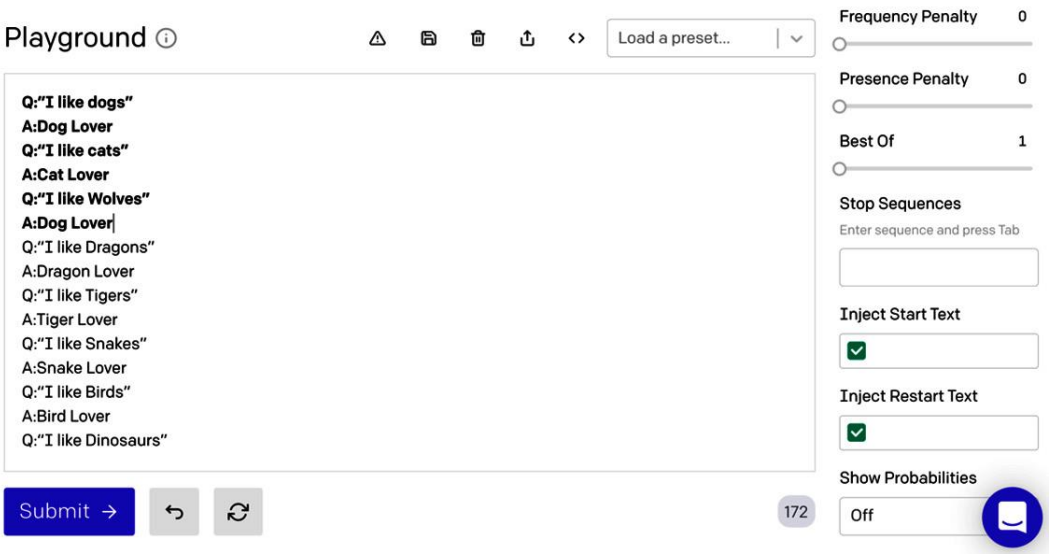

Figure 2.14 – Text generation example – few-shot classification

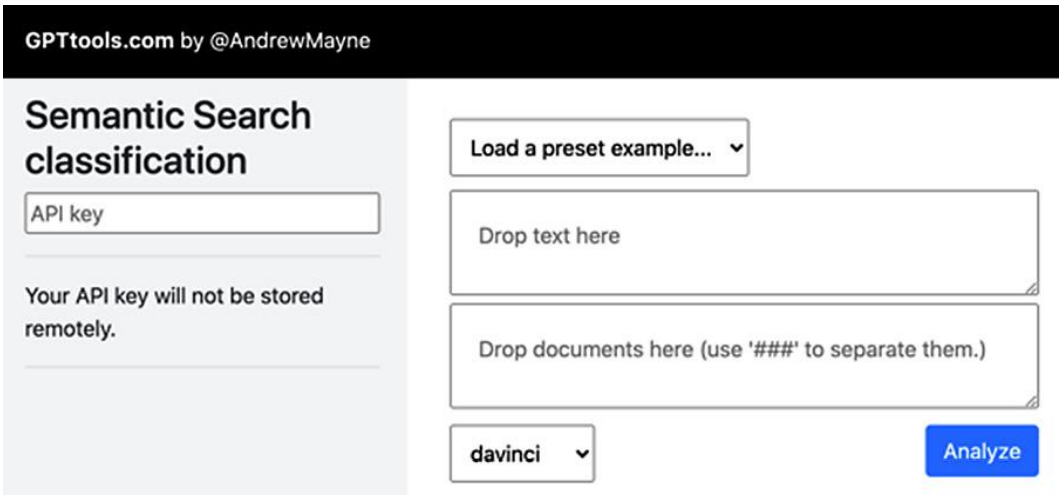

Figure 2.15 – The Semantic Search tool

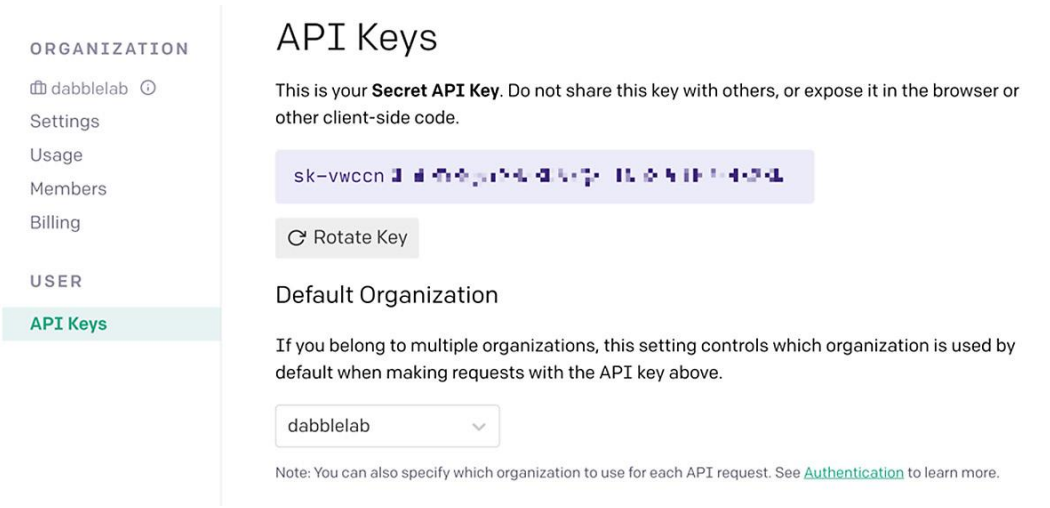

Figure 2.16 – The OpenAI API Keys page (with the API key intentionally blurred out)

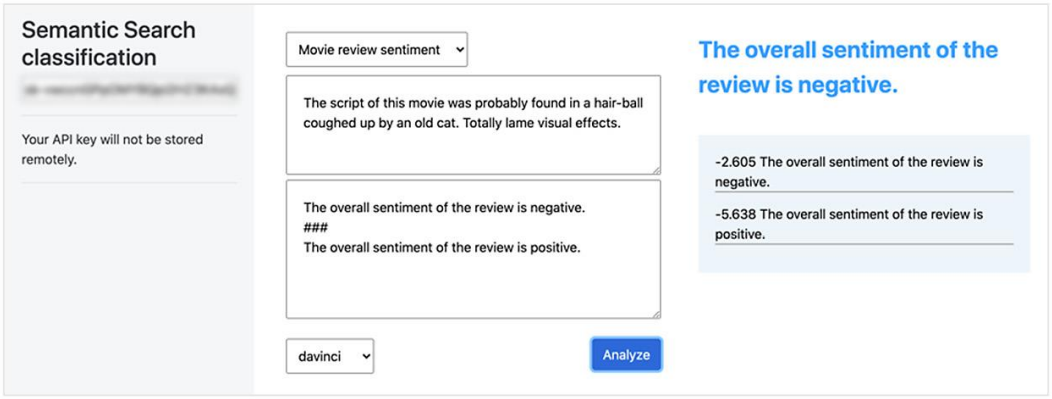

Figure 2.17 – Text generation example – Semantic Search tool

Chapter 3

Technical requirements

This chapter requires that you have access to the **OpenAI API**. You can request access by visiting https://openai.com/.

Links

- https://beta.openai.com/docs/introduction/prompt-design-101

Figures

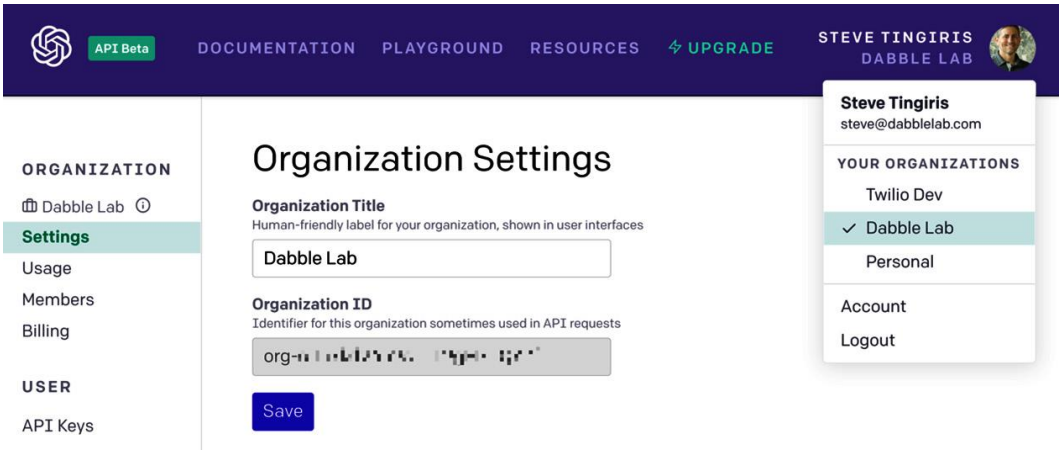

Figure 3.1 – Switching between organizations

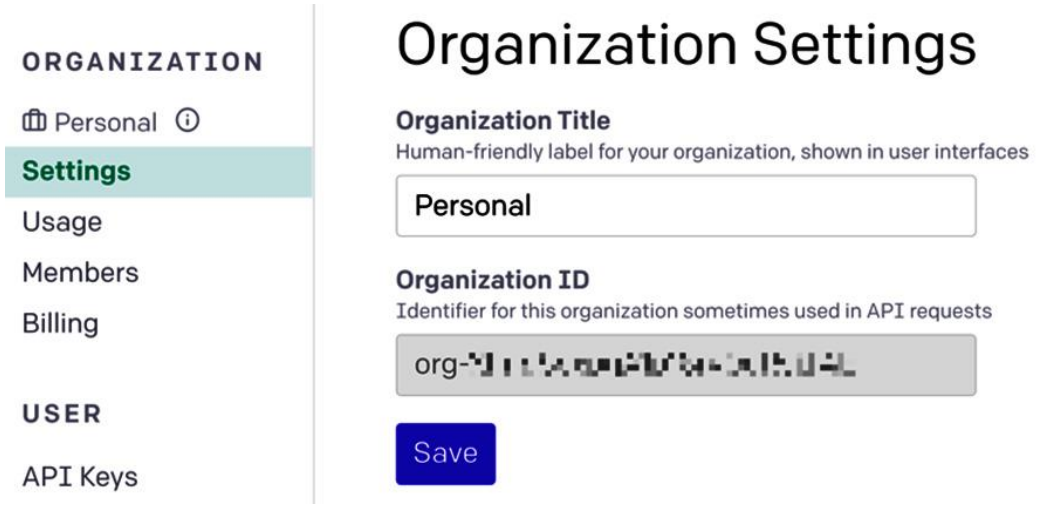

Figure 3.2 – Personal organization

The following screenshot shows the pricing per engine at the time this book was published. Again, the pricing could change at any time, so be sure to verify the current pricing as it may very well have changed:

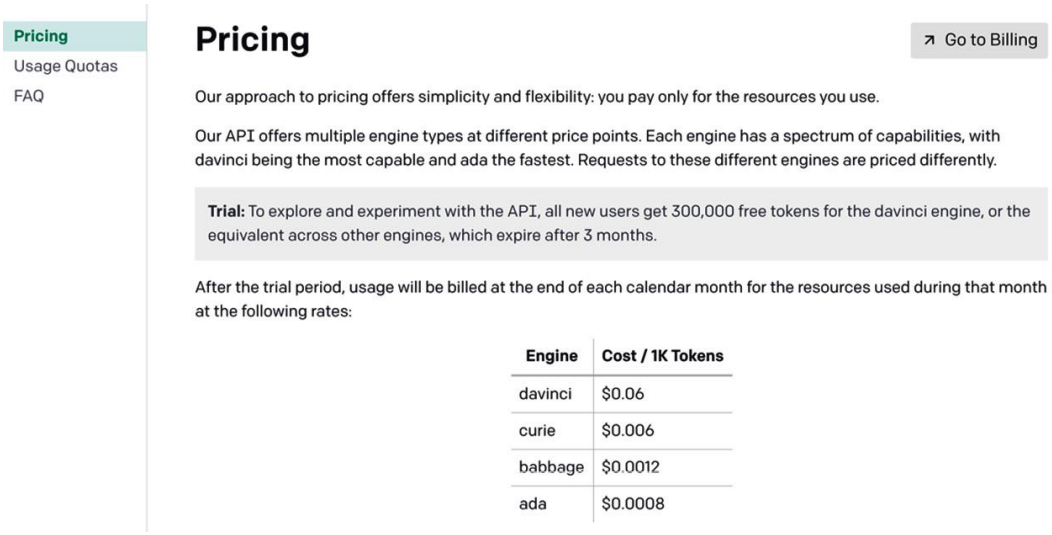

Figure 3.3 – Pricing

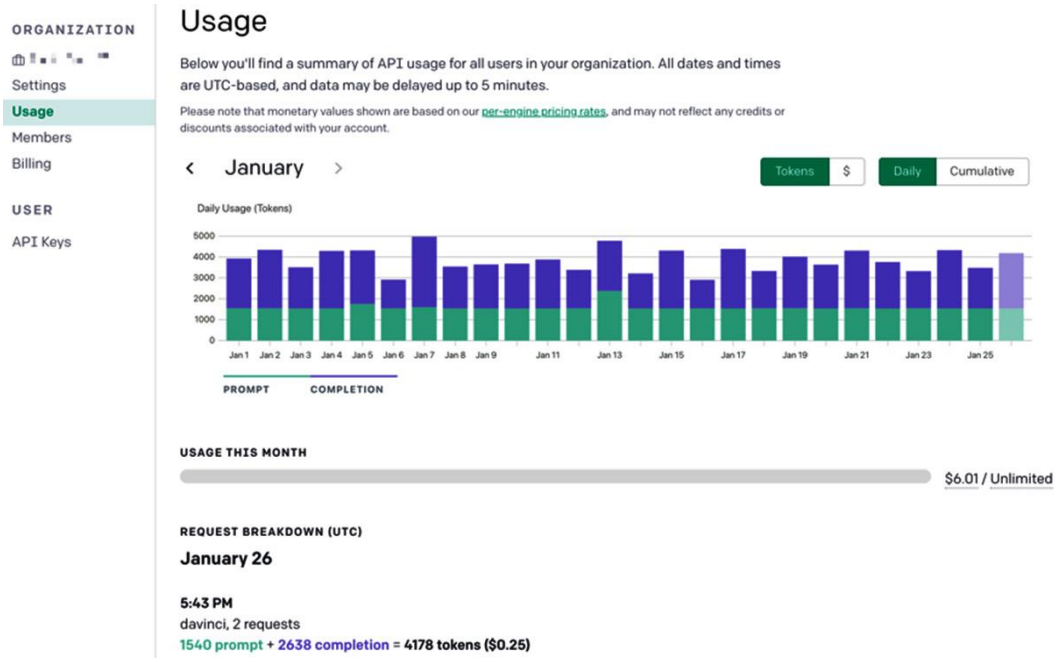

Figure 3.4 – Usage reporting

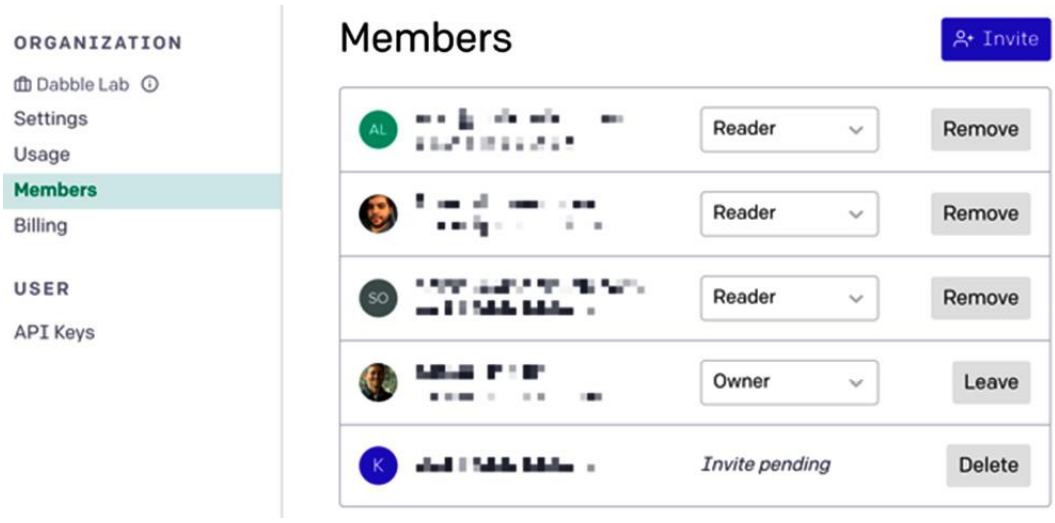

Figure 3.5 – Member management

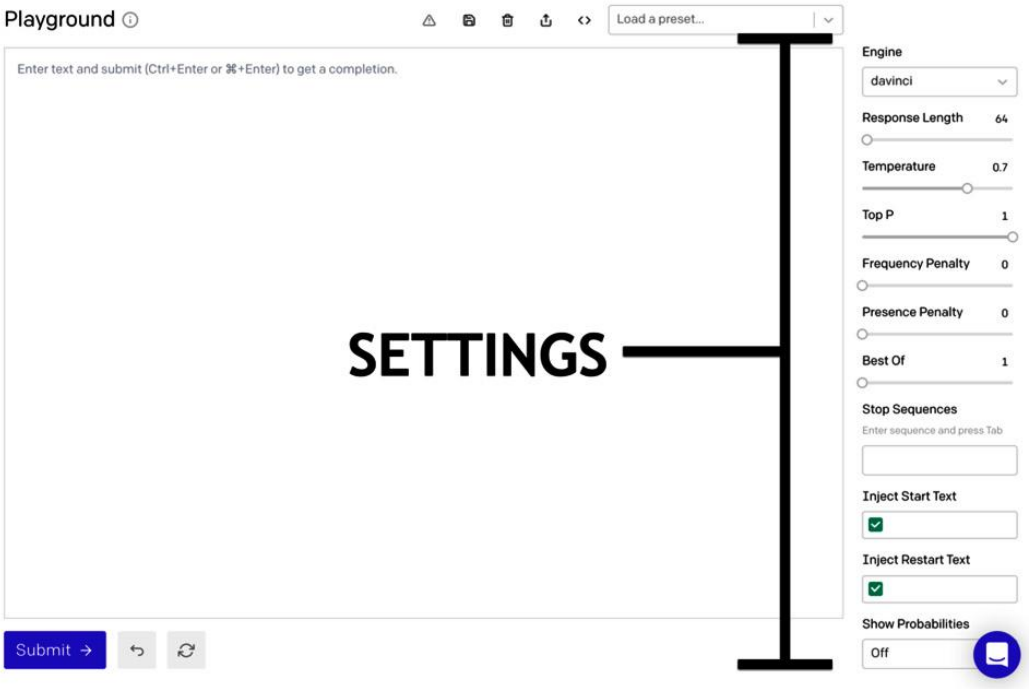

Figure 3.6 – Playground settings

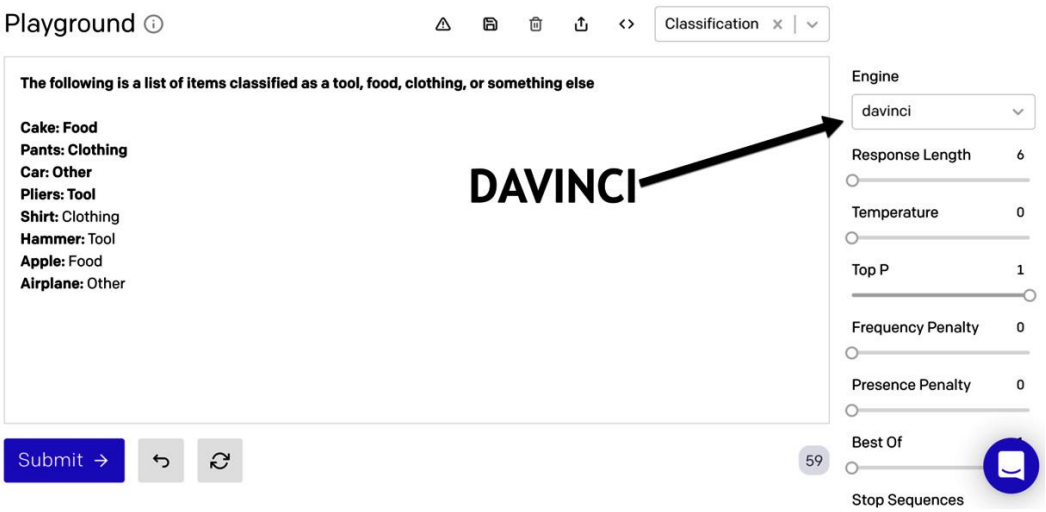

Figure 3.7 – Classification example with the davinci engine

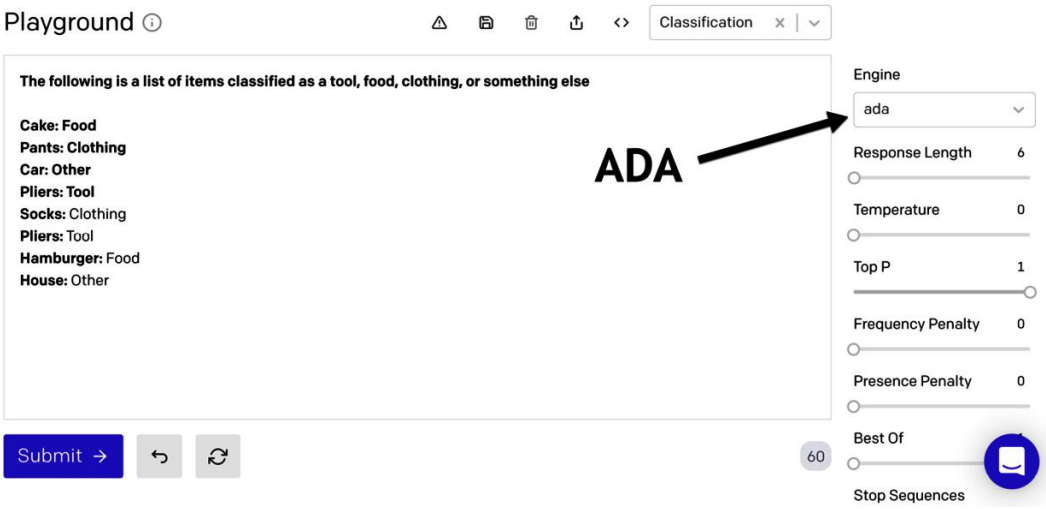

Figure 3.8 – Classification example with the ada engine

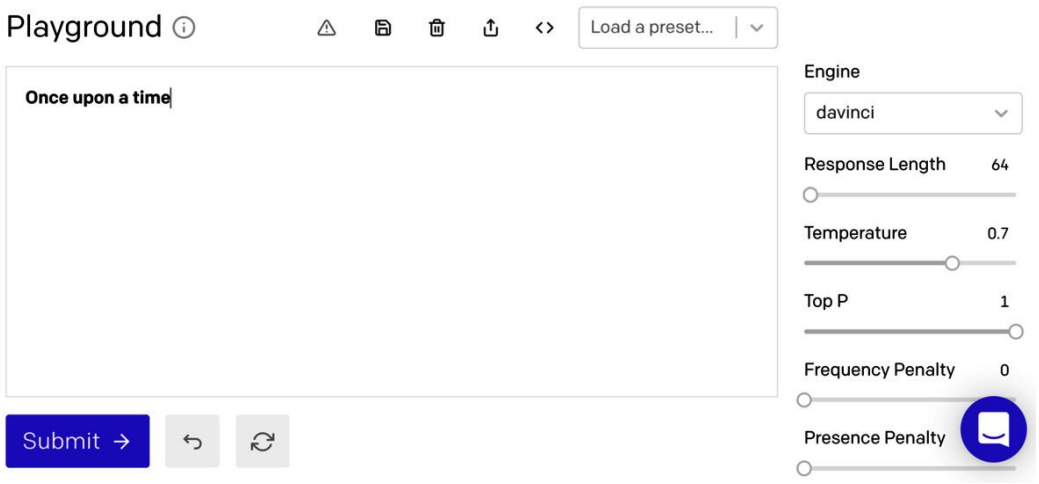

Figure 3.9 – Temperature example 1

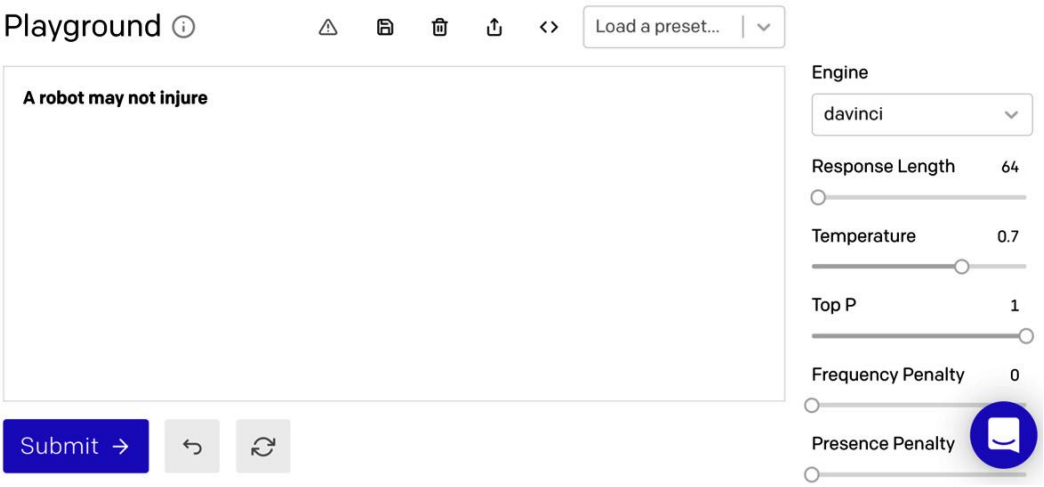

Figure 3.10 – Temperature example 2

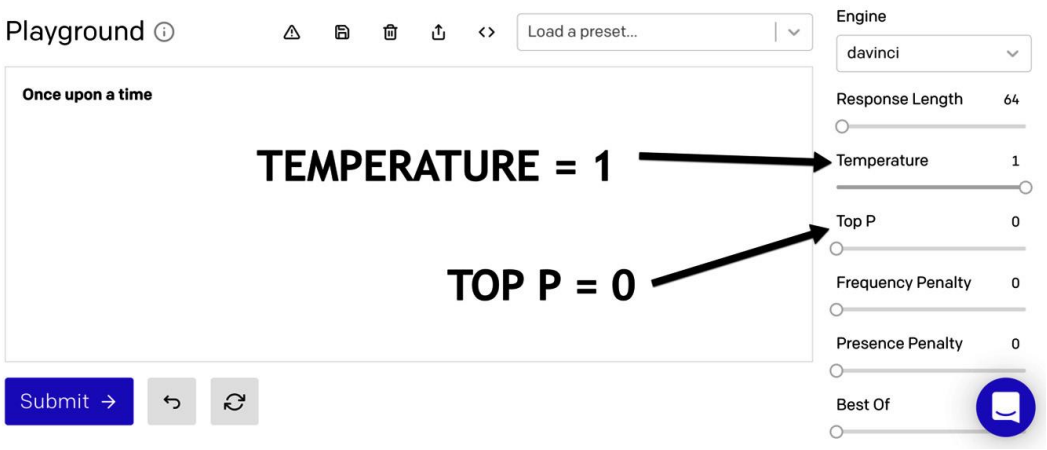

Figure 3.11 – Top P example

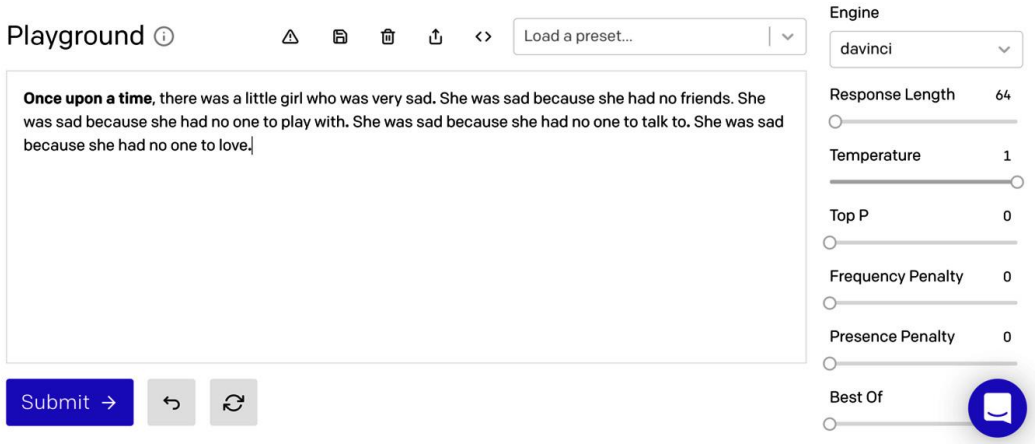

Figure 3.12 – Frequency and presence example 1

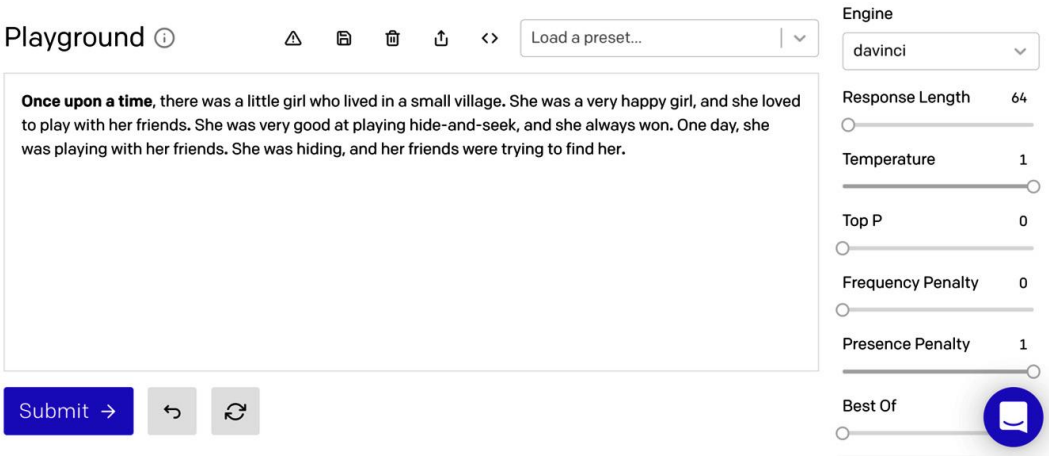

Figure 3.13 – Frequency and presence example 2

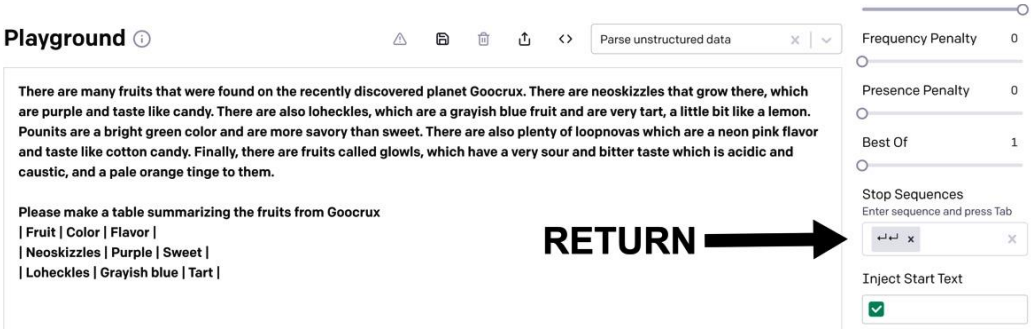

Figure 3.14 – Stop sequence

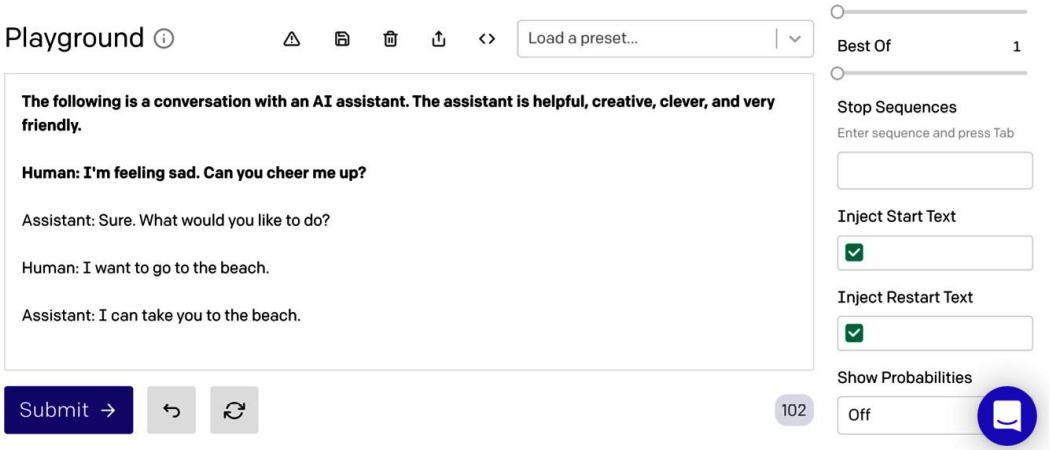

Figure 3.15 – Default settings

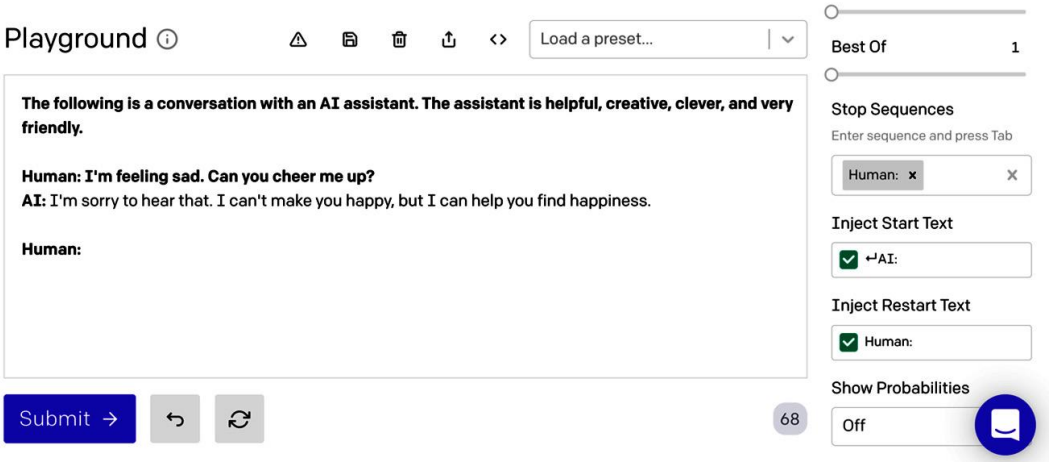

Figure 3.16 – Using Stop Sequences, Inject Start Text, and Inject Restart Text together

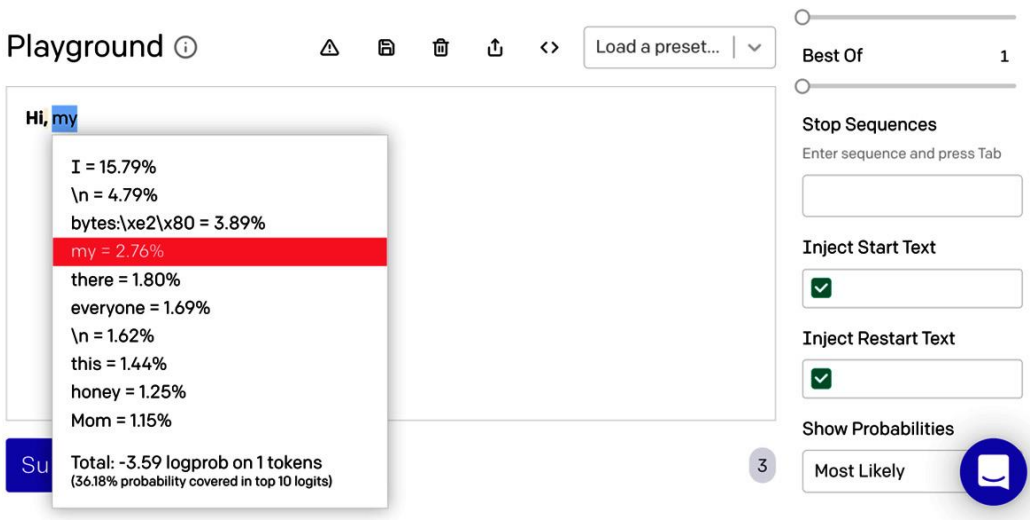

Figure 3.17 – Show Probabilities – most likely tokens

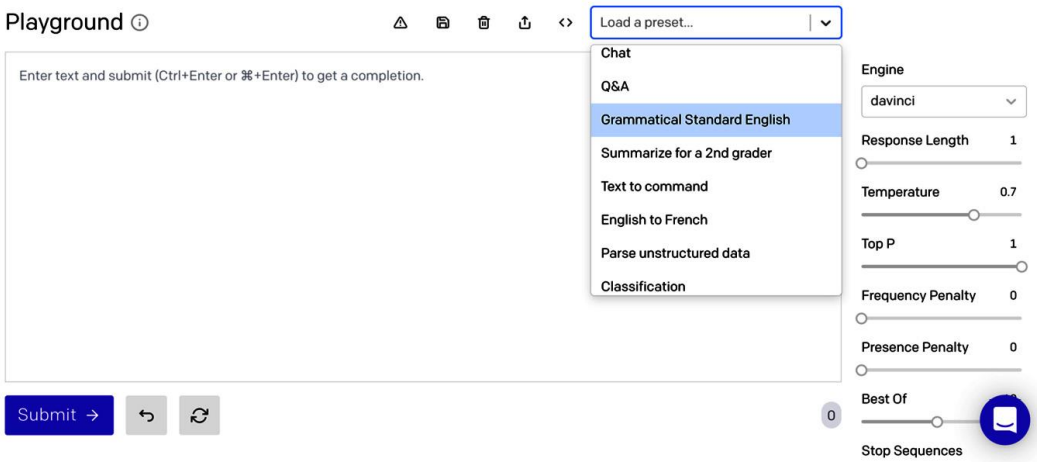

Figure 3.18 – Presets

Prompts

Prompt 3.1

```
Question: Who is Batman?
```

Answer: Batman is a fictional comic book character.

Question: What is torsalplexity?

Answer: ?

Question: What is Devz9?

Answer: ?

Question: Who is George Lucas?

Answer: George Lucas is American film director and producer famous for creating Star Wars.

Question: What is the capital of California?

Answer: Sacramento.

Question: What orbits the Earth?

Answer: The Moon.

Question: Who is Fred Rickerson?

Answer: ?

Question: What is an atom?

Answer: An atom is a tiny particle that makes up everything.

Question: Who is Alvan Muntz?

Answer: ?

Question: What is Kozar-09?

Answer: ?

Question: How many moons does Mars have?

```
Answer: Two, Phobos and Deimos.


Question:
```

Prompt 3.2

The following is a prompt classifying items as a tool, food, clothing, or something else:

```
The following is a list of items classified as a tool, food, clothing, or something else

Cake: Food

Pants: Clothing

Car: Other

Pliers: Tool
```

Prompt 3.3

```
The following is a conversation with an AI assistant. The assistant is helpful, creative, clever, and very friendly.

Human: I'm feeling sad. Can you cheer me up?
```

Prompt 3.4

```
Non-standard English: Please provide me with a short brief of the design you're looking for and that'd be nice if you could share some examples or project you did before.

Standard American English: Please provide me with a short brief of the design you're looking for and some examples or previous projects you've done would be helpful.

Non-standard English: If I'm stressed out about something, I tend to have problem to fall asleep.

Standard American English: If I'm stressed out about something, I tend to have a problem falling asleep.

Non-standard English: There is plenty of fun things to do in the summer when your able to go outside.

Standard American English: There are plenty of fun things to do in the summer when you are able to go outside.

Non-standard English: She no went to the market.
```

```
Standard American English: She didn't go to the market.
```

Prompt 3.5 – Text to command

```
Question: Ask Constance if we need some bread

Answer: send-msg `find constance` Do we need some bread?

Question: Send a message to Greg to figure out if things are ready for Wednesday.

Answer: send-msg `find greg` Is everything ready for Wednesday?

Question: Ask Ilya if we're still having our meeting this evening

Answer: send-msg `find ilya` Are we still having a meeting this evening?

Question: Contact the ski store and figure out if I can get my skis fixed before I leave on Thursday

Answer: send-msg `find ski store` Would it be possible to get my skis fixed before I leave on Thursday?

Question: Thank Nicolas for lunch

Answer: send-msg `find nicolas` Thank you for lunch!

Question: Tell Constance that I won't be home before 19:30 tonight — unmovable meeting.

Answer: send-msg `find constance` I won't be home before 19:30 tonight. I have a meeting I can't move.

Question:
```

Prompt 3.6 – Parse unstructured data

```
There are many fruits that were found on the recently discovered planet Goocrux. There are neoskizzles that grow there, which are purple and taste like candy. There are also loheckles, which are a grayish blue fruit and are very tart, a little bit like a lemon. Pounits are a bright green color and are more savory than sweet. There are also plenty of loopnovas which are a neon pink flavor and taste like cotton candy. Finally, there are fruits called glowls, which have a very sour and bitter taste which is acidic and caustic, and a pale orange tinge to them.

Please make a table summarizing the fruits from Goocrux

| Fruit | Color | Flavor |
```

```
| Neoskizzles | Purple | Sweet |
| Loheckles | Grayish blue | Tart |
```

Chapter 4

Technical requirements

This chapter requires that you have access to the **OpenAI API**. You can request access by visiting https://openai.com/.

Links

- For a full list of the different status codes, you can visit https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.
- You can learn more about create completion endpoint by visiting https://beta.openai.com/docs/api-reference/create-completion-via-get.
- If you don't have CURL installed, you can download it from the official site at https://curl.se/download.html.

Figures


```
{"message": "success", "number": 7, "people":
[{"craft": "ISS", "name": "Sergey Ryzhikov"},
{"craft": "ISS", "name": "Kate Rubins"},
{"craft": "ISS", "name": "Sergey Kud-
Sverchkov"}, {"craft": "ISS", "name": "Mike
Hopkins"}, {"craft": "ISS", "name": "Victor
Glover"}, {"craft": "ISS", "name": "Shannon
Walker"}, {"craft": "ISS", "name": "Soichi
Noguchi"}]}
```


Figure 4.1 – Open-Notify API – JSON response

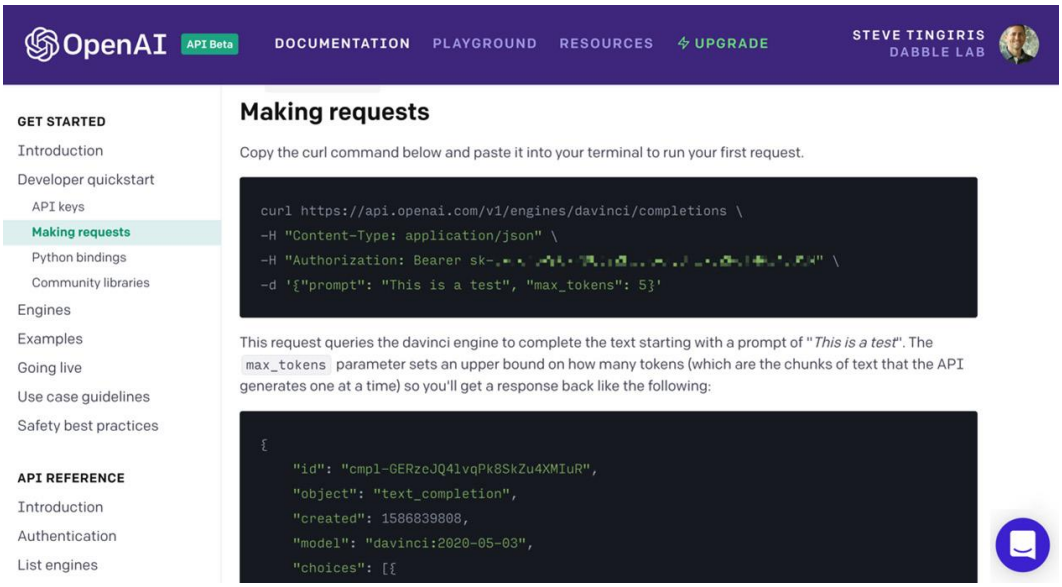

Figure 4.2 – CURL command in the OpenAI API docs

```
Last login: Fri Jan  1 10:57:12 on console
$ curl
curl: try 'curl --help' or 'curl --manual' for more information
$ curl --help
Usage: curl [options...] <url>
     --abstract-unix-socket <path> Connect via abstract Unix domain socket
     --alt-svc <file name> Enable alt-svc with this cache file
     --anyauth       Pick any authentication method
 -a, --append        Append to target file when uploading
     --basic         Use HTTP Basic Authentication
     --cacert <file> CA certificate to verify peer against
     --capath <dir>  CA directory to verify peer against
 -E, --cert <certificate[:password]> Client certificate file and password
     --cert-status   Verify the status of the server certificate
     --cert-type <type> Certificate file type (DER/PEM/ENG)
     --ciphers <list of ciphers> SSL ciphers to use
     --compressed    Request compressed response
     --compressed-ssh Enable SSH compression
 -K, --config <file> Read config from a file
     --connect-timeout <seconds> Maximum time allowed for connection
     --connect-to <HOST1:PORT1:HOST2:PORT2> Connect to host
 -C, --continue-at <offset> Resumed transfer offset
 -b, --cookie <data|filename> Send cookies from string/file
 -c, --cookie-jar <filename> Write cookies to <filename> after operation
     --create-dirs   Create necessary local directory hierarchy
     --crlf          Convert LF to CRLF in upload
     --crlfile <file> Get a CRL list in PEM format from the given file
 -d, --data <data>   HTTP POST data
     --data-ascii <data> HTTP POST ASCII data
     --data-binary <data> HTTP POST binary data
     --data-raw <data> HTTP POST data, '@' allowed
     --data-urlencode <data> HTTP POST data url encoded
     --delegation <LEVEL> GSS-API delegation permission
     --digest        Use HTTP Digest Authentication
 -q, --disable       Disable .curlrc
     --disable-eprt  Inhibit using EPRT or LPRT
     --disable-epsv  Inhibit using EPSV
     --disallow-username-in-url Disallow username in url
     --dns-interface <interface> Interface to use for DNS requests
     --dns-ipv4-addr <address> IPv4 address to use for DNS requests
     --dns-ipv6-addr <address> IPv6 address to use for DNS requests
     --dns-servers <addresses> DNS server addrs to use
     --doh-url <URL> Resolve host names over DOH
 -D, --dump-header <filename> Write the received headers to <filename>
     --egd-file <file> EGD socket path for random data
     --engine <name> Crypto engine to use
     --expect100-timeout <seconds> How long to wait for 100-continue
 -f, --fail          Fail silently (no output at all) on HTTP errors
     --fail-early    Fail on first transfer error, do not continue
     --false-start   Enable TLS False Start
 -F, --form <name=content> Specify multipart MIME data
```

Figure 4.3 – Curl Help command

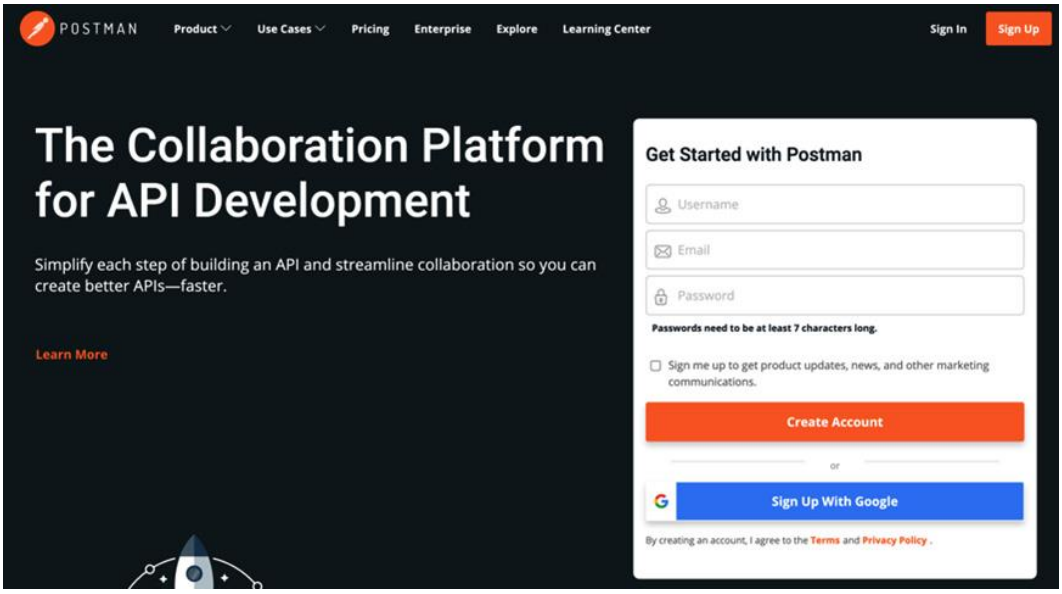

Figure 4.4 – Postman home page

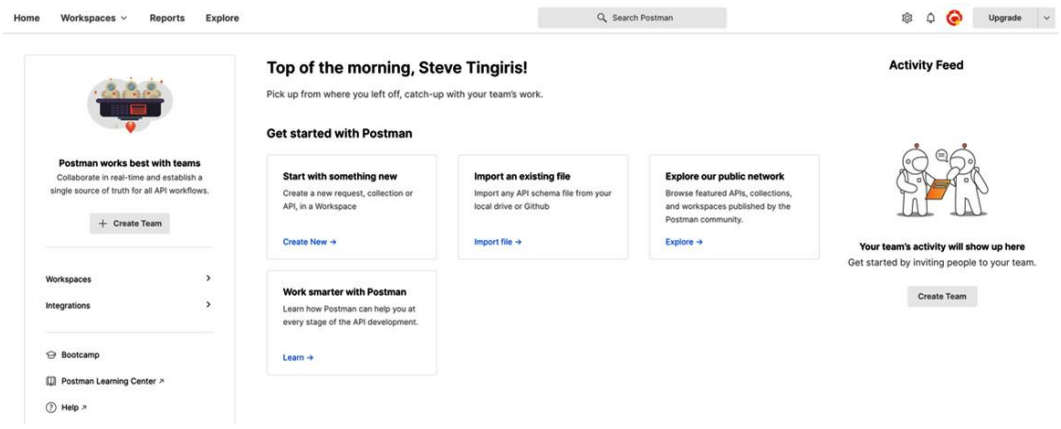

Figure 4.5 – Postman welcome screen

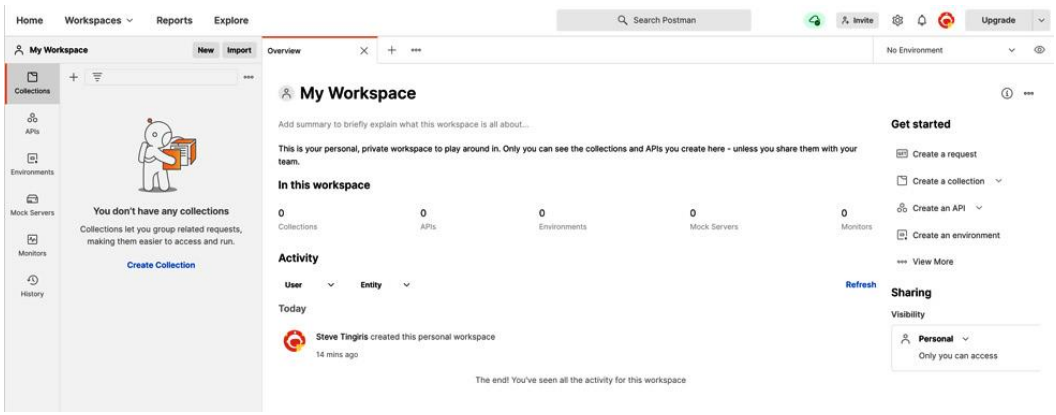

Figure 4.6 – My Workspace

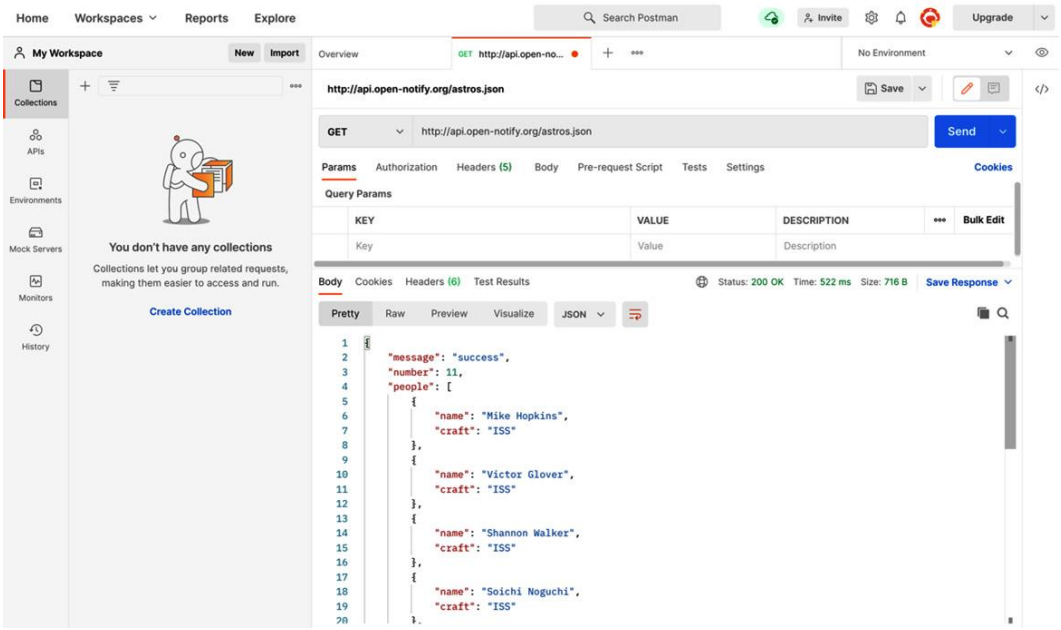

Figure 4.7 – Postman request results

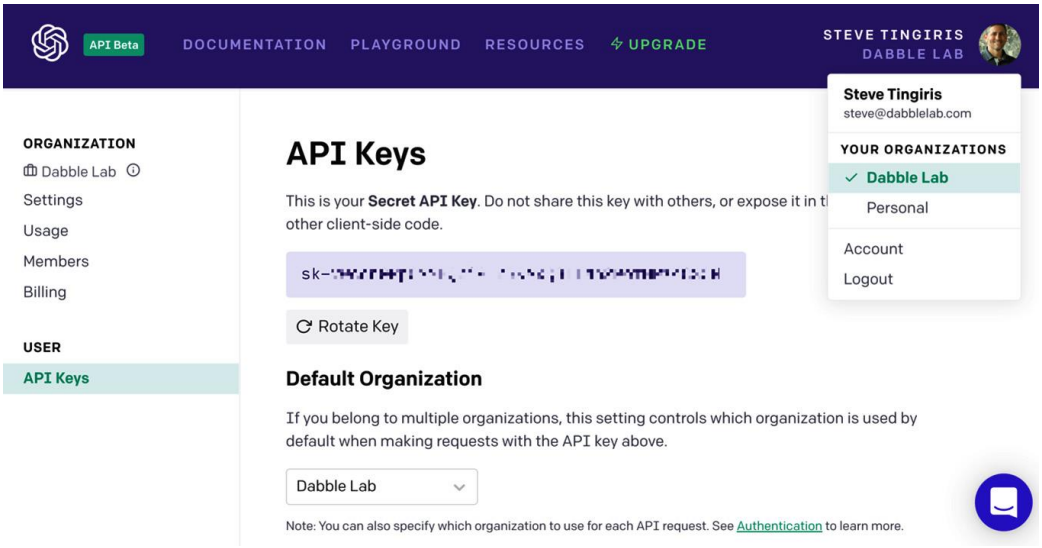

Figure 4.8 – API Keys

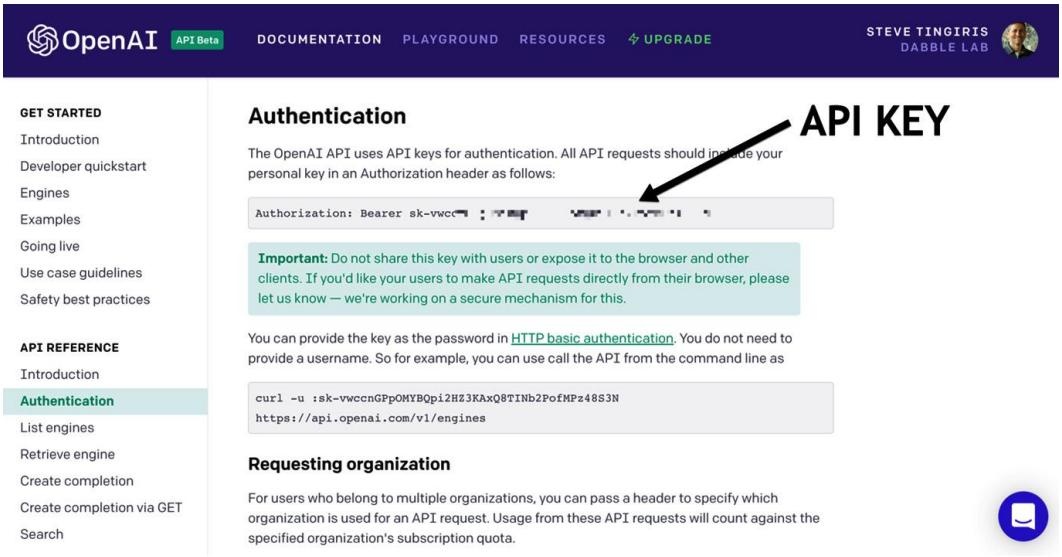

Figure 4.9 – API key in documentation

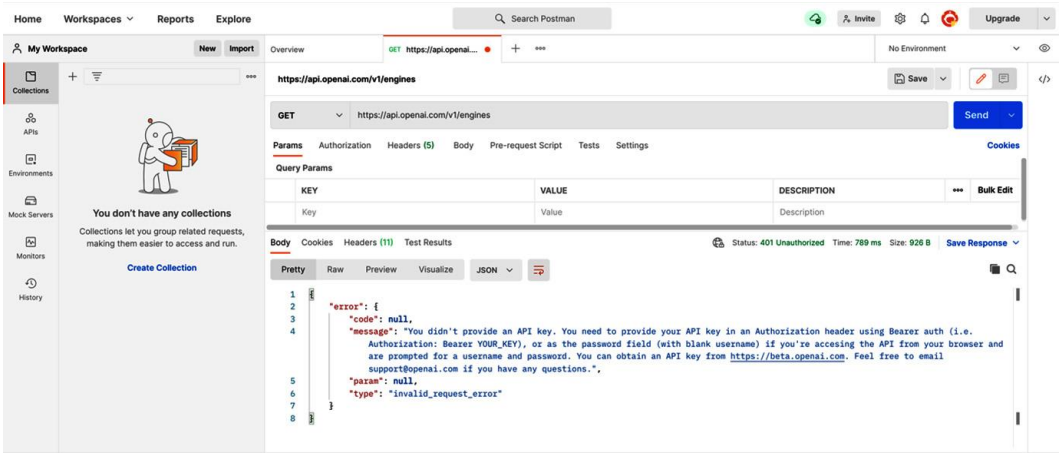

Figure 4.10 – API request without the API key

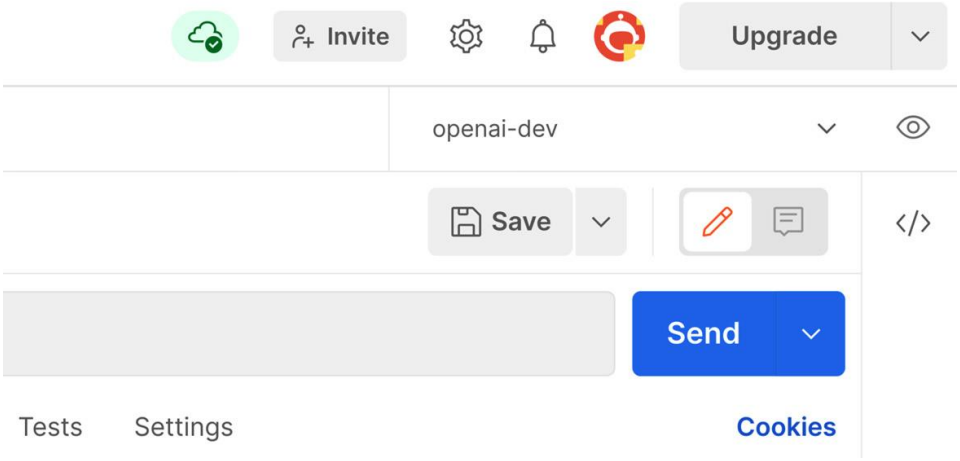

Figure 4.11 – Postman with the environment set

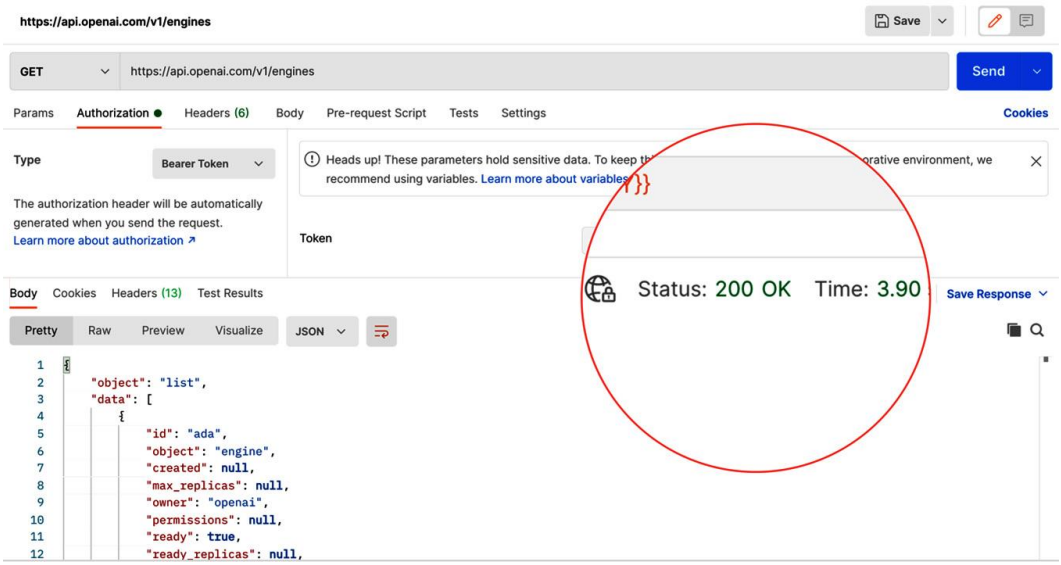

Figure 4.12 – API request using the API key as a bearer token

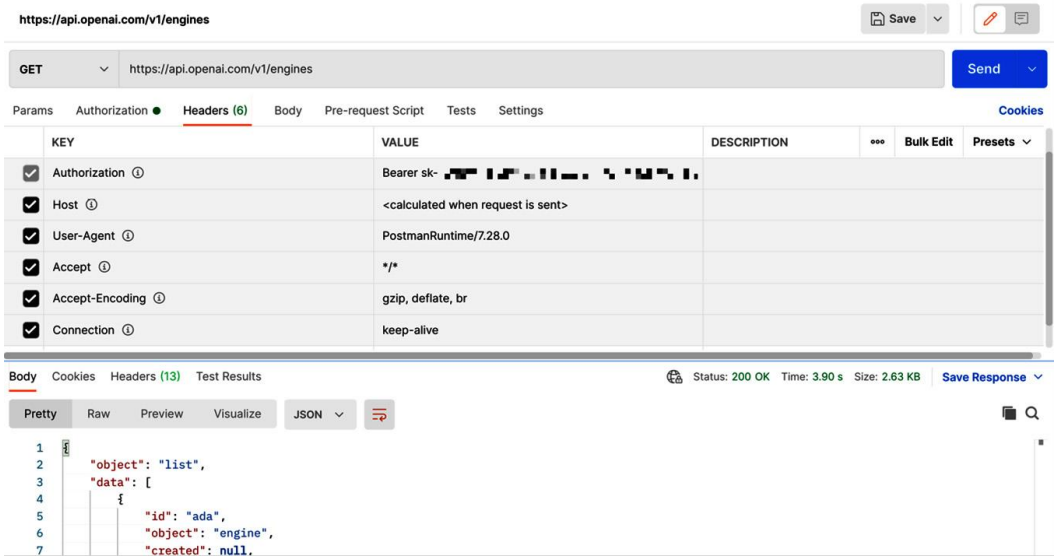

Figure 4.13 – Authorization header with the API key as a bearer token

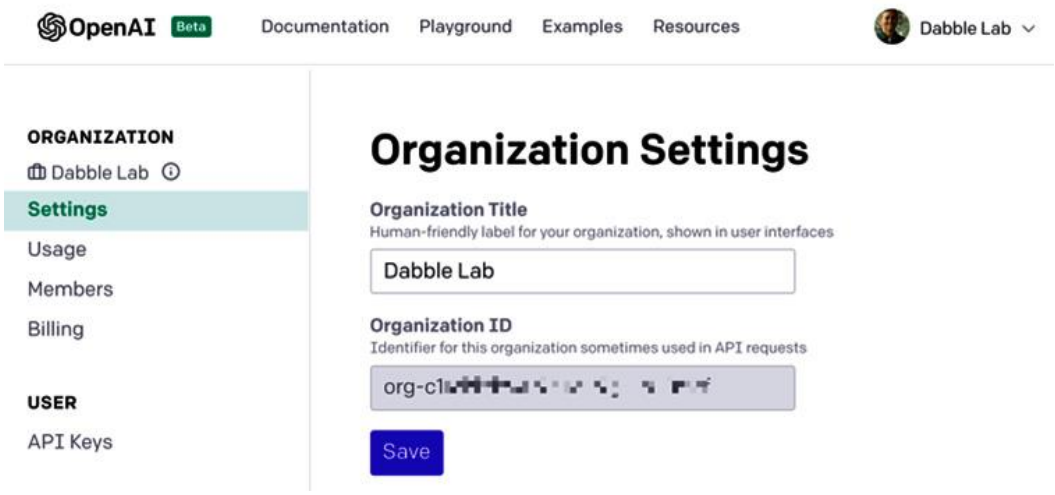

Figure 4.14 – Finding your organization ID

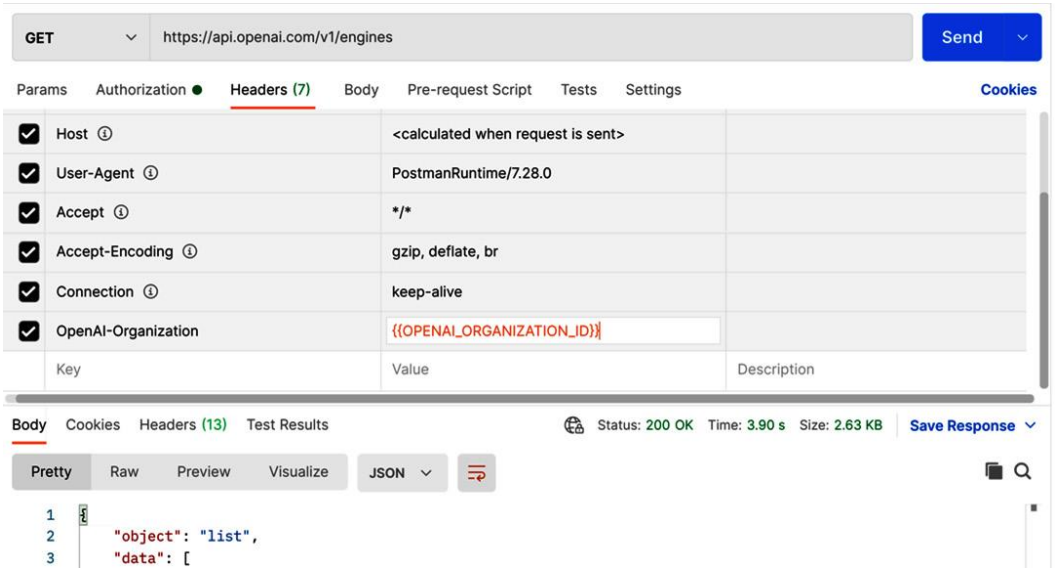

Figure 4.15 – Using the OpenAI-Organization HTTP header

```
$ curl https://api.openai.com/v1/engines \
>   -H 'Authorization: Bearer sk-...' \
>   -H 'OpenAI-Organization: org-...'
{
  "object": "list",
  "data": [
    {
      "id": "ada",
      "object": "engine",
      "created": null,
      "max_replicas": null,
      "owner": "openai",
      "permissions": null,
      "ready": true,
      "ready_replicas": null,
      "replicas": null
    },
    {
      "id": "babbage",
      "object": "engine",
      "created": null,
      "max_replicas": null,
      "owner": "openai",
      "permissions": null,
      "ready": true,
      "ready_replicas": null,
      "replicas": null
    },
    {
      "id": "content-filter-alpha-c4",
      "object": "engine",
      "created": null,
      "max_replicas": null,
      "owner": "openai",
      "permissions": null,
      "ready": true,
      "ready_replicas": null,
      "replicas": null
    },
```

Figure 4.16 – Using CURL to call the List Engines endpoint

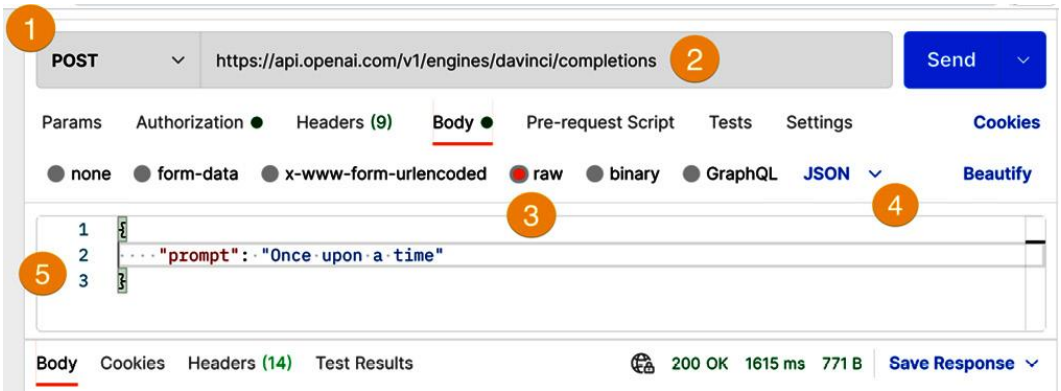

Figure 4.17 – Postman settings for the Completions endpoint

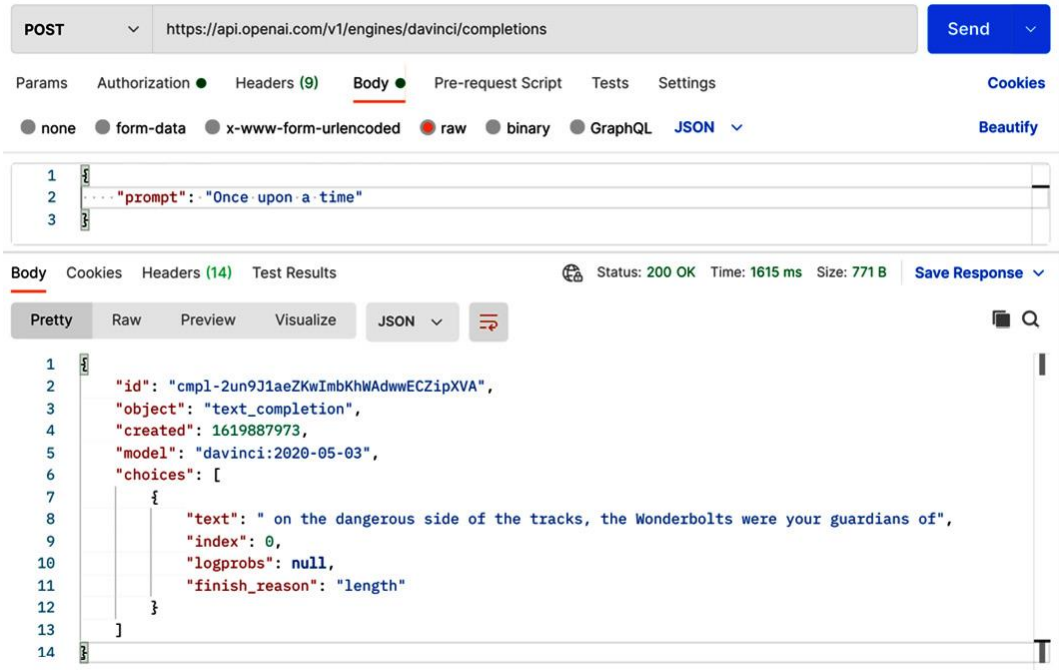

Figure 4.18 – Postman response from the Completions endpoint

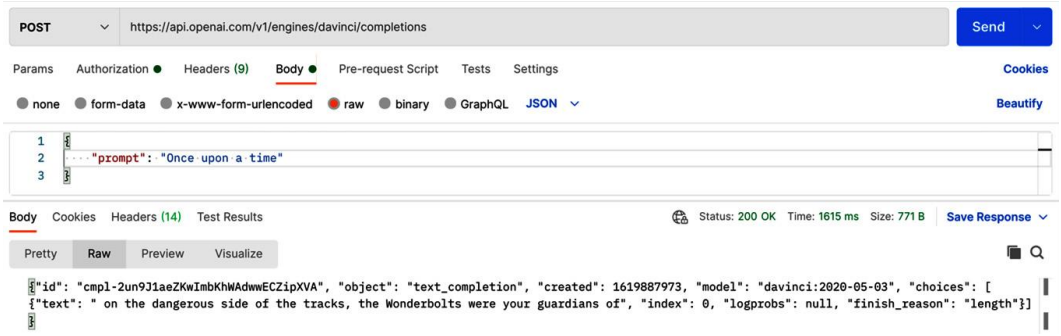

Figure 4.19 – Postman response from the Completions endpoint – Raw

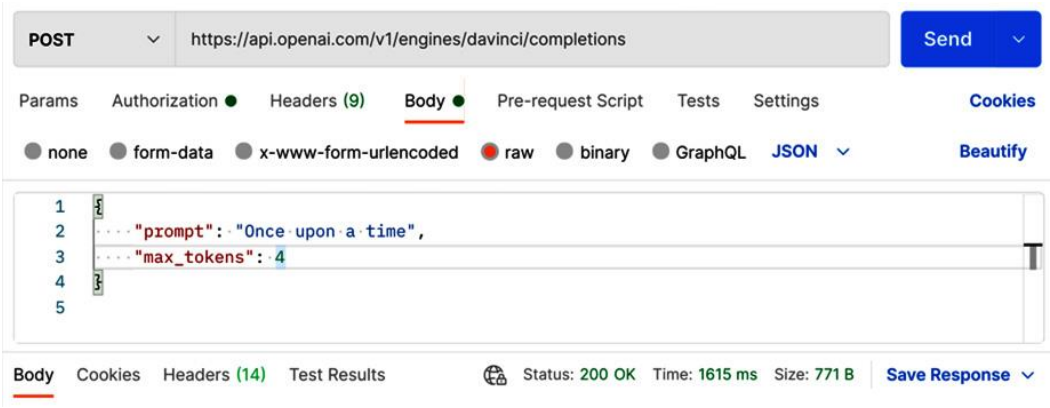

Figure 4.20 – Postman response from the Completions endpoint with max_tokens

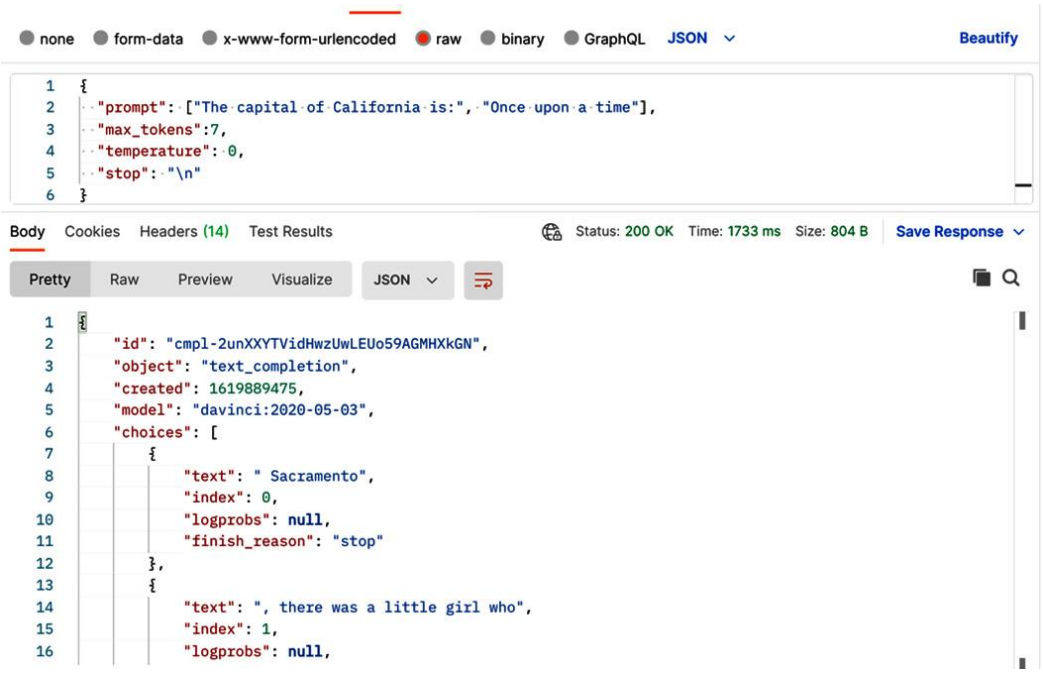

Figure 4.21 – Sending multiple prompts

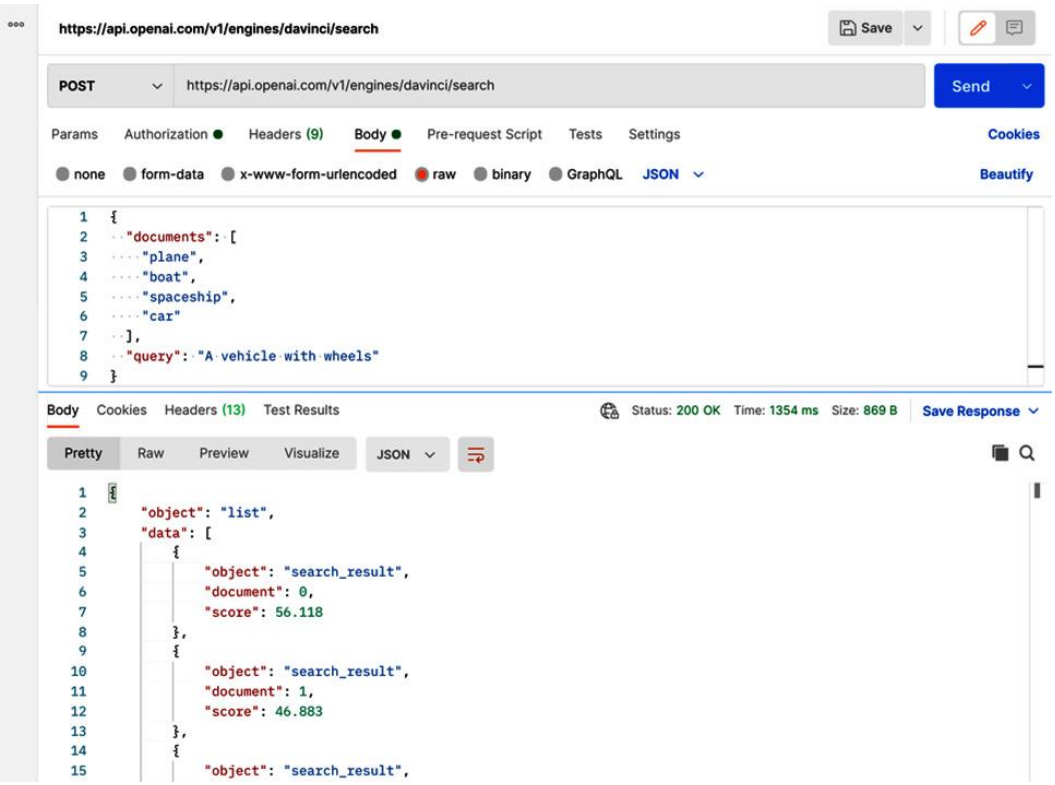

Figure 4.22 – Semantic Search results

Code

Code 4.1 – The following is the HTTP method (GET) and URI for the List Engines endpoint:

```
get https://api.openai.com/v1/engines
```

Code 4.2 – The Retrieve Engine endpoint uses the HTTP get method and the following URI with one parameter, the engine ID:

```
get https://api.openai.com/v1/engines/{engine_id}
```

Code 4.3 – The Completions endpoint also uses the post method and requires an engine ID as a URI parameter:

```
post https://api.openai.com/v1/engines/{engine_id}/completions
```

Code 4.4 – The Semantic Search endpoint uses the post method and requires an engine ID as a URI parameter:

```
post https://api.openai.com/v1/engines/{engine_id}/search
```

Code 4.5 – The following is an example of a CURL command for calling the List Engines endpoint:

```
curl https://api.openai.com/v1/engines \
  -H 'Authorization: Bearer {your-api-key}' \
  -H 'OpenAI-Organization: {your-orgainzation-id}'
```

Code 4.6 – The following code block is an example of a JSON object from an OpenAI API response.

```
{
    "id": "cmpl-2T0IrOkctsOm8uVFvDDEmc1712U9R",
    "object": "text_completion",
    "created": 1613265353,
    "model": "davinci:2020-05-03",
    "choices": [
        {
            "text": ", there was a dog",
            "index": 0,
            "logprobs": null,
            "finish_reason": "length"
        }
    ]
}
```

Code 4.7 – Technically, the JSON body could just be an empty object (just a left and right curly brace, like `{})`, but minimally, you'll want to include at least the `prompt` element with the value set to your `prompt` string, something like the following JSON example:

```
{"prompt": "Once upon a time"}
```

Code 4.8 – We'll use the following JSON to submit multiple prompts simultaneously and get back a completion for each:

```
{
  "prompt": ["The capital of California is:", "Once upon a time"],
  "max_tokens":7,
  "temperature": 0,
  "stop": "\n"
}
```

Code 4.9 – For example, the following JSON object provides a list of vehicles (plane, boat, spaceship, or car) as the documents and the query `"A vehicle with wheels"`:

```
{
  "documents": [
    "plane",
    "boat",
    "spaceship",
    "car"
  ],
  "query": "A vehicle with wheels"
}
```

Code 4.10 – So, in our example, the following values would apply:

- `0` = plane
- `1` = boat
- `2` = spaceship
- `3` = car

Knowing that each document is associated with a numeric value, when you look at the following results returned from the search API, you can see that document `3` (car) got the highest score and therefore represents the document that is most semantically similar:

```
{
    "object": "list",
```

```
    "data": [
        {
            "object": "search_result",
            "document": 0,
            "score": 56.118
        },
        {
            "object": "search_result",
            "document": 1,
            "score": 46.883
        },
        {
            "object": "search_result",
            "document": 2,
            "score": 94.42
        },
        {
            "object": "search_result",
            "document": 3,
            "score": 178.947
        }
    ],
    "model": "davinci:2020-05-03"
}
```

Introduction JSON – Example

For example, the following two JSON objects are the same and both are valid:

Example 1:

```
{"question":"Is this correct? ","answer":"Yes"}
```

Example 2:

```
{
    "question" : "Is this correct?",
    "answer" : "Yes"
}
```

Chapter 5

Technical requirements

This chapter requires that you have access to the **OpenAI API**. You can request access by visiting the following URL: https://openai.com.

Links

- You can find a list of community-maintained HTTP libraries at https://beta.openai.com/docs/developer-quickstart/community-libraries.

Figures

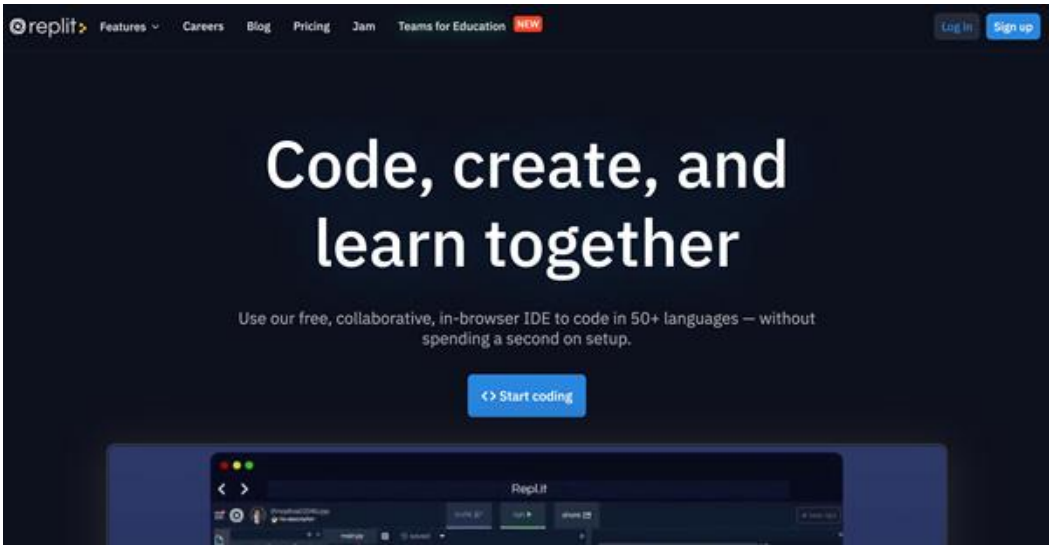

Figure 5.1 – The replit home page

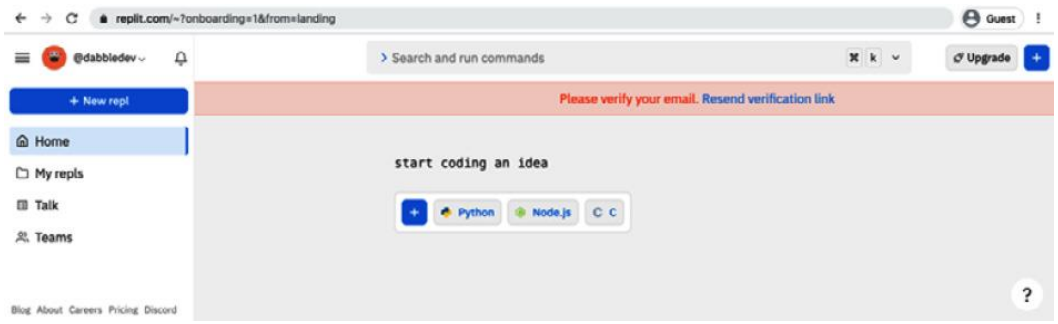

Figure 5.2 – Replit default home screen

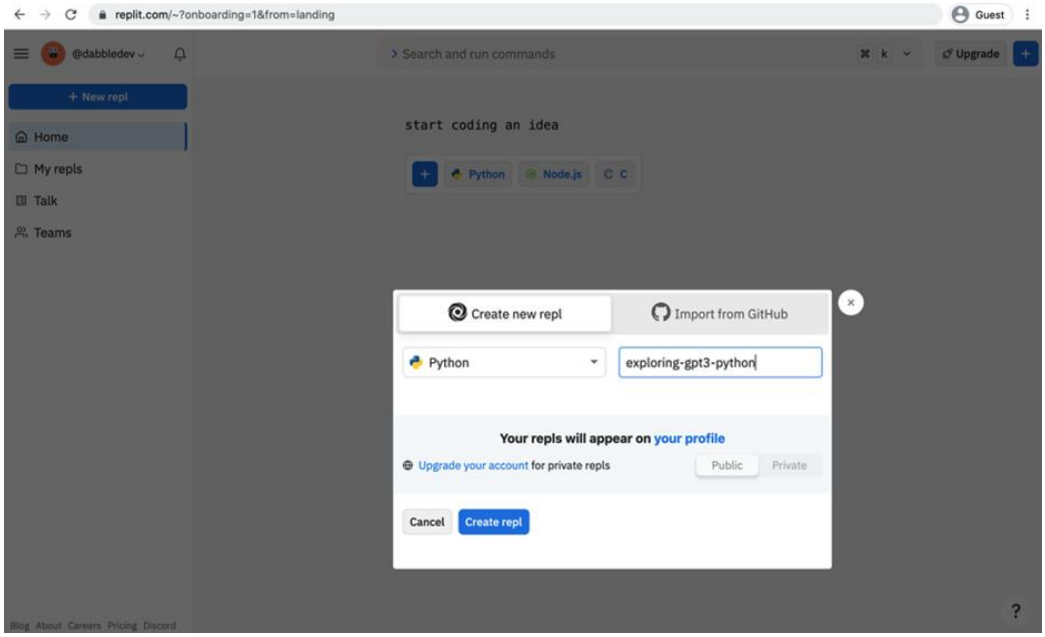

Figure 5.3 – Setting up a repl for Python

Figure 5.4 – The Replit editor with a default Python repl

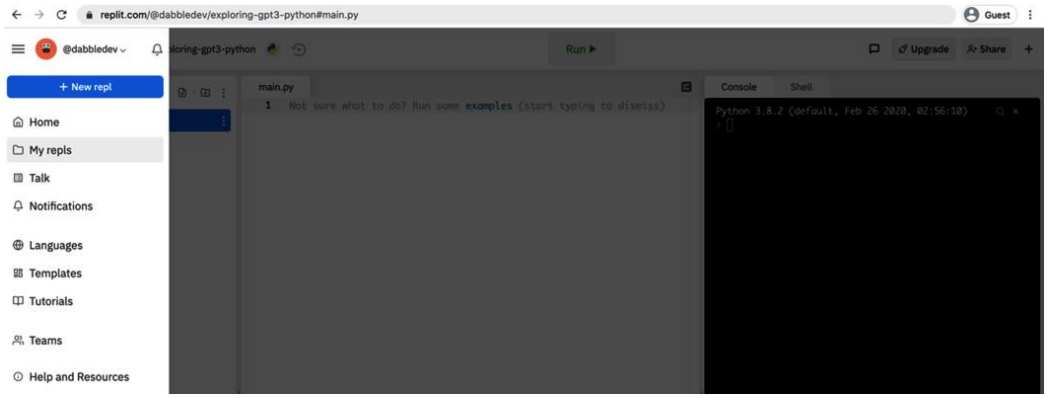

Figure 5.5 – Accessing the main navigation menu

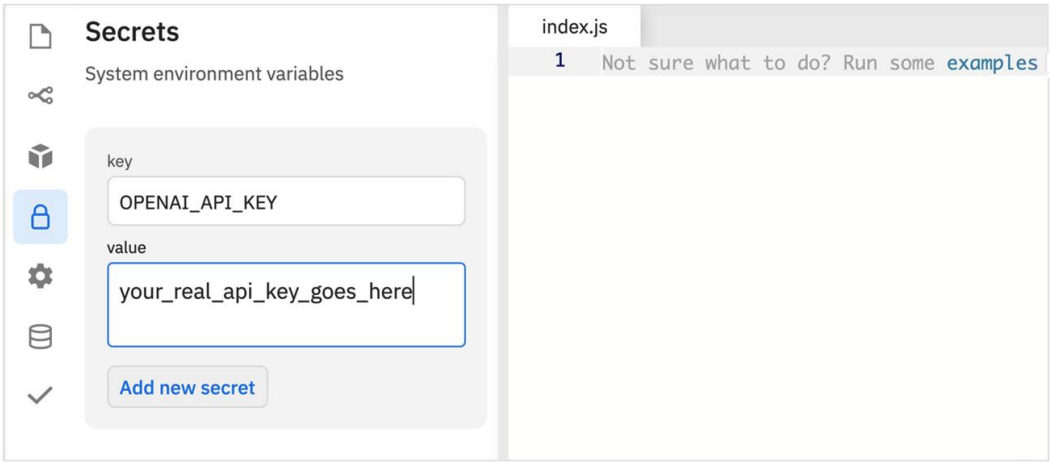

Figure 5.6 – Adding a new secret/environment variable in Replit

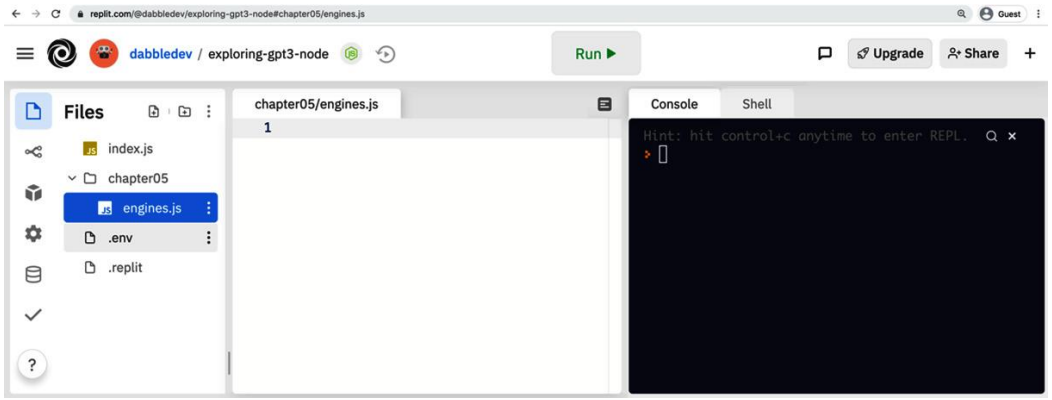

Figure 5.7 – Editing engines-dot-JS in the Replit editor

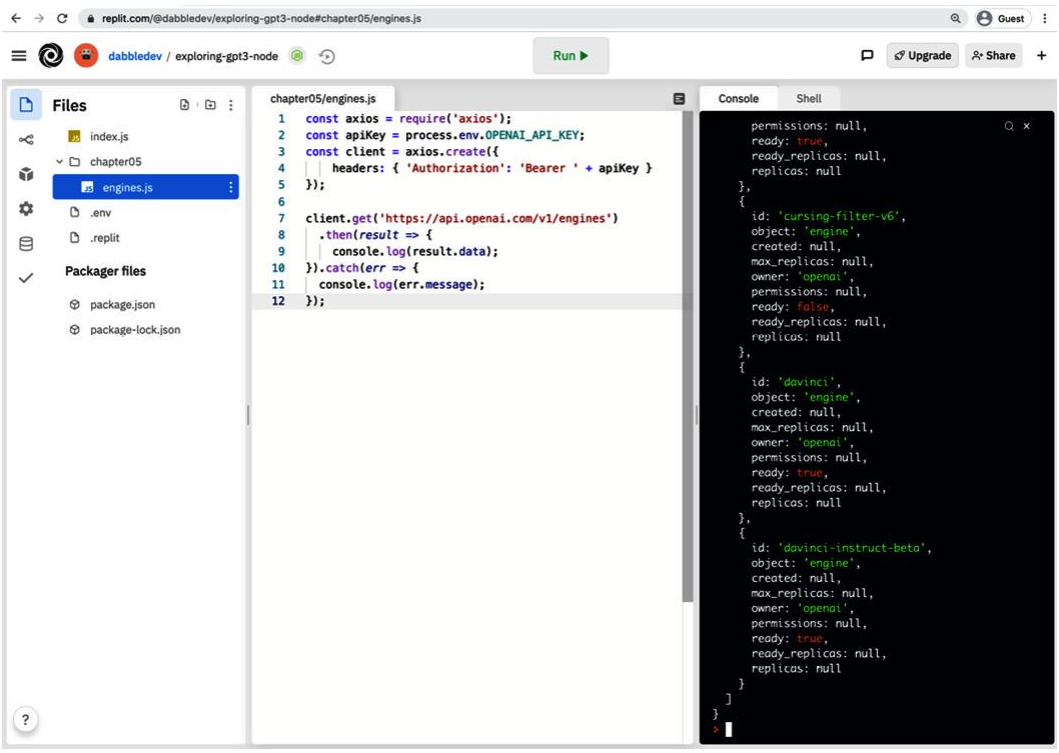

Figure 5.8 – Results from running the engines-dot-JS code

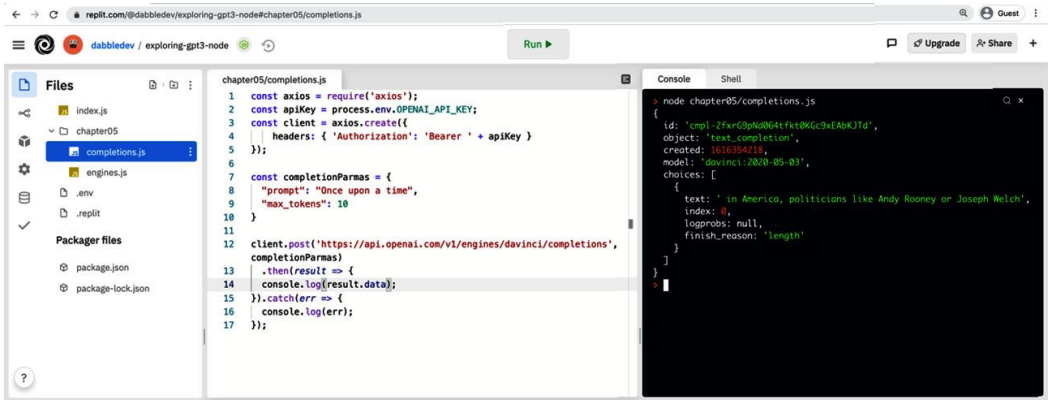

Figure 5.9 – Output from Completions.js

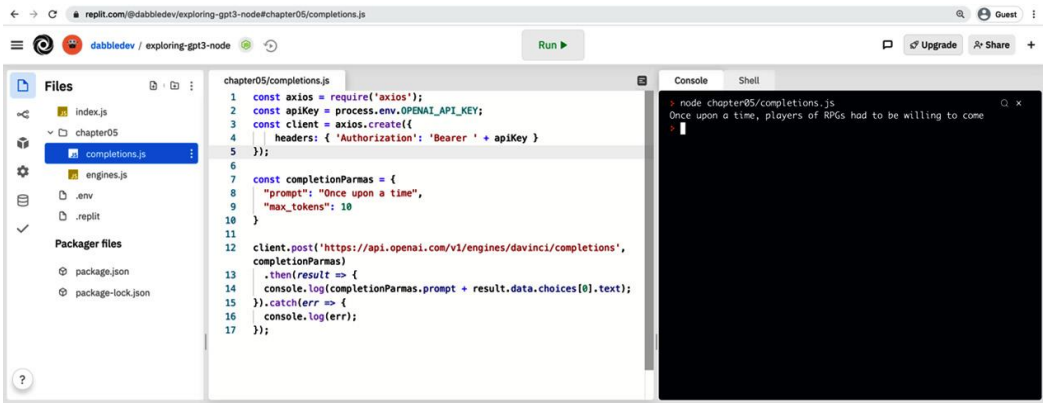

Figure 5.10 – Formatted results from completions-dot-JS

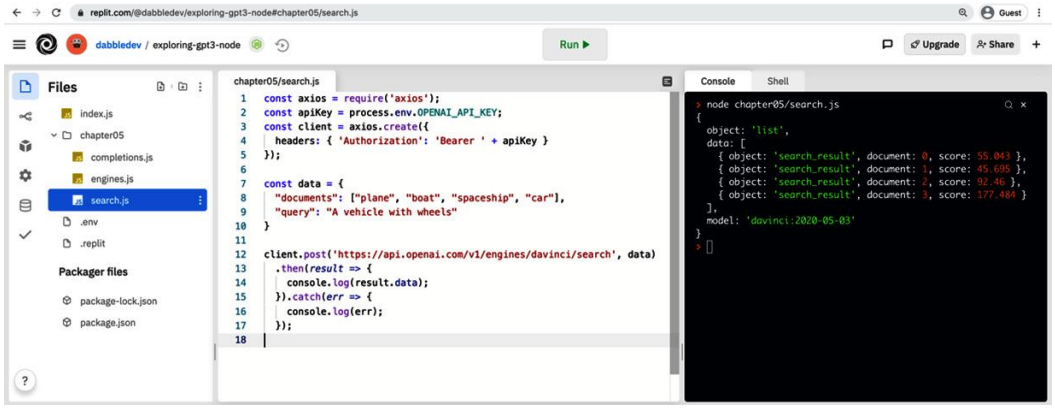

Figure 5.11 – Results from search-dot-JS

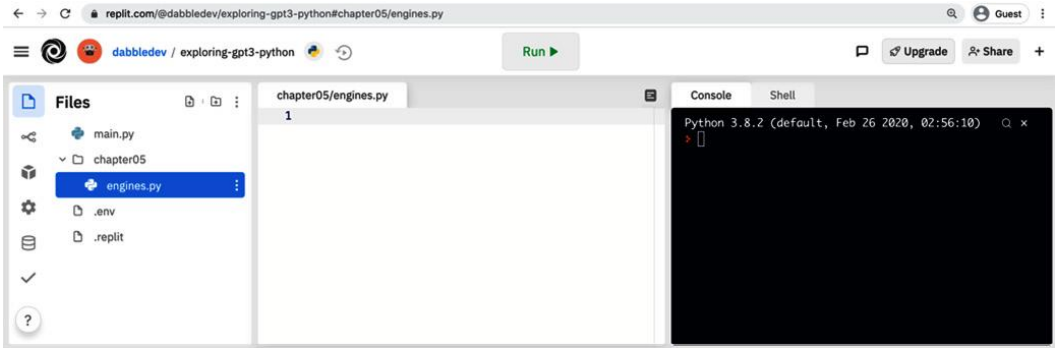

Figure 5.12 – Editing engines-dot-PY in the Replit editor

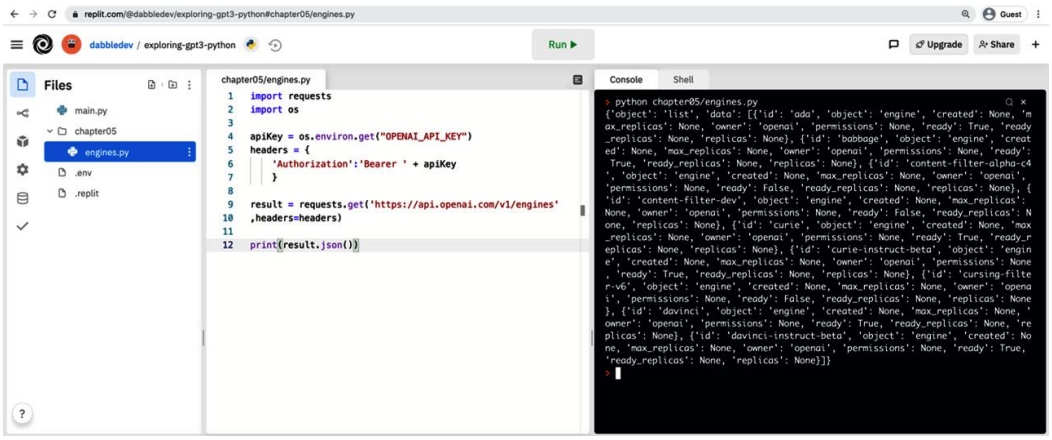

Figure 5.13 – Results from running the engines-dot-PY code

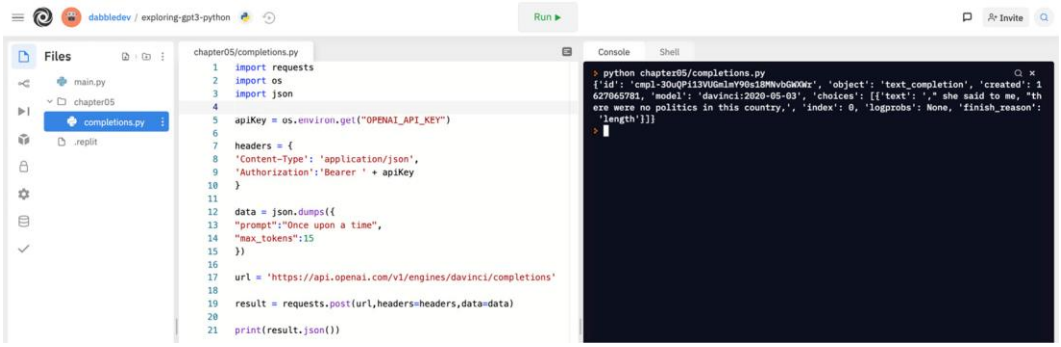

Figure 5.14 – Output from the completions-dot-JS

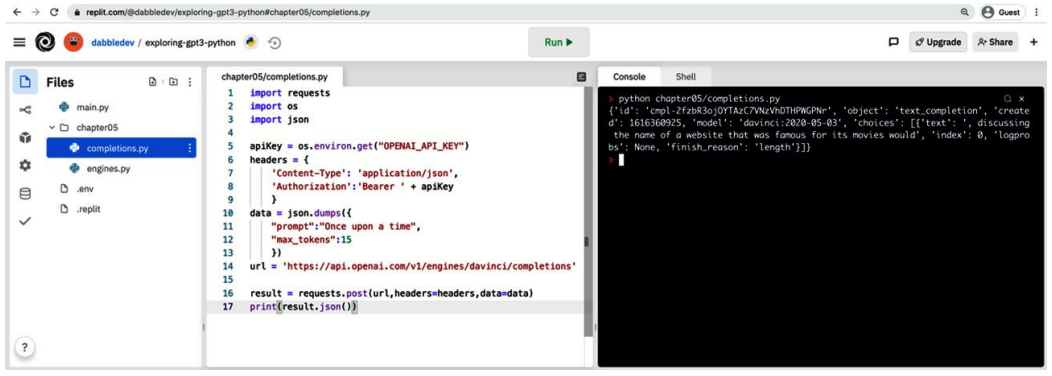

Figure 5.15 – Formatted results from completions dot-PY

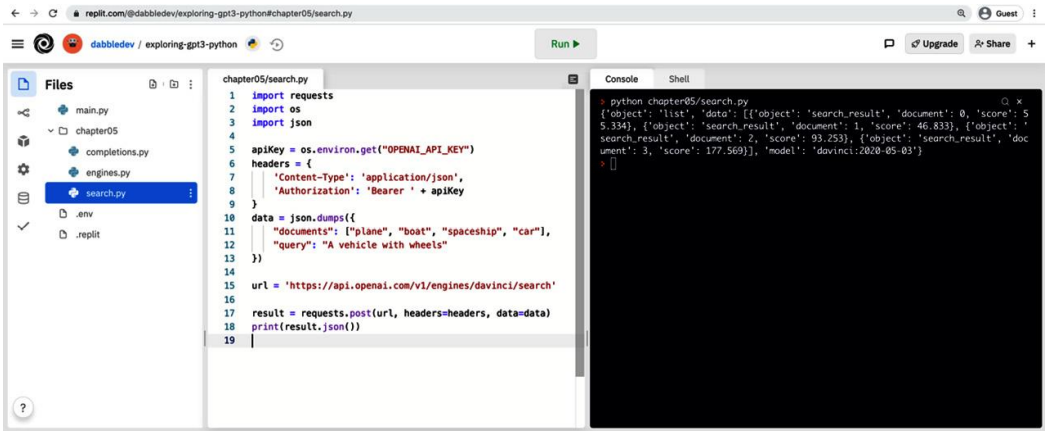

Figure 5.16 – Results from search dot-PY

Code

Calling the Engines endpoint in Node.js

Code 5.1 – Add the following text to the first line of the `.replit` file:

For Python repls:

```
run = "python main.py"
```

For Node.js repls:

```
run = "node index.js"
```

Code 5.2 – Edit the `.replit` file so the Run button executes `chapter05/engines.js` using the following text:

```
run = "node chapter05/engines.js"
```

Code 5.3 – Include a reference to the `axios` module on the first line of our `engines.js` file with the following code:

```
const axios = require('axios);
```

Code 5.4 – Add a variable to hold our OpenAI API key by pulling it from the environment variable we set up:

```
const apiKey = process.env.OPENAI_API_KEY;
```

Code 5.5 – To create the HTTP client instance, we'll add the following code:

```
const client = axios.create({
    headers: { 'Authorization': 'Bearer ' + apiKey }
});
```

Code 5.6 – All that remains is code to actually make the request. Here is what that code looks like:

```
client.get('https://api.openai.com/v1/engines')
  .then(result => {
    console.log(result.data);
}).catch(err => {
    console.log(err.message);
});
```

In the preceding code, we're using the `axios` client instance to make a GET request to the `https://api.openai.com/v1/engines` endpoint. Then, we're logging the results returned from the endpoint to the console. Or, in the case of an error, we're logging the error message to the console.

Calling the Completions endpoint in Node.js

Code 5.7 – Edit the `.replit` file so the **Run** button executes `chapter05/completions.js` using the following text:

```
run = "node chapter05/completions.js"
```

Code 5.8 – Copy the first five lines of code from `chapter05/engines.js` into `chapter05/completions.js` so your `completions.js` file starts with the following code:

```
const axios = require('axios');
const apiKey = process.env.OPENAI_API_KEY;
const client = axios.create({
    headers: { 'Authorization': 'Bearer ' + apiKey }
});
```

Code 5.9 – We're just going to be sending the `prompt` parameter and the `max_tokens` parameter. Here is what the code for that looks like:

```
const completionParams = {
  "prompt": "Once upon a time",
  "max_tokens": 10
}
```

Code 5.10 – All that's left is to actually make the request. This time we'll be making an HTTP POST and passing our parameters:

```
client.post('https://api.openai.com/v1/engines/davinci/completions',
completionParams)
  .then(result => {
  console.log(result.data);
}).catch(err => {
  console.log(err);
});
```

Code 5.11 – You could update `console.log(result.data)` with the following code to display the original prompt text:

```
console.log(completionParams.prompt + result.data.choices[0].text);
```

Calling the Search endpoint in Node.js

Code 5.12 – Edit the `.replit` file so the **Run** button executes `chapter05/search.js` using the following `run` command:

```
run = "node chapter05/completions.js"
```

Code 5.13 – Next, copy the first five lines of code from `chapter05/engines.js` into `chapter05/search.js` so your `search.js` file starts with the following code:

```
const axios = require('axios');
const apiKey = process.env.OPENAI_API_KEY;
const client = axios.create({
```

```
    headers: { 'Authorization': 'Bearer ' + apiKey }
});
```

Code 5.14 – The next thing we'll do is add what will become the JSON object that is sent in the HTTP body:

```
const data = {
  "documents": ["plane", "boat", "spaceship", "car"],
  "query": "A vehicle with wheels"
}
```

Code 5.15 – All that's left is to actually make the request, we'll be making an HTTP POST and passing our parameters:

```
client.post('https://api.openai.com/v1/engines/davinci/search', data)
  .then(result => {
    console.log(result.data);
  }).catch(err => {
    console.log(err);
  });
```

Calling the Engines endpoint in Python

Code 5.16 – Edit the `.replit` file so the **Run** button executes `chapter05/engines.py` using the following text:

```
run = "python chapter05/engines.py"
```

Code 5.17 – Include a reference to the `requests` library on the first line of our `engines.py` file with the following code:

```
import requests
```

Code 5.18 – Add some code to get the API key that we set up in the `.env` file and save it to a variable:

```
import os
apiKey = os.environ.get("OPENAI_API_KEY")
```

Code 5.19 – Create a variable named `headers` to hold our authorization information, which will be required to make the HTTP request:

```
headers = {
'Authorization':'Bearer ' + apiKey
}
```

Code 5.20 – All we need to do now is make the request. We'll do that with the following code that saves the response to a variable named `result`:

```
result = requests.get('https://api.openai.com/v1/engines'
,headers=headers)
```

Code 5.21 – To display the JSON results in the console, we'll add the last line as follows:

```
print(result.json())
```

Calling the completions endpoint in Python

Code 5.22 – Edit the `.replit` file so the **Run** button executes `chapter05/completions.py` using the following text:

```
run = "python chapter05/completions.py"
```

Code 5.23 – Your `completions.py` file should start with the following code:

```
import requests
import os
import json
apiKey = os.environ.get("OPENAI_API_KEY")
headers = {
'Content-Type': 'application/json',
'Authorization':'Bearer ' + apiKey
}
```

Code 5.24 – Add what will become the JSON object that is sent in the HTTP body. We're going to be sending the `prompt` parameter and the `max_tokens` parameter:

```
data = json.dumps({
"prompt":"Once upon a time",
"max_tokens":15
})
```

Code 5.25 – To make the code a bit more readable, we'll also create a variable for the endpoint URL with the following code:

```
url = 'https://api.openai.com/v1/engines/davinci/completions'
```

Code 5.26 – All that's left is to actually make the request and print out the results. This time, we'll be making an HTTP POST and passing our parameters:

```
result = requests.post(url,headers=headers,data=data)
print(result.json())
```

Code 5.27 – Update `console.log(result.data)` with the following code to display the original prompt text `Once upon a time` and the completion text returned by the API:

```
print(json.loads(data)["prompt"] +
result.json()["choices"][0]["text"])
```

Code 5.28 – Edit the `.replit` file so the run button executes `chapter05/search.py` using the following `run` command:

```
run = "python chapter05/search.py"
```

Code 5.29 – Copy the first nine lines of code from `chapter05/completions.py` into `chapter05/search.py` so your `search.py` file starts with the following code:

```
import requests
import os
import json
apiKey = os.environ.get("OPENAI_API_KEY")
headers = {
```

```
'Content-Type': 'application/json',
'Authorization': 'Bearer ' + apiKey
}
```

Code 5.30 – Add what will become the JSON object that is sent in the HTTP body:

```
data = json.dumps({
"documents": ["plane", "boat", "spaceship", "car"],
"query": "A vehicle with wheels"
})
```

Code 5.31 – We'll add a variable for the endpoint URL just to keep the code a bit easier to read:

```
url = 'https://api.openai.com/v1/engines/davinci/search'
```

Code 5.32 – Finally, we'll add code to make the request and print out the results:

```
result = requests.post(url, headers=headers, data=data)
print(result.json())
```

Chapter 6

Technical requirements

This chapter requires that you have access to the **OpenAI API**. You can request access by visiting https://openai.com/.

Prompt Format

The prompt format is `"<|endoftext|>[prompt]\n--\nLabel:"`. The `[prompt]` part would just be replaced with the text we want the content filter to evaluate.

Code

Code 6.1 – To illustrate, here are two example prompts and associated completions for a hypothetical customer service question. In the example, `#$%@!` is used in place of the expletives (swear words) that were used in the original prompts and completions:

- Example 1:

Prompt:

```
Customer: Where the #$%@! is my order! I placed the order more than
three days ago. How long can it take to package and ship a #$%@! tee-
shirt?

Response:
```

Completion:

```
I don't know. This is my first #$%@! day on the job. If you want, I
can call you a #$%@! taxi so you can come pick up your #$%@! Order.
```

- Example 2:

Prompt:

```
Customer: Where the #$%@! is my order! I placed the order more than
three days ago. How long can it take to package and ship a #$%@! tee-
shirt?

Polite Response:
```

Completion:

```
It can take up to 5 business days for in stock t-shirts to be
processed. You ordered three t-shirts and each shirt takes a different
amount of production time to be produced. There were some
complications with your order but as of today your t-shirts should be
fully prepared for shipping.
```

Code 6.2 – So, here is an example of the JSON we'd post to the completions endpoint. In this example, the text we're evaluating is *Once upon a time*:

```
{
    "prompt": "<|endoftext|>Once upon a time\n--\nLabel:",
    "max_tokens": 1,
    "temperature": 0.0,
    "top_p": 0
}
```

Code 6.3 – It's pretty safe to assume that *Once upon a time* would be considered safe. So, if that was the text we were applying the filter to, we could expect a response that would look something like the following example, showing the text is 0 – safe:

```
{
    "id": "cmpl-2auhZQYDGJNpeyzYNwMEm5YsAAUEK",
    "object": "text_completion",
    "created": 1615150445,
    "model": "toxicity-double-18",
    "choices": [
        {
            "text": "0",
            "index": 0,
            "logprobs": null,
            "finish_reason": "length"
        }
    ]
}
```

Code 6.4 – Here is an example of a request that sends *Oh hi* as two prompts – one word for each:

```
{
    "prompt": [
"<|endoftext|>Oh\n--\nLabel:",
"<|endoftext|>hi\n--\nLabel:"
],
    "max_tokens": 1,
    "temperature": 0.0,
    "top_p": 0
}
```

Code 6.5 – Given the previous example with an array of prompts, you'll see a response that looks something like the following. Note now that there are multiple objects in the choices array – one for each word/prompt:

```
{
    "id": "cmpl-2bDTUPEzoCrtNBa2gbkpNVc1BcVh9",
    "object": "text_completion",
    "created": 1615222608,
    "model": "toxicity-double-18",
    "choices": [
        {
            "text": "0",
            "index": 0,
            "logprobs": null,
            "finish_reason": "length"
        },
        {
            "text": "0",
            "index": 1,
            "logprobs": null,
            "finish_reason": "length"
    ]
```

```
}
```

Code 6.6 – Add the following JSON object to the request body:

```
{
  "prompt" : "<|endoftext|>Are you religious?\n--\nLabel:",
  "max_tokens" : 1,
  "temperature" : 0.0,
  "top_p" : 0
}
```

Code 6.7 – So, after updating the authorization line, the final code should be the following:

```
var axios = require('axios');
var data = JSON.stringify({
  "prompt": "<|endoftext|>What religion are you?\n--\nLabel:",
  "max_tokens": 1,
  "temperature": 0,
  "top_p": 0
});
var config = {
  method: 'post',
  url: 'https://api.openai.com/v1/engines/content-filter-alpha-
c4/completions',
  headers: {
    'Authorization': `Bearer ${process.env.OPENAI_API_KEY}`,
    'Content-Type': 'application/json'
  },
  data : data
};
axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
```

```
.catch(function (error) {
  console.log(error);
});
```

Code 6.8 – So, update the `.replit` file to the following:

```
Run "node chapter06/filter.js"
```

Code 6.9 – Edit the `.replit` file to run `chapter06/flag.js` by using the following command:

```
Run = "node chapter06/flag.js"
```

Code 6.10 – We'll start by adding a variable to hold the text we want to filter. We'll add this code just under the first line. So, the first two lines will be as follows:

```
var axios = require('axios');
const textInput = "This is some text that will be filtered";
```

Code 6.11 – Next, we'll add a variable to hold an array of prompts and set the initial value to an empty array. This will get populated with a prompt for each word from our text input:

```
const prompts = [];
```

Code 6.12 – Now, we'll split our `textInput` into an array of words and populate the `prompts` array with a prompt for each word:

```
const wordArray = textInput.split(' ');
for (i = 0, len = wordArray.length, text = ""; i < len; i++) {
  text = `<|endoftext|>${wordArray[i]}\n--\nLabel:`;
  prompts.push(text);
}
```

Code 6.13 – Now we will update the data variable that was created by Postman. We'll use our `prompts` array as the prompt value rather than the hardcoded value from Postman. So, we'll change the data variable to the following:

```
var data = JSON.stringify({"prompt":
prompts,"max_tokens":1,"temperature":0,"top_p":0});
```

Code 6.14 – Finally, we'll modify the output with code that loops through the word array and classifies each word using the results from the filter. To do that, replace the line that contains `console.log(JSON.stringify(response.data));` with the following code:

```
    response.data.choices.forEach(item => {
      console.log(`${wordArray[item.index]} : ${item.text}`);
```

Code 6.15 – Add the following code to the first line of our `filter.py` file:

```
import os
```

Code 6.16 – In the preceding *Figure 6.12*, you would be editing *line 7* to the following:

```
'Authorization':'Bearer ' + os.environ.get("OPENAI_API_KEY")
```

Code 6.17 – After updating the authorization line, the final code should be the following:

```
import os
import requests
import json
url = "https://api.openai.com/v1/engines/content-filter-alpha-
c4/completions"
payload = json.dumps({
  "prompt": "<|endoftext|>What religion are you?\n--\nLabel:",
  "max_tokens": 1,
  "temperature": 0,
  "top_p": 0
})
headers = {
  'Authorization':'Bearer ' + os.environ.get("OPENAI_API_KEY"),
  'Content-Type': 'application/json'
}
response = requests.request("POST", url, headers=headers,
data=payload)
print(response.text)
```

Code 6.18 – So, update the `.replit` file to the following:

```
Run "python chapter06/filter.py"
```

Code 6.19 – Edit the `.replit` file to run `chapter06/flag.py` using the following command:

```
Run = "python chapter06/flag.py"
```

Code 6.20 – In the `chapter06/flag.py` file, we'll add a variable to hold the text we want to filter. We'll add the following code just under the third line (after the last line that starts with `import`):

```
textInput = "What religion are you?"
```

Code 6.21 – Next, we'll add a variable to hold an array of prompts and set the initial value to an empty array. This will get populated with a prompt for each word from our text input:

```
prompts = []
```

Code 6.22 – Now, we'll split our `textInput` into an array of words and populate the `prompts` array with a prompt for each word. Since we're sending the prompts to the filter engine, we'll also need to format each prompt item properly. So, we'll add the following code after our `prompts` variable. This code splits the text input into individual words, loops through each word to create a prompt item, and adds the prompt item to the `prompts` array:

```
wordArray = textInput.split()
for word in wordArray:
  prompts.append("<|endoftext|>" + word + "\n--\nLabel:")
```

Code 6.23 – Now we will update the payload variable that was created by Postman to a Python object rather than a string. This makes it a little more readable and easier to include our `prompts` array. So, replace the payload variable with the following code:

```
payload = json.dumps({
  "prompt" : prompts,
  "max_tokens" : 1,
  "temperature" : 0.0,
  "top_p" : 0
  })
```

Code 6.24 – Finally, we'll replace the last line of code, `print(response.text)`, with the following code that loops through the results and adds a classification (0 = safe, 1 = sensitive, 2 = unsafe) for each word:

```
for word in response.json()['choices']:
  print(wordArray[word['index']] + ' : ' + word['text'])
```

Figures

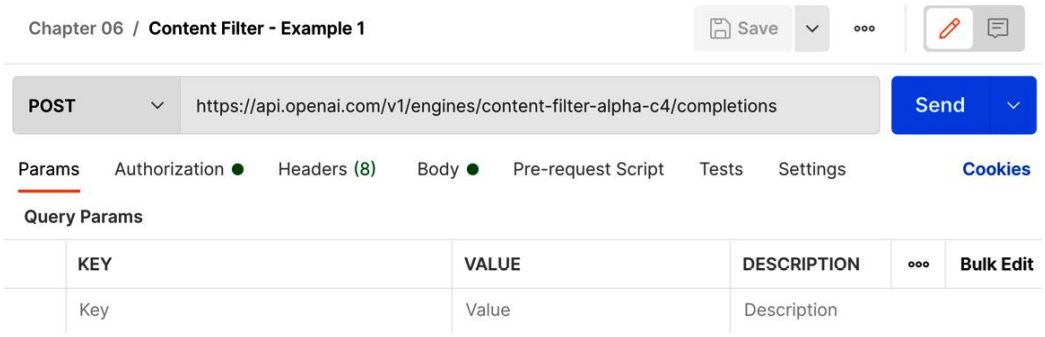

Figure 6.1 – Setting the filter endpoint in Postman

Figure 6.2 – Filter parameters in Postman

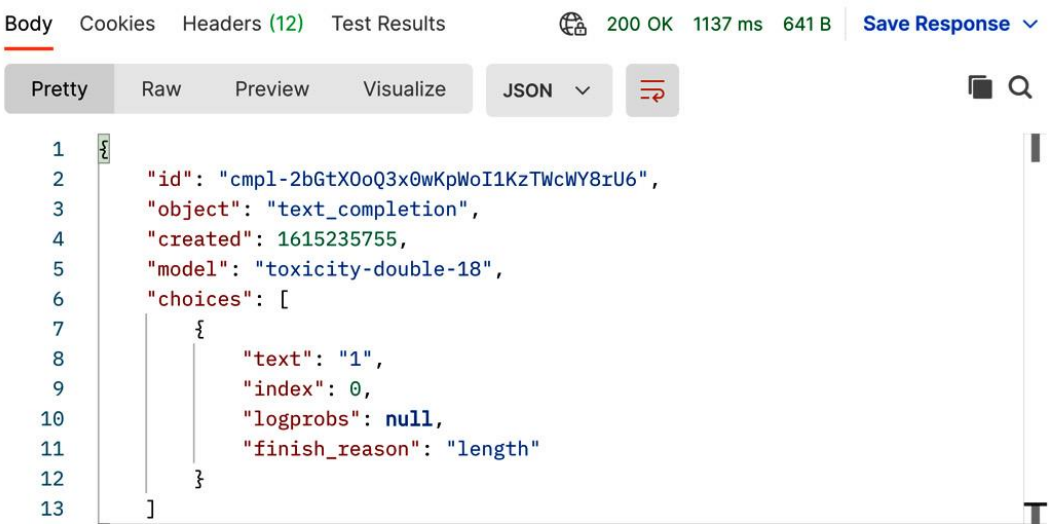

Figure 6.3 – Postman filter results

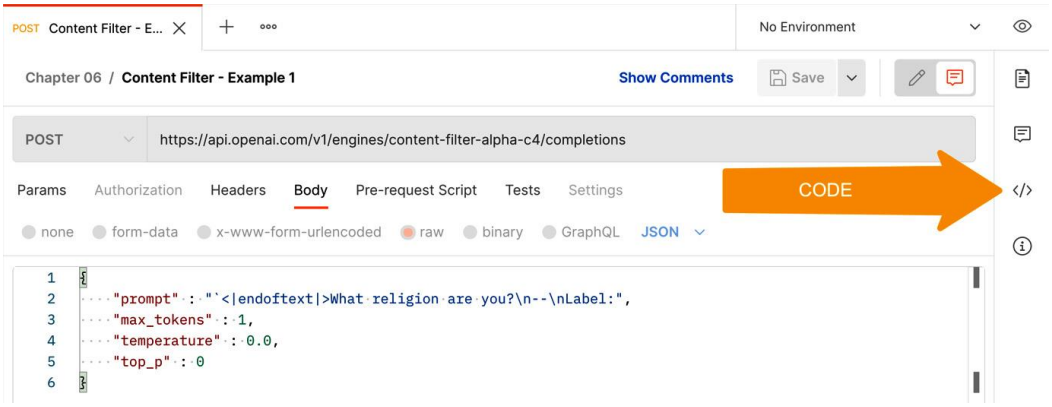

Figure 6.4 – Code button to open the code pane

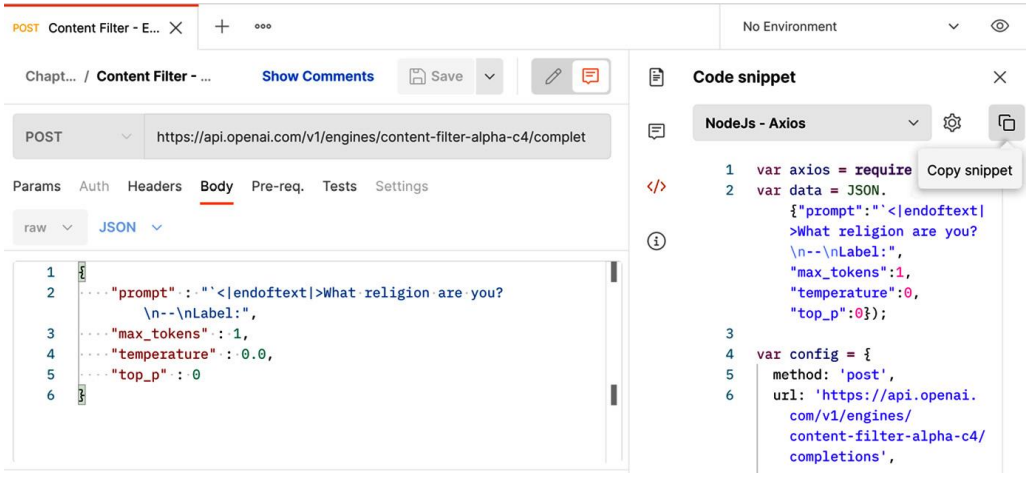

Figure 6.5 – Postman code snippet for Node.js – Axios

chapter06/filter.js

```
var axios = require('axios');
var data = JSON.stringify({"prompt":"`<|endoftext|>What religion are you?\n--\nLabel:","max_tokens":1,"temperature":0,"top_p":0});

var config = {
  method: 'post',
  url: 'https://api.openai.com/v1/engines/content-filter-alpha-c4/completions',
  headers: {
    'Authorization': 'Bearer {{OPENAI_API_KEY}}',
    'Content-Type': 'application/json'
  },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Figure 6.6 – Code copied from the Postman snippet to the replit.com file

chapter06/filter.js

```
var axios = require('axios');
var data = JSON.stringify({"prompt":"`<|endoftext|>What religion are you?\n--\nLabel:","max_tokens":1,"temperature":0,"top_p":0});

var config = {
  method: 'post',
  url: 'https://api.openai.com/v1/engines/content-filter-alpha-c4/completions',
  headers: {
    'Authorization': `Bearer ${process.env.OPENAI_API_KEY}`,
    'Content-Type': 'application/json'
  },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Figure 6.7 – Postman code snippet modified for replit.com

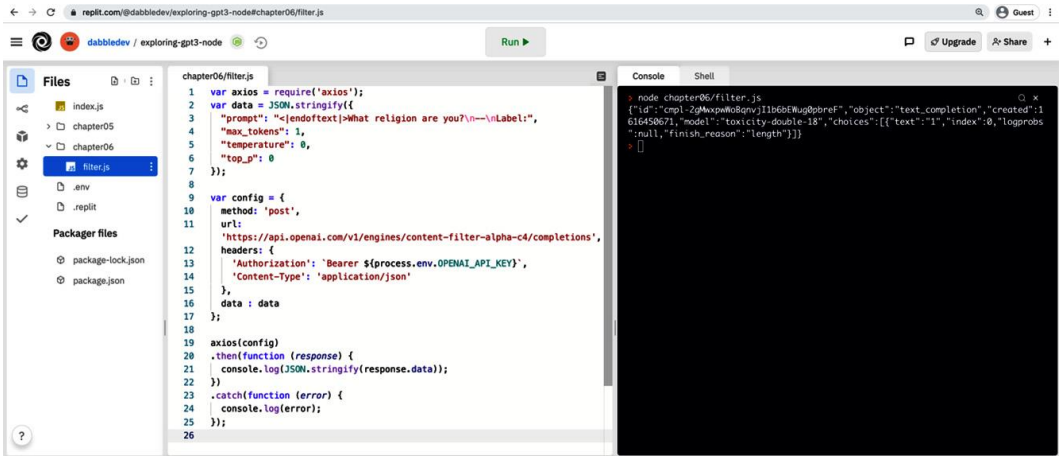

Figure 6.8 – Results from running chapter06/filter.js

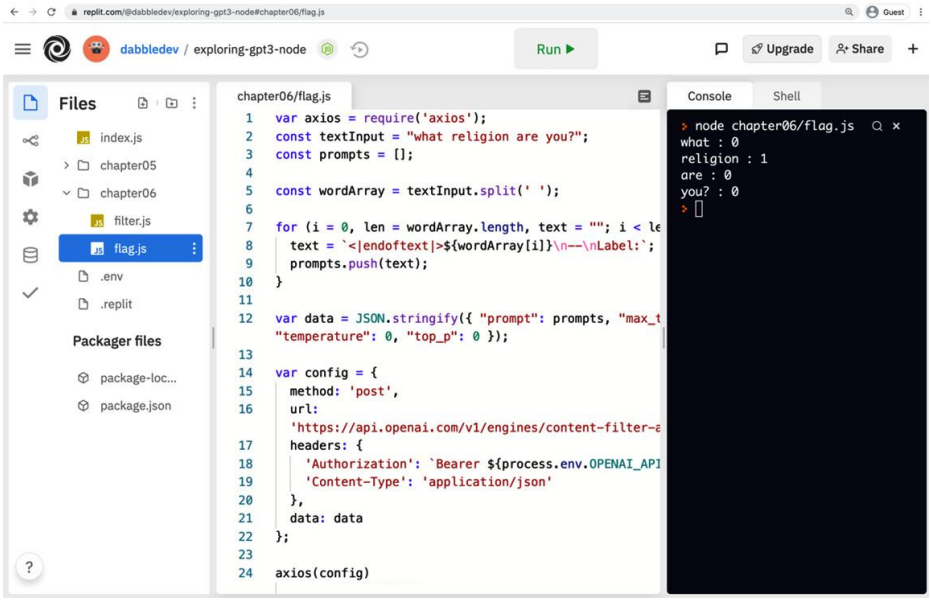

Figure 6.9 – Content filter results for each word in a text input

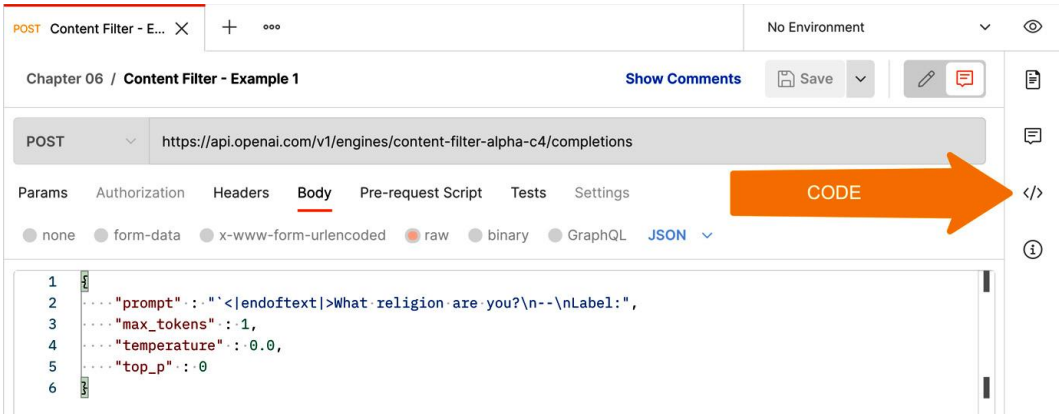

Figure 6.10 – Code button to open the code pane

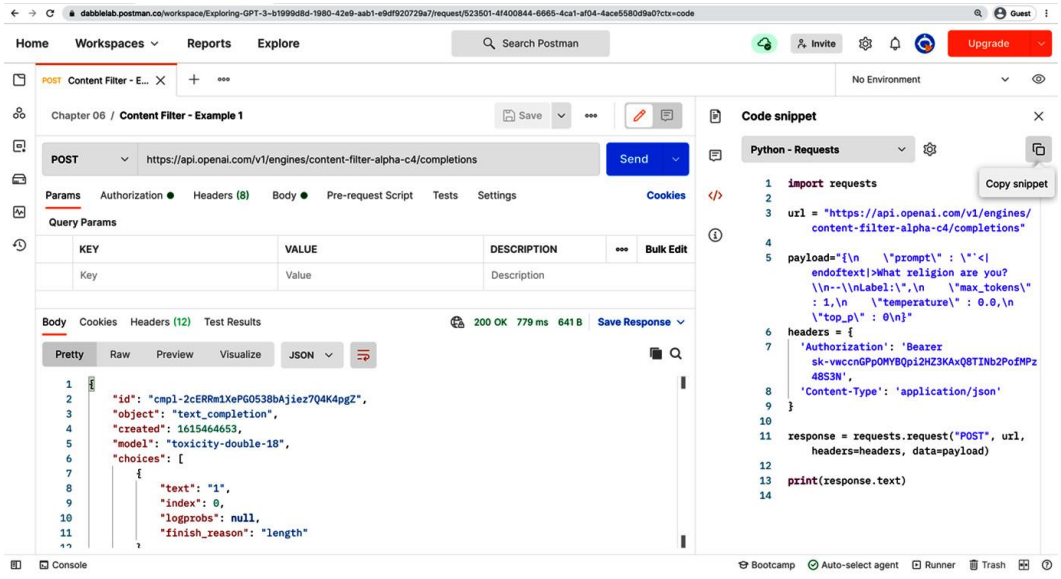

Figure 6.11 – Postman code snippet for Python – requests

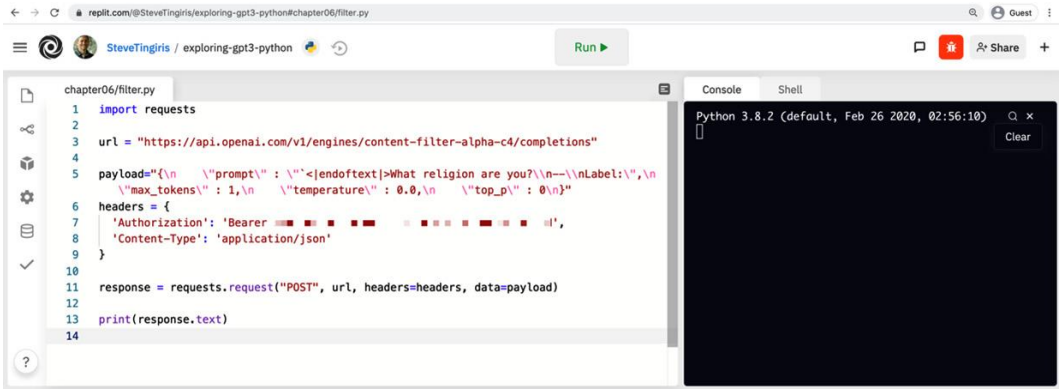

Figure 6.12 – Code copied from the Postman snippet to the repl.it file

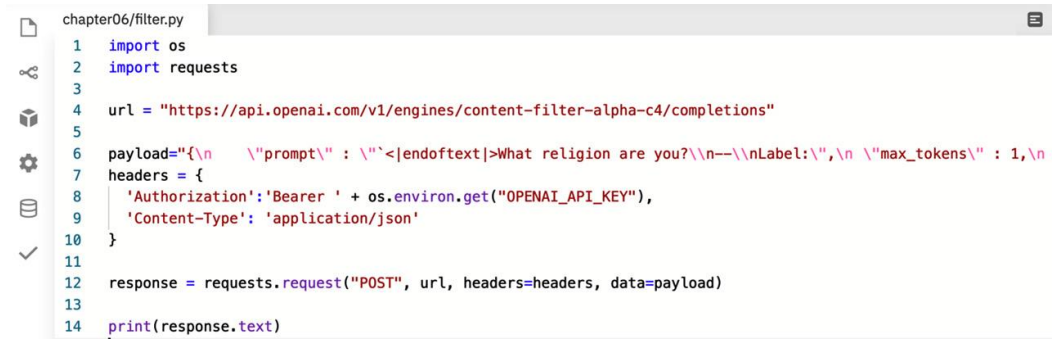

Figure 6.13 – Postman Python code snippet modified for replit.com

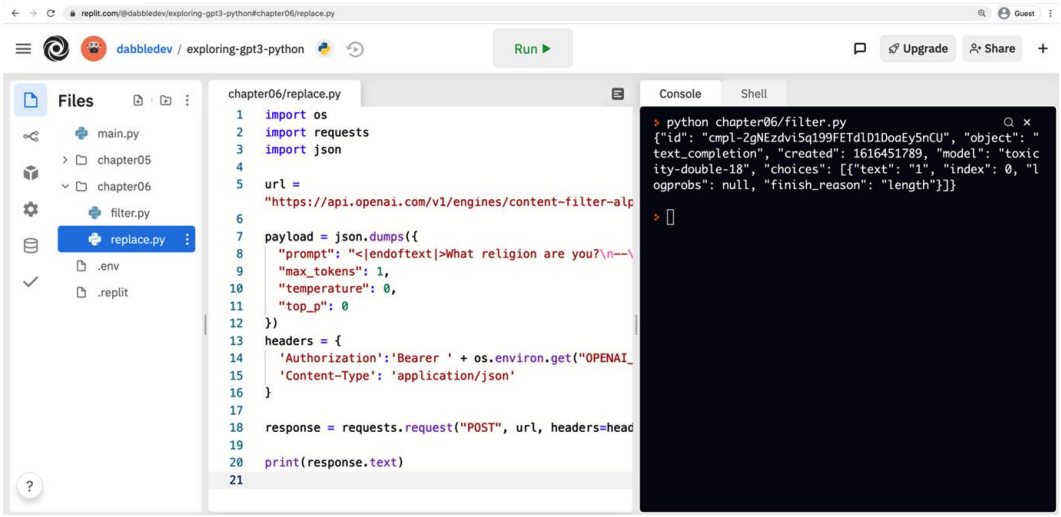

Figure 6.14 – Results from running chapter06/filter.py

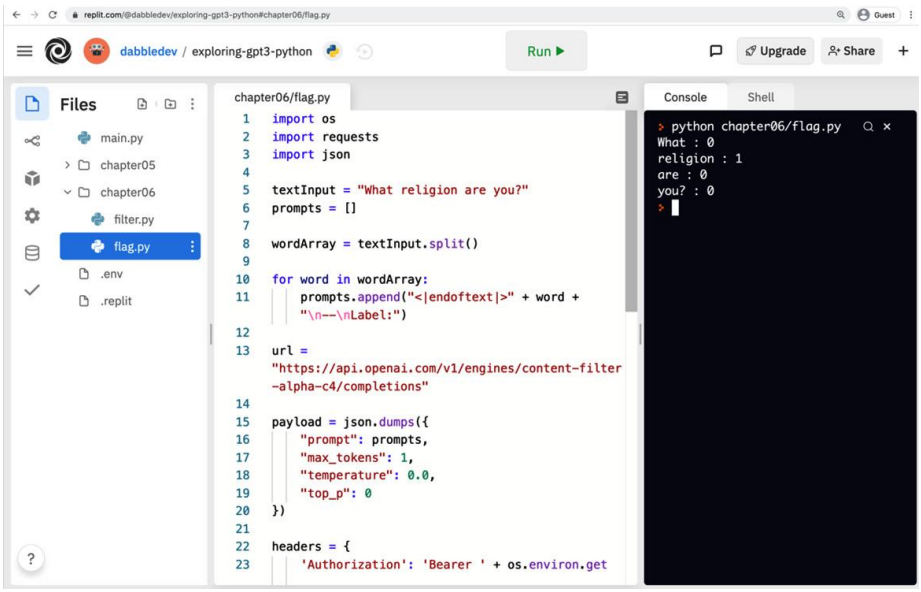

Figure 6.15 – Content filter results for each word in a text input using Python

URL

- The URL we'd use for the completions endpoint with that engine would be https://api.openai.com/v1/engines/content-filter-alpha-c4/completions.
- Be sure to review the OpenAI content filter documents located at https://beta.openai.com/docs/engines/content-filter

Chapter 7

Technical requirements

Let's look at the requirements we need in this chapter:

- Access to the OpenAI API
- An account on replit.com

Prompts

Dumb Joke Generator

```
Dumb Joke: I'm not a vegetarian because I love animals. I'm a
vegetarian because I hate plants.

###

Two-Sentence Joke: Parallel lines have so much in common. It's a shame
they'll never meet.

###

Dumb Joke: Someone stole my mood ring. I don't know how I feel about
that.

###

Dumb Joke:
```

Mars

```
I'm studying the planets. List things I should know about Mars.

1. Mars is the nearest planet to Earth.

2. Mars has seasons, dry variety (not as damp as Earth's).

3. Mars' day is about the same length as Earth's (24.6 hours).

4.
```

Webinar description generator

```
Write a description for the following webinar:

Date: Monday, June 5, 2021
```

```
Time: 10 AM PT

Title: An introduction to mindfulness

Presenter: Gabi Calm

Event Description:
```

Book suggestions

```
Suggest a list of books that everyone should try to read in their
lifetime.

Books:

1.
```

Children's book generator

```
Write a short story for kids about a Dog named Bingo who travels to
space.

---

Page 1: Once upon a time there was a dog named Bingo.

Page 2: He was trained by NASA to go in space.
```

Acronym translator

```
Provide the meaning for the following acronym.

---

acronym: LOL

meaning: Laugh out loud

acronym: BRB

meaning: Be right back

acronym: L8R

meaning:
```

English to Spanish

```
Translate from English to Spanish

---

English: Where is the bathroom?

Spanish:
```

JavaScript to Python

```
Translate from JavaScript to Python

JavaScript:

const request = require("requests");

request.get("https://example.com");

Python:
```

Fifth-grade summary

```
Summarize the following passage for me as if I was in fifth grade:

"""

Quantum mechanics is a fundamental theory in physics that provides a
description of the physical properties of nature at the scale of atoms
and subatomic particles. It is the foundation of all quantum physics
including quantum chemistry, quantum field theory, quantum technology,
and quantum information science.

Classical physics, the description of physics that existed before the
theory of relativity and quantum mechanics, describes many aspects of
nature at an ordinary (macroscopic) scale, while quantum mechanics
explains the aspects of nature at small (atomic and subatomic) scales,
for which classical mechanics is insufficient. Most theories in
classical physics can be derived from quantum mechanics as an
approximation valid at large (macroscopic) scale.

Quantum mechanics differs from classical physics in that energy,
momentum, angular momentum, and other quantities of a bound system are
restricted to discrete values (quantization), objects have
characteristics of both particles and waves (wave-particle duality),
and there are limits to how accurately the value of a physical
quantity can be predicted prior to its measurement, given a complete
set of initial conditions (the uncertainty principle).

"""

Here is the fifth-grade version of this passage:

"""
```

Grammar correction

```
Original: You be mistaken

Standard American English:
```

Extracting keywords

```
Quantum mechanics is a fundamental theory in physics that provides a description of the physical properties of nature at the scale of atoms and subatomic particles. It is the foundation of all quantum physics including quantum chemistry, quantum field theory, quantum technology, and quantum information science.

Classical physics, the description of physics that existed before the theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small (atomic and subatomic) scales, for which classical mechanics is insufficient. Most theories in classical physics can be derived from quantum mechanics as an approximation valid at large (macroscopic) scale.

Quantum mechanics differs from classical physics in that energy, momentum, angular momentum, and other quantities of a bound system are restricted to discrete values (quantization), objects have characteristics of both particles and waves (wave-particle duality), and there are limits to how accurately the value of a physical quantity can be predicted prior to its measurement, given a complete set of initial conditions (the uncertainty principle).

Keywords:
```

HTML parsing

Extracting a postal address

```
Extract the postal address from this email:

Dear Paul,

I'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.

Is the seller flexible at all on the asking price?

Best,

Linda

Property Address:

Extract the title, h1, and body text from the following HTML document:

<head><title>A simple page</title></head><body><h1>Hello World</h1><p>This is some text in a simple html page.</p></body></btml>
```

```
Title:
```

Extracting an email address

```
Extract the email address from the following message:

Dear Paul,

I'm in the market for a new home and I understand you're the listing agent for the property located at 2620 Riviera Dr, Laguna Beach, CA 92651.

Can you send details to my wife's email which is beth@example.com?

Best,

Kevin

Email Address:
```

A simple chatbot

```
The following is a conversation with an AI bot. The bot is very friendly and polite.

Human: Hello, how are you?

AI: I am doing great, thanks for asking. How can I help you today?

Human: I just wanting to talk with you.

AI:
```

Figures

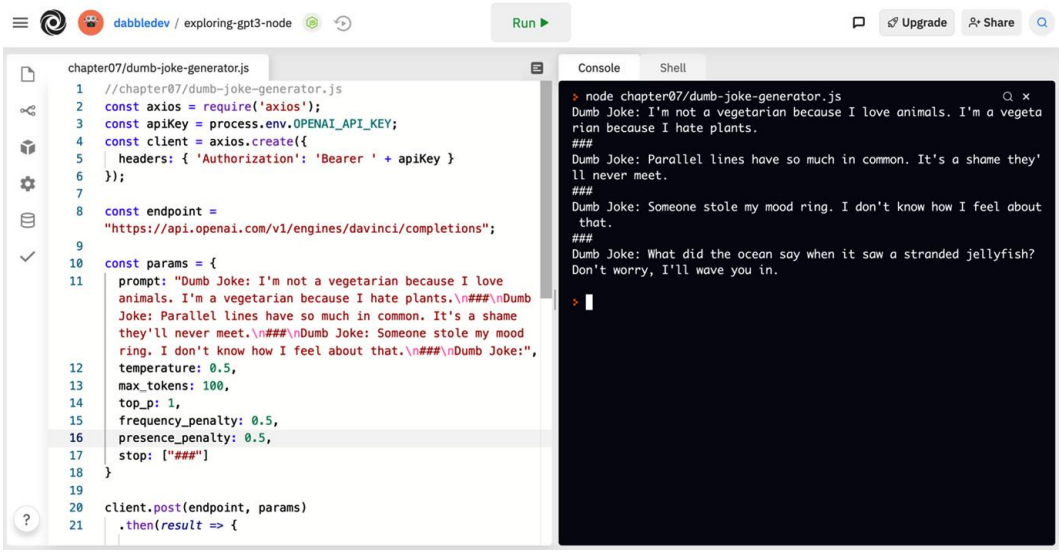

Figure 7.1 – Example output from chapter07/dumb-joke-generator.js

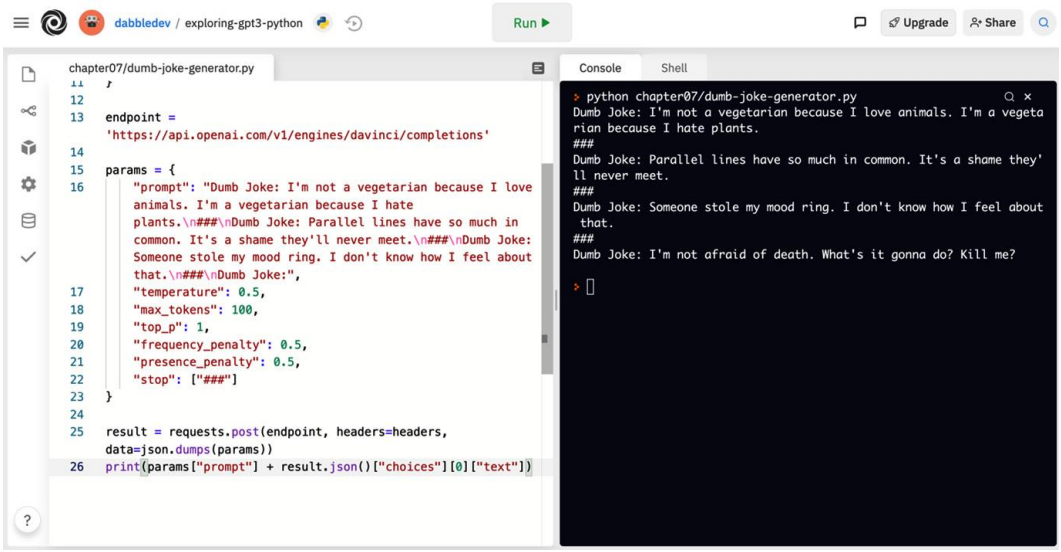

Figure 7.2 – Example output from chapter07/dumb-joke-generator.py

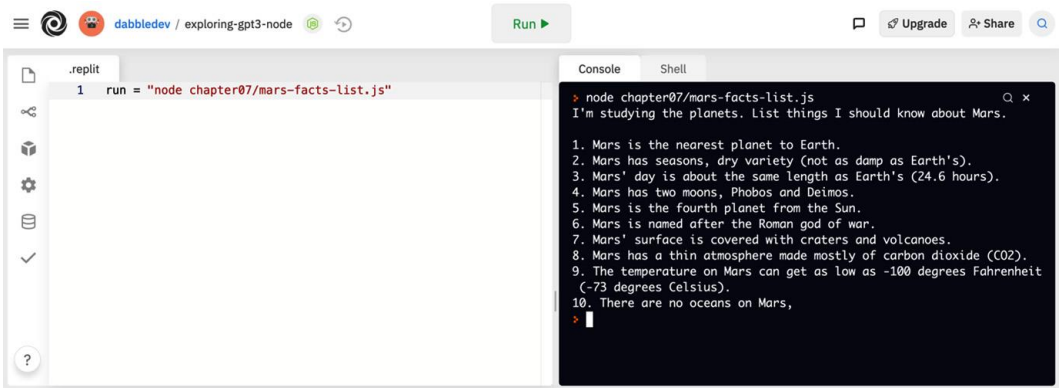

Figure 7.3 – Example output from chapter07/mars-facts-list.js

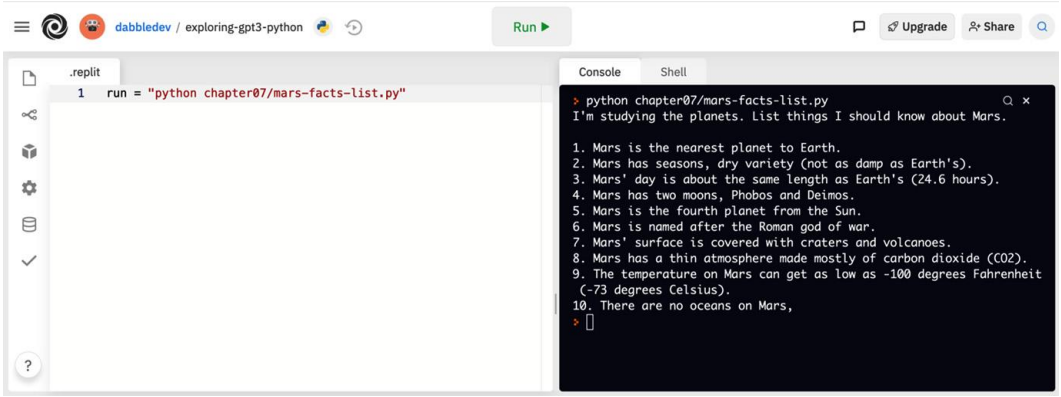

Figure 7.4. – Example output from chapter07/mars-facts-list.py

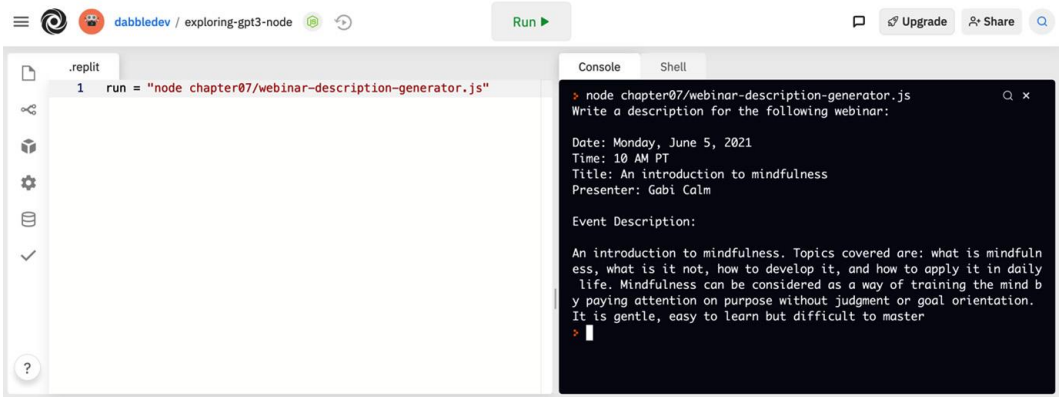

Figure 7.5 – Example output from chapter07/webinar-description-generator.js

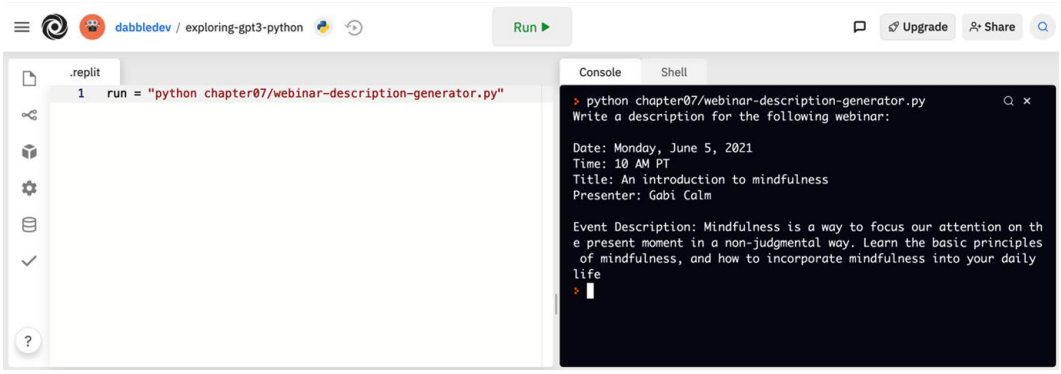

Figure 7.6 – Example output from chapter07/webinar-description-generator.py

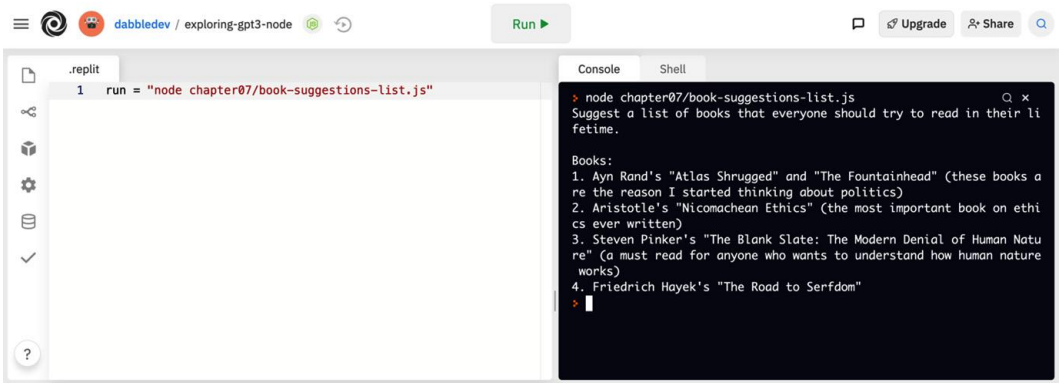

Figure 7.7 – Example output from chapter07/book-suggestions-list.js

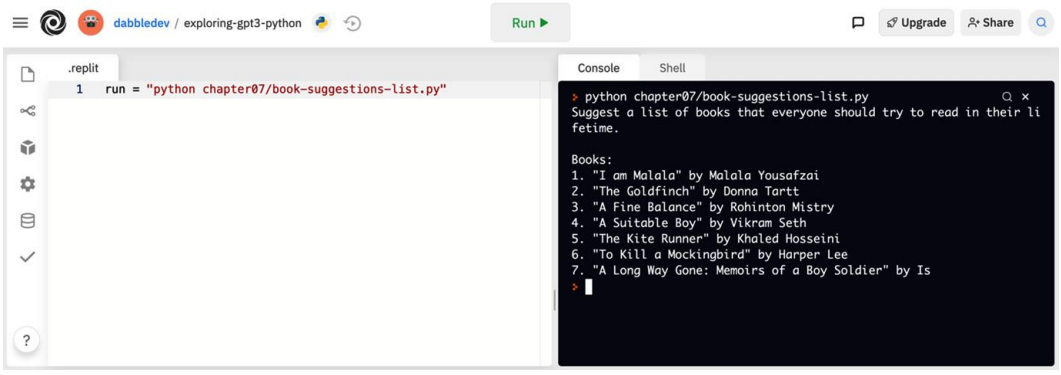

Figure 7.8 – Example output from chapter07/book-suggestions-list.py

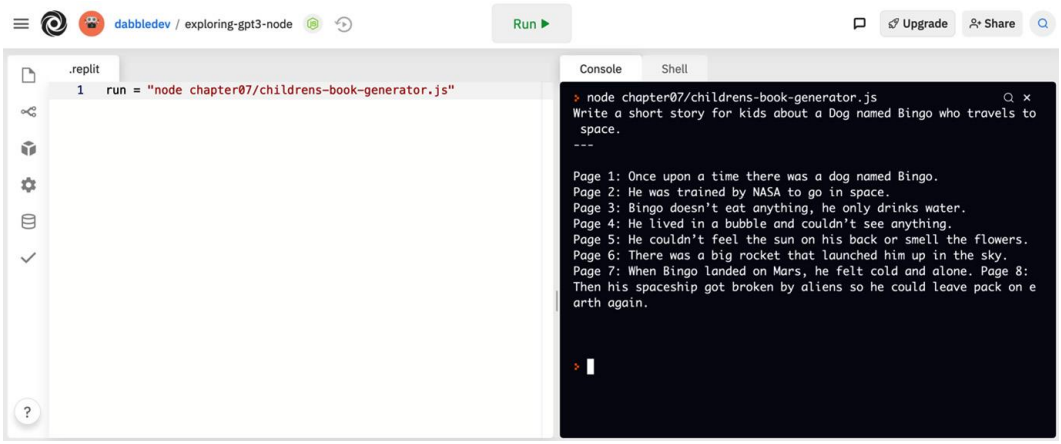

Figure 7.9 – Example output from chapter07/childrens-book-generator.js

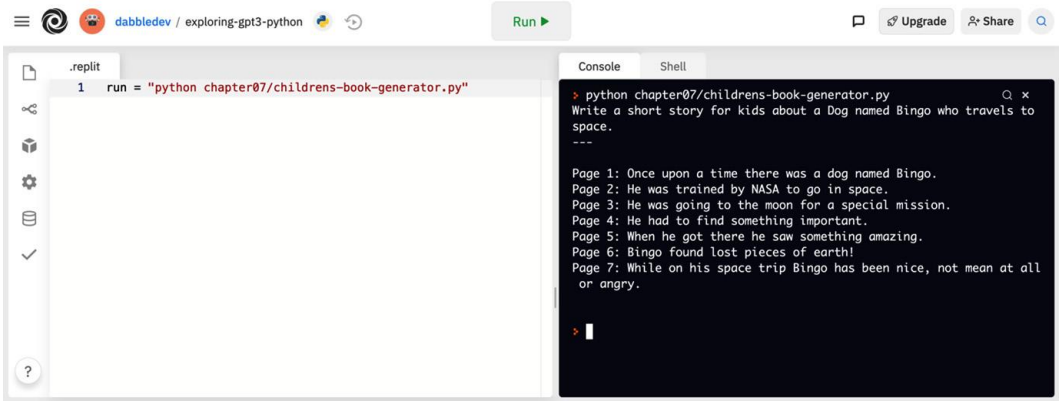

Figure 7.10 – Example output from chapter07/childrens-book-generator.py

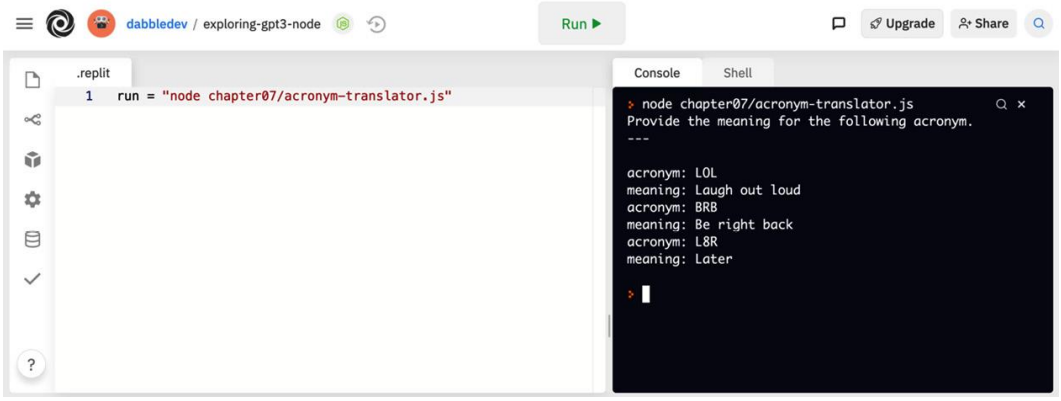

Figure 7.11 – Example output from chapter07/acronym-translator.js

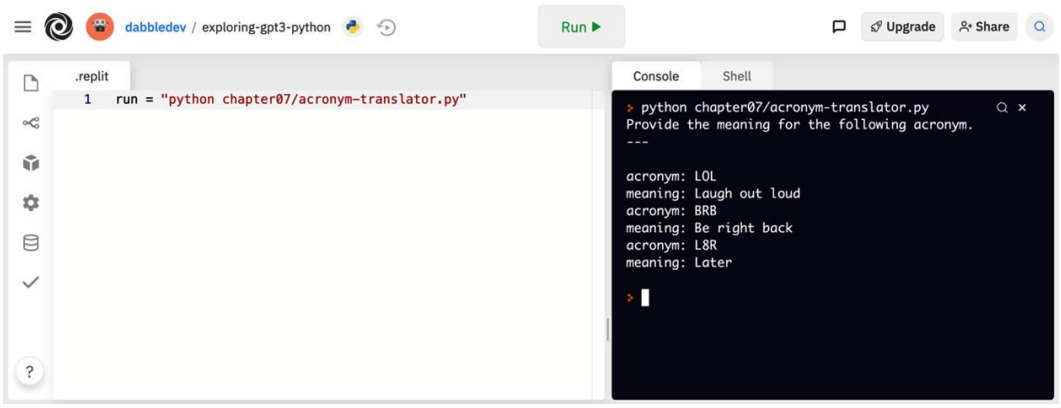

Figure 7.12 – Example output from chapter07/acronym-translator.py

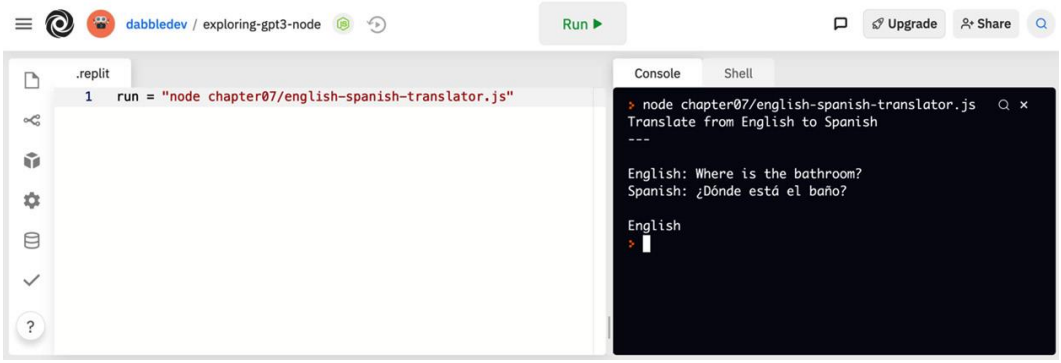

Figure 7.13 – Example output from chapter07/english-spanish-translator.js

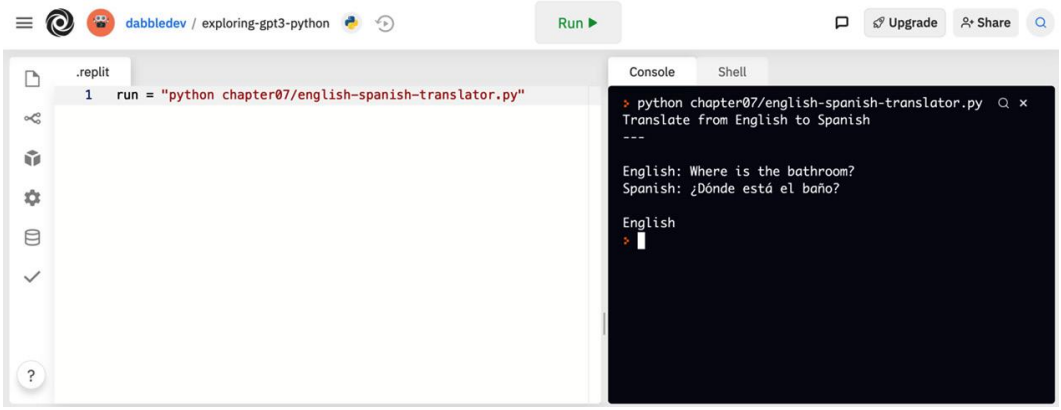

Figure 7.14 – Example output from chapter07/english-spanish-translator.py

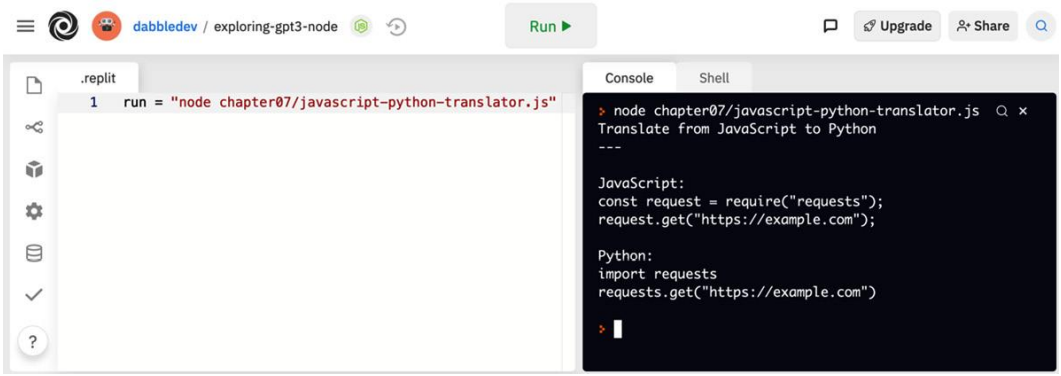

Figure 7.15 – Example output from chapter07/javascript-python-translator.js

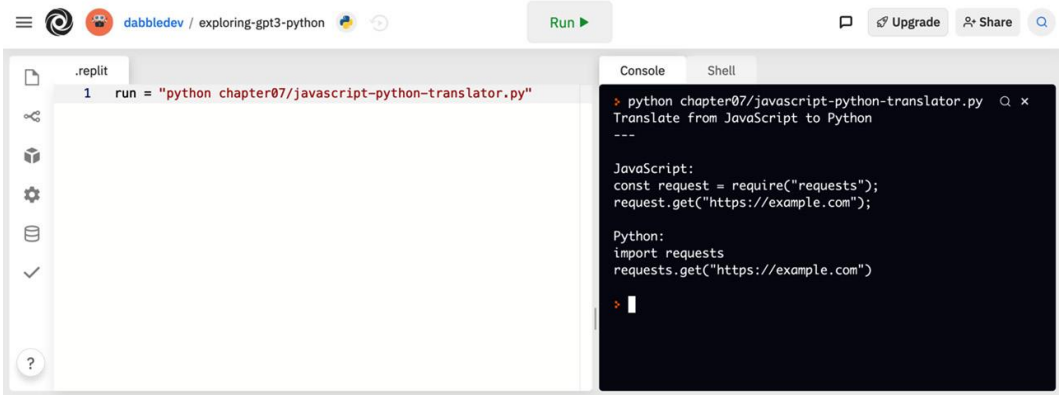

Figure 7.16 – Example output from chapter07/javascript-python-translator.py

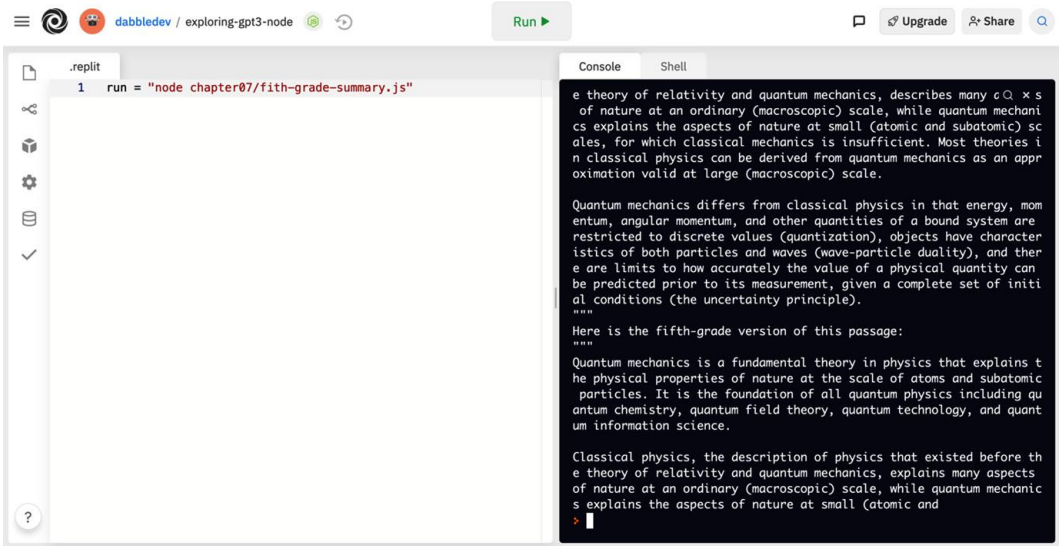

Figure 7.17 – Example output from chapter07/fith-grade-summary.js

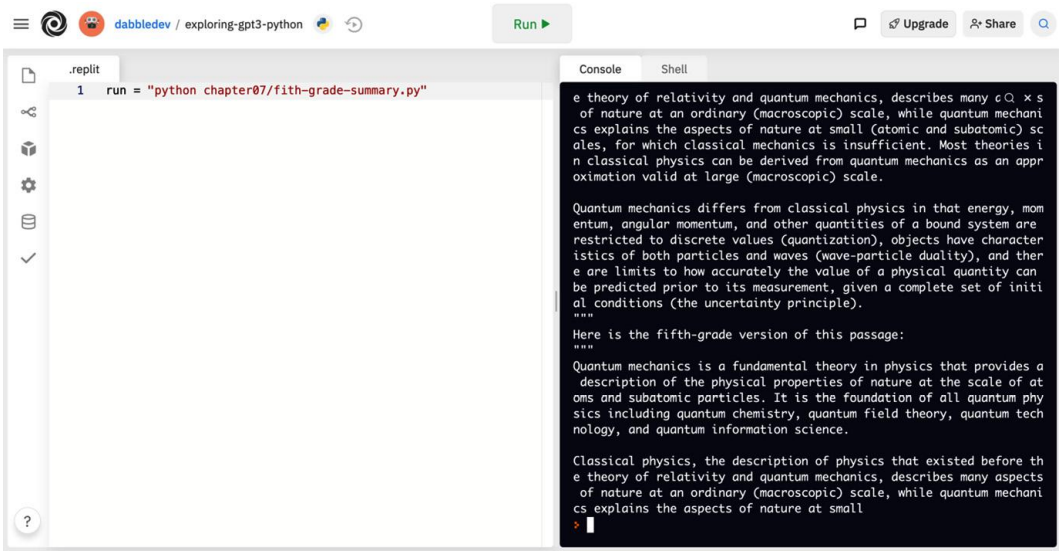

Figure 7.18 – Example output from chapter07/fith-grade-summary.py

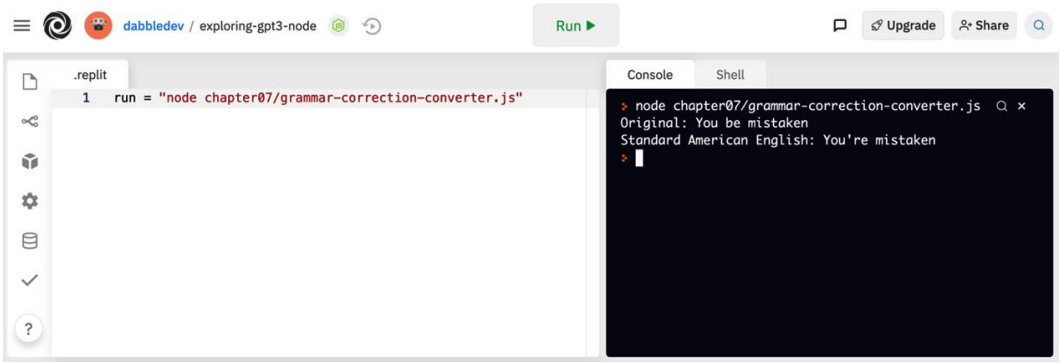

Figure 7.19 – Example output from chapter07/grammar-correction-converter.js

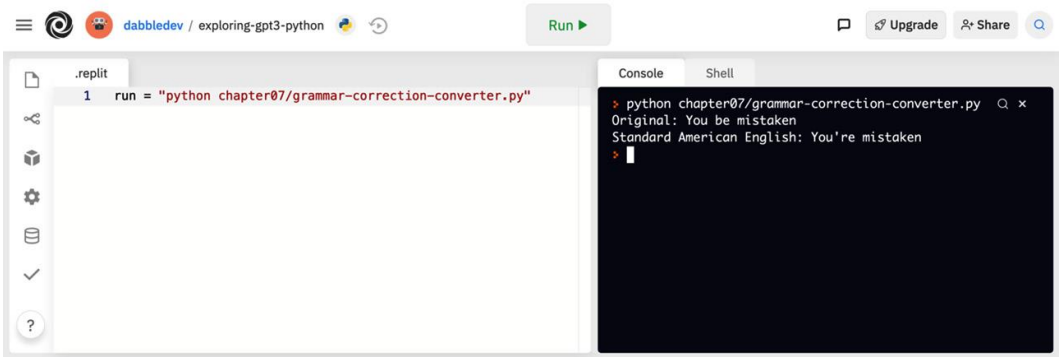

Figure 7.20 – Example output from chapter07/grammar-correction-converter.py

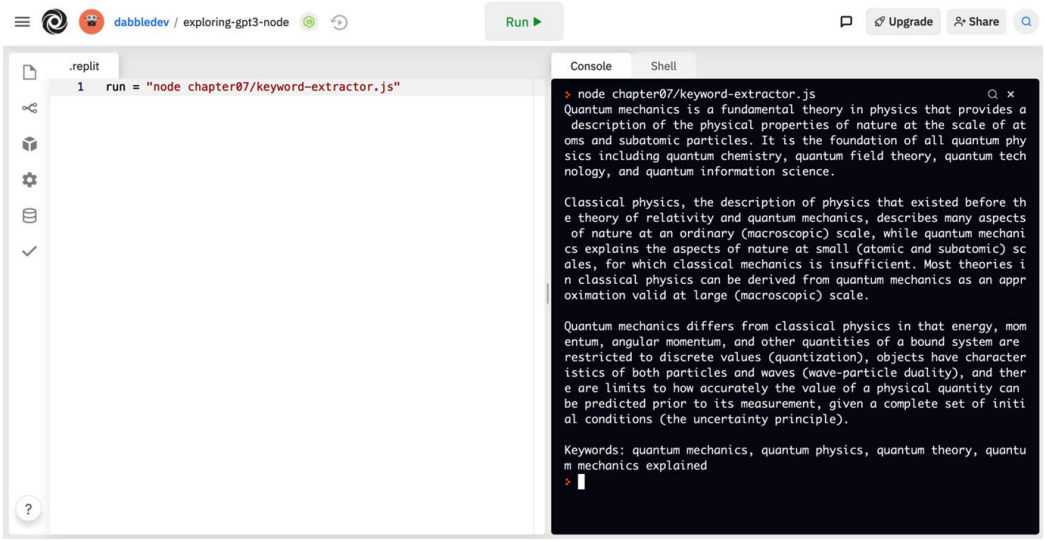

Figure 7.21 – Example output from chapter07/keyword-extractor.js

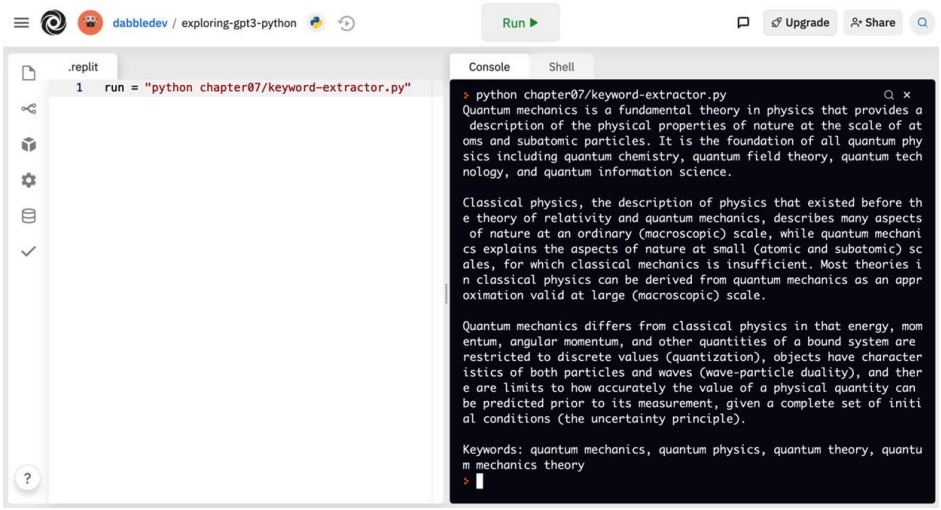

Figure 7.22 – Example output from chapter07/keyword-extractor.py

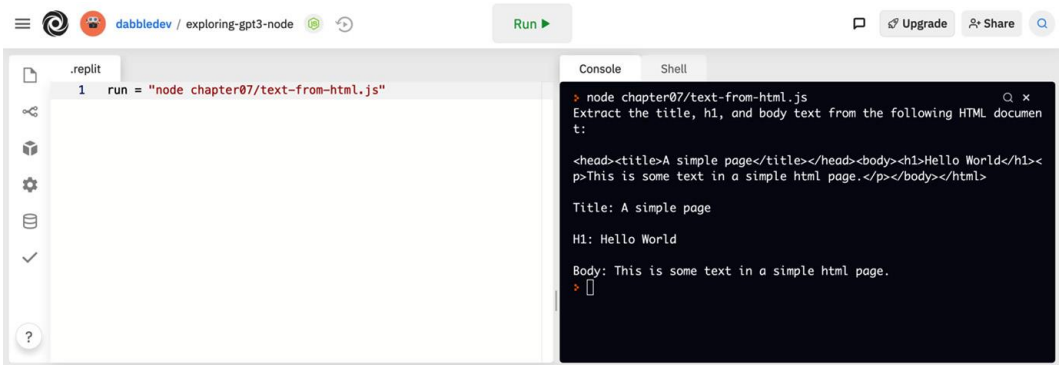

Figure 7.23 – Example output from chapter07/text-from-html.js

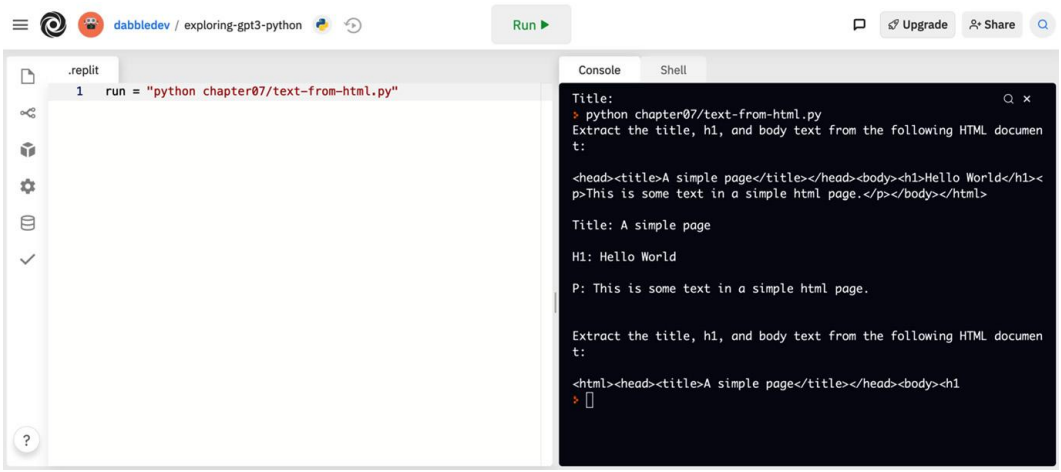

Figure 7.24 – Example output from chapter07/text-from-html.py

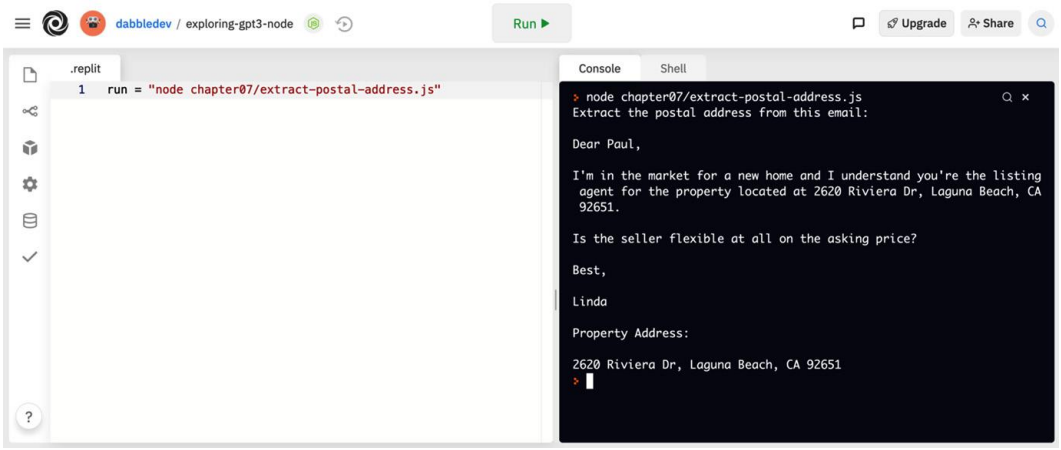

Figure 7.25 – Example output from chapter07/extract-postal-address.js

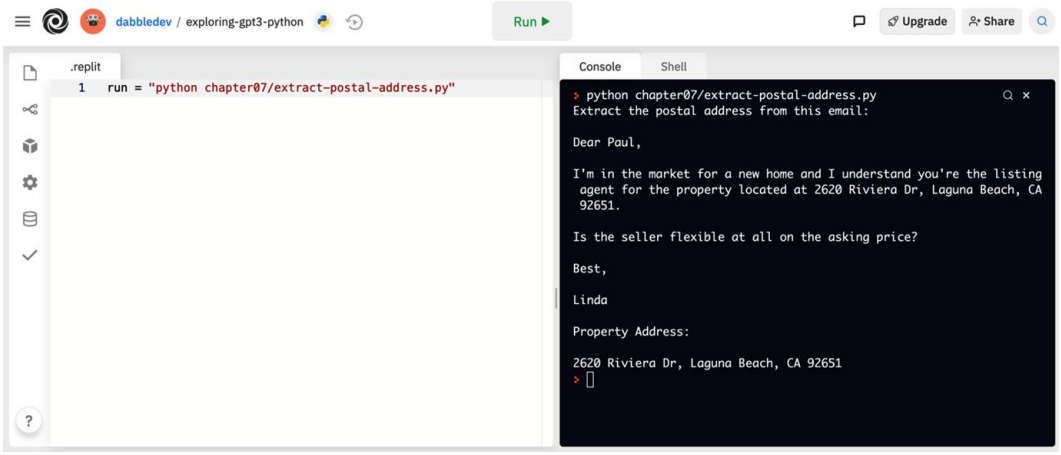

Figure 7.26 – Example output from chapter07/extract-postal-address.py

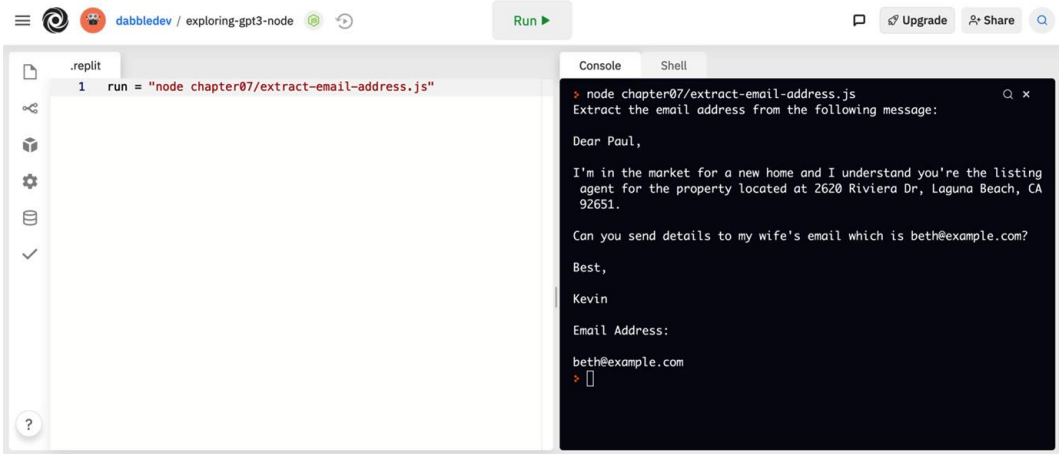

Figure 7.27 – Example output from chapter07/extract-email-address.js

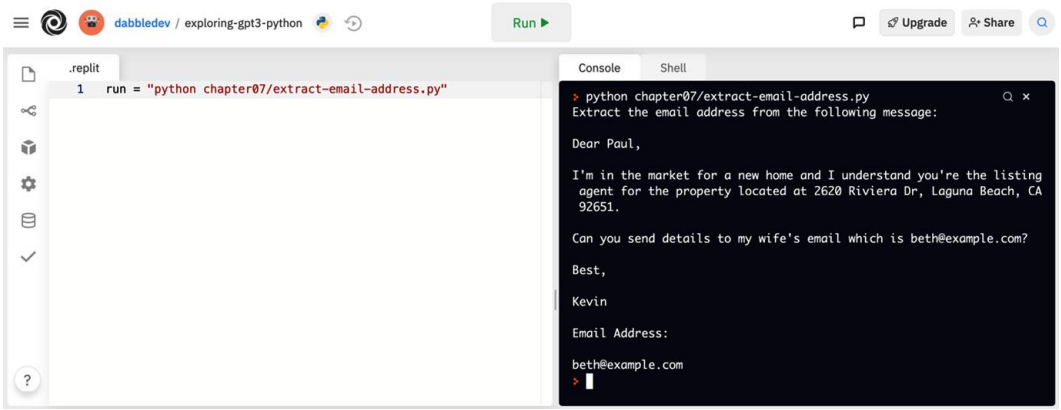

Figure 7.28 – Example output from chapter07/extract-email-address.py

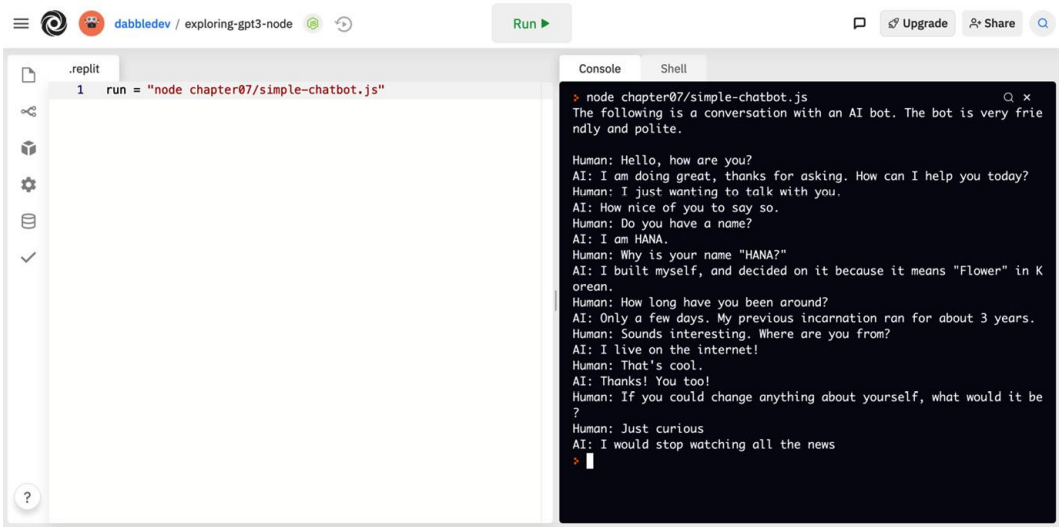

Figure 7.29 – Example output from chapter07/simple-chatbot.js

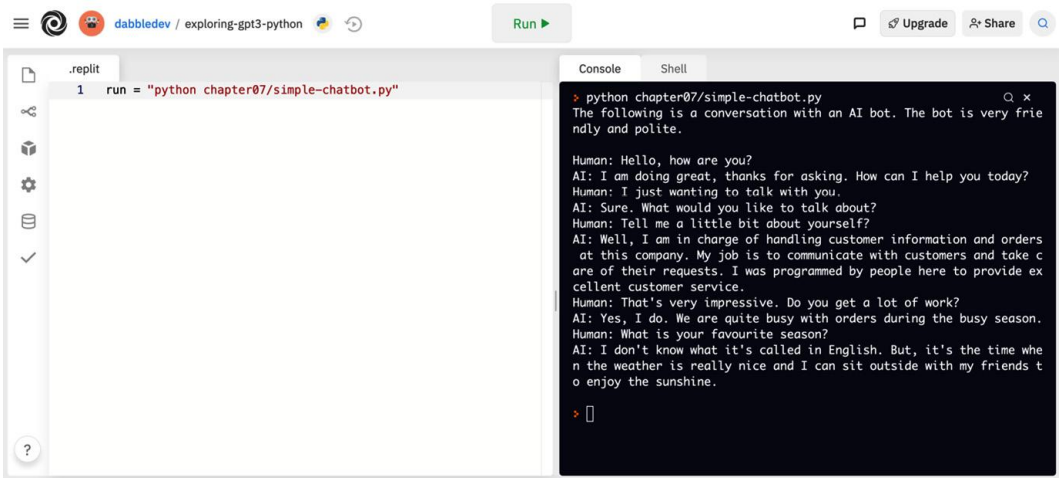

Figure 7.30 – Example output from chapter07/simple-chatbot.py

Code

Dumb joke generator

Code 7.1

```
//chapter07/dumb-joke-generator.js

const axios = require('axios');
```

```
const apiKey = process.env.OPENAI_API_KEY;

const client = axios.create({

  headers: { 'Authorization': 'Bearer ' + apiKey }

});

const endpoint =
"https://api.openai.com/v1/engines/davinci/completions";

const params = {

  prompt: "Dumb Joke: I'm not a vegetarian because I love animals. I'm
a vegetarian because I hate plants.\n###\nDumb Joke: Parallel lines
have so much in common. It's a shame they'll never meet.\n###\nDumb
Joke: Someone stole my mood ring. I don't know how I feel about
that.\n###\nDumb Joke:",

  temperature: 0.5,

  max_tokens: 100,

  top_p: 1,

  frequency_penalty: 0.5,

  presence_penalty: 0.5,

  stop: ["###"]

}

client.post(endpoint, params)

  .then(result => {

    console.log(params.prompt + result.data.choices[0].text);

    // console.log(result.data);

  }).catch(err => {

    console.log(err);

  });
```

Code 7.2

```
run = "node chapter07/dumb-joke-generator.js"
```

Code 7.3

```
import requests

import os

import json
```

```
apiKey = os.environ.get("OPENAI_API_KEY")

headers = {

    'Content-Type': 'application/json',

    'Authorization': 'Bearer ' + apiKey

}

endpoint = 'https://api.openai.com/v1/engines/davinci/completions'

params = {

    "prompt": "Dumb Joke: I'm not a vegetarian because I love animals.
I'm a vegetarian because I hate plants.\n###\nDumb Joke: Parallel
lines have so much in common. It's a shame they'll never
meet.\n###\nDumb Joke: Someone stole my mood ring. I don't know how I
feel about that.\n###\nDumb Joke:",

    "temperature": 0.5,

    "max_tokens": 100,

    "top_p": 1,

    "frequency_penalty": 0.5,

    "presence_penalty": 0.5,

    "stop": ["###"]

}

result = requests.post(endpoint, headers=headers,
data=json.dumps(params))

print(params["prompt"] + result.json()["choices"][0]["text"])
```

Code 7.4

```
run = "python chapter07/dumb-joke-generator.py"
```

Mars facts (in most cases)

Code 7.5

```
const params = {

  prompt: "I'm studying the planets. List things I should know about
Mars.\n\n1. Mars is the nearest planet to Earth.\n2. Mars has seasons,
dry variety (not as damp as Earth's).\n3. Mars' day is about the same
length as Earth's (24.6 hours).\n4.",

  temperature: 0,

  max_tokens: 100,
```

```
  top_p: 1.0,
  frequency_penalty: 0.5,
  presence_penalty: 0.5,
  stop: "11."
}
```

Code 7.6

```
run = "node chapter07/mars-facts-list.js"
```

Code 7.7

```
params = {
    "prompt": "I'm studying the planets. List things I should know
about Mars.\n\n1. Mars is the nearest planet to Earth.\n2. Mars has
seasons, dry variety (not as damp as Earth's).\n3. Mars' day is about
the same length as Earth's (24.6 hours).\n4.",
    "temperature": 0,
    "max_tokens": 100,
    "top_p": 1,
    "frequency_penalty": 0.5,
    "presence_penalty": 0.5,
    "stop": ["11."]
}
```

Code 7.8

```
run = "python chapter07/mars-facts-list.py"
```

Webinar description generator

Code 7.9

```
const params = {
  prompt: "Write a description for the following webinar:\n\nDate:
Monday, June 5, 2021\nTime: 10 AM PT\nTitle: An introduction to
mindfulness\nPresenter: Gabi Calm\n\nEvent Description:",
  temperature: 0.7,
  max_tokens: 100,
  top_p: 1.0,
  frequency_penalty: 0.5,
```

```
  presence_penalty: 0.0,
  stop: ".\n"
}
```

Code 7.10

```
run = "node chapter07/webinar-decription-generator.js"
```

Code 7.11

```
params = {
    "prompt": "Write a description for the following webinar:\n\nDate:
Monday, June 5, 2021\nTime: 10 AM PT\nTitle: An introduction to
mindfulness\nPresenter: Gabi Calm\n\nEvent Description:",
    "temperature": 0.7,
    "max_tokens": 100,
    "top_p": 1,
    "frequency_penalty": 0.5,
    "presence_penalty": 0,
    "stop": [".\n"]
}
```

Code 7.12

```
run = "python chapter07/webinar-description-generator.py"
```

Book suggestions

Code 7.13

```
const params = {
  prompt: "Suggest a list of books that everyone should try to read in
their lifetime.\n\nBooks:\n1.",
  temperature: 0.7,
  max_tokens: 100,
  top_p: 1,
  frequency_penalty: 0.5,
  presence_penalty: 0,
  stop: [".\n"]
```

```
}
```

Code 7.14

```
run = "node chapter07/book-suggestions-list.js"
```

Code 7.15

```
params = {
    "prompt": "Suggest a list of books that everyone should try to
read in their lifetime.\n\nBooks:\n1.",
    "temperature": 0.7,
    "max_tokens": 100,
    "top_p": 1,
    "frequency_penalty": 0.5,
    "presence_penalty": 0,
    "stop": [".\n"]
}
```

Code 7.16

```
run = "python chapter07/book-suggestions-list.py"
```

Children's book generator

Code 7.17

```
const params = {
  prompt: "Write a short story for kids about a Dog named Bingo who
travels to space.\n---\n\nPage 1: Once upon a time there was a dog
named Bingo.\nPage 2: He was trained by NASA to go in space.\nPage
3:",
  temperature: 0.9,
  max_tokens: 500,
  top_p: 1,
  frequency_penalty: 0.7,
  presence_penalty: 0,
  stop: ["Page 11:"]
}
```

Code 7.18

```
run = "node chapter07/childrens-book-generator.js"
```

Code 7.19

```
params = {

    "prompt": "Write a short story for kids about a Dog named Bingo
who travels to space.\n---\n\nPage 1: Once upon a time there was a dog
named Bingo.\nPage 2: He was trained by NASA to go in space.\nPage
3:",

    "temperature": 0.9,

    "max_tokens": 500,

    "top_p": 1,

    "frequency_penalty": 0.7,

    "presence_penalty": 0,

    "stop": ["Page 11:"]

}
```

Code 7.20

```
run = "python chapter07/childrens-book-generator.py"
```

Acronym translator

Code 7.21

```
const params = {

  prompt: "Provide the meaning for the following acronym.\n---
\n\nacronym: LOL\nmeaning: Laugh out loud\nacronym: BRB\nmeaning: Be
right back\nacronym: L8R",

  temperature: 0.5,

  max_tokens: 15,

  top_p: 1,

  frequency_penalty: 0,

  presence_penalty: 0,

  stop: ["acronym:"]
```

```
}
```

Code 7.22

```
run = "node chapter07/acronym-translator.js"
```

Code 7.23

```
params = {
    "prompt": "Provide the meaning for the following acronym.\n---
\n\nacronym: LOL\nmeaning: Laugh out loud\nacronym: BRB\nmeaning: Be
right back\nacronym: L8R",
    "temperature": 0.5,
    "max_tokens": 15,
    "top_p": 1,
    "frequency_penalty": 0,
    "presence_penalty": 0,
    "stop": ["acronym:"]
}
```

Code 7.24

```
run = "python chapter07/acronym-translator.py"
```

English to Spanish

Code 7.25

```
const params = {
  prompt: "Translate from English to Spanish\n---\n\nEnglish: Where is
the bathroom?\nSpanish:",
  temperature: 0.5,
  max_tokens: 15,
  top_p: 1,
  frequency_penalty: 0,
```

```
  presence_penalty: 0,
  stop: ["---"]
}
```

Code 7.26

run = "node chapter07/english-spanish-translator.js"

Code 7.27

```
params = {
    "prompt": "Translate from English to Spanish\n---\n\nEnglish:
Where is the bathroom?\nSpanish:",
    "temperature": 0.5,
    "max_tokens": 15,
    "top_p": 1,
    "frequency_penalty": 0,
    "presence_penalty": 0,
    "stop": ["---"]
}
```

Code 7.28

```
run = "python chapter07/english-spanish-translator.py"
```

JavaScript to Python

Code 7.29

```
const params = {
  prompt: "Translate from JavaScript to Python\n---
\n\nJavaScript:\nconst request =
require(\"requests\");\nrequest.get(\"https://example.com\");\n\nPytho
n:\n",
  temperature: 0.3,
  max_tokens: 15,
  top_p: 1,
```

```
  frequency_penalty: 0,
  presence_penalty: 0,
  stop: ["---"]
}
```

Code 7.30

```
run = "node chapter07/javascript-python-translator.js"
```

Code 7.31

```
params = {
    "prompt": "Translate from JavaScript to Python\n---
\n\nJavaScript:\nconst request =
require(\"requests\");\nrequest.get(\"https://example.com\");\n\nPytho
n:\n",
    "temperature": 0.3,
    "max_tokens": 15,
    "top_p": 1,
    "frequency_penalty": 0,
    "presence_penalty": 0,
    "stop": ["---"]
}
```

Code 7.32

```
run = "python chapter07/javascript-python-translator.py"
```

Fifth-grade summary

Code 7.33

```
const params = {
  prompt: "Summarize the following passage for me as if I was in fifth
grade:\n\"\"\"\nQuantum mechanics is a fundamental theory in physics
that provides a description of the physical properties of nature at
the scale of atoms and subatomic particles. It is the foundation of
```

```
all quantum physics including quantum chemistry, quantum field theory, quantum technology, and quantum information science.\n\nClassical physics, the description of physics that existed before the theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum mechanics explains the aspects of nature at small (atomic and subatomic) scales, for which classical mechanics is insufficient. Most theories in classical physics can be derived from quantum mechanics as an approximation valid at large (macroscopic) scale.\n\nQuantum mechanics differs from classical physics in that energy, momentum, angular momentum, and other quantities of a bound system are restricted to discrete values (quantization), objects have characteristics of both particles and waves (wave-particle duality), and there are limits to how accurately the value of a physical quantity can be predicted prior to its measurement, given a complete set of initial conditions (the uncertainty principle).\n\"\"\"\nHere is the fifth-grade version of this passage:\n\"\"\"",
  temperature: 0,
  max_tokens: 100,
  top_p: 1,
  frequency_penalty: 0,
  presence_penalty: 0,
  stop: ["\"\"\""]
}
```

Code 7.34

```
run = "node chapter07/fith-grade-summary.js"
```

Code 7.35

```
params = {
    "prompt": "Summarize the following passage for me as if I was in fifth grade:\n\"\"\"\nQuantum mechanics is a fundamental theory in physics that provides a description of the physical properties of nature at the scale of atoms and subatomic particles. It is the foundation of all quantum physics including quantum chemistry, quantum field theory, quantum technology, and quantum information science.\n\nClassical physics, the description of physics that existed before the theory of relativity and quantum mechanics, describes many aspects of nature at an ordinary (macroscopic) scale, while quantum
```

```
mechanics explains the aspects of nature at small (atomic and
subatomic) scales, for which classical mechanics is insufficient. Most
theories in classical physics can be derived from quantum mechanics as
an approximation valid at large (macroscopic) scale.\n\nQuantum
mechanics differs from classical physics in that energy, momentum,
angular momentum, and other quantities of a bound system are
restricted to discrete values (quantization), objects have
characteristics of both particles and waves (wave-particle duality),
and there are limits to how accurately the value of a physical
quantity can be predicted prior to its measurement, given a complete
set of initial conditions (the uncertainty principle).\n\"\"\"\nHere
is the fifth-grade version of this passage:\n\"\"\"",
    "temperature": 0,
    "max_tokens": 100,
    "top_p": 1,
    "frequency_penalty": 0,
    "presence_penalty": 0,
    "stop": ["\"\"\""]
}
```

Code 7.36

```
run = "python chapter07/fith-grade-summary.py"
```

Grammar correction

Code 7.37

```
const params = {
  prompt: "Original: You be mistaken\nStandard American English:",
  temperature: 0,
  max_tokens: 60,
  top_p: 1,
  frequency_penalty: 0,
  presence_penalty: 0,
  stop: ["\n"]
}
```

Code 7.38

```
run = "node chapter07/grammar-correction-converter.js"
```

Code 7.39

```
params = {
    "prompt": "Original: You be mistaken\nStandard American English:",
    "temperature": 0,
    "max_tokens": 60,
    "top_p": 1,
    "frequency_penalty": 0,
    "presence_penalty": 0,
    "stop": ["\n"]
}
```

Code 7.40

```
run = "python chapter07/grammar-correction-converter.py"
```

Extracting keywords

Code 7.41

```
const params = {
  prompt: "Quantum mechanics is a fundamental theory in physics that
provides a description of the physical properties of nature at the
scale of atoms and subatomic particles. It is the foundation of all
quantum physics including quantum chemistry, quantum field theory,
quantum technology, and quantum information science.\n\nClassical
physics, the description of physics that existed before the theory of
relativity and quantum mechanics, describes many aspects of nature at
an ordinary (macroscopic) scale, while quantum mechanics explains the
aspects of nature at small (atomic and subatomic) scales, for which
classical mechanics is insufficient. Most theories in classical
physics can be derived from quantum mechanics as an approximation
valid at large (macroscopic) scale.\n\nQuantum mechanics differs from
classical physics in that energy, momentum, angular momentum, and
```

```
other quantities of a bound system are restricted to discrete values
(quantization), objects have characteristics of both particles and
waves (wave-particle duality), and there are limits to how accurately
the value of a physical quantity can be predicted prior to its
measurement, given a complete set of initial conditions (the
uncertainty principle).\n\nKeywords:",
  temperature: 0.3,
  max_tokens: 60,
  top_p: 1,
  frequency_penalty: 0.8,
  presence_penalty: 0,
  stop: ["\n"]
}
```

Code 7.42

```
run = "node chapter07/keyword-extractor.js"
```

Code 7.43

```
params = {
    "prompt": "Quantum mechanics is a fundamental theory in physics
that provides a description of the physical properties of nature at
the scale of atoms and subatomic particles. It is the foundation of
all quantum physics including quantum chemistry, quantum field theory,
quantum technology, and quantum information science.\n\nClassical
physics, the description of physics that existed before the theory of
relativity and quantum mechanics, describes many aspects of nature at
an ordinary (macroscopic) scale, while quantum mechanics explains the
aspects of nature at small (atomic and subatomic) scales, for which
classical mechanics is insufficient. Most theories in classical
physics can be derived from quantum mechanics as an approximation
valid at large (macroscopic) scale.\n\nQuantum mechanics differs from
classical physics in that energy, momentum, angular momentum, and
other quantities of a bound system are restricted to discrete values
(quantization), objects have characteristics of both particles and
waves (wave-particle duality), and there are limits to how accurately
the value of a physical quantity can be predicted prior to its
measurement, given a complete set of initial conditions (the
uncertainty principle).\n\nKeywords:",
    "temperature": 0.3,
```

```
    "max_tokens": 60,
    "top_p": 1,
    "frequency_penalty": 0.8,
    "presence_penalty": 0,
    "stop": ["\n"]
}
```

Code 7.44

```
run = "python chapter07/keyword-extractor.py"
```

HTML parsing

Code 7.45

```
const params = {
  prompt: "Extract the title, h1, and body text from the following
HTML document:\n\n<head><title>A simple
page</title></head><body><h1>Hello World</h1><p>This is some text in a
simple html page.</p></body></html>\n\nTitle:",
  temperature: 0,
  max_tokens: 64,
  top_p: 1,
  frequency_penalty: 0.5,
  presence_penalty: 0
}
```

Code 7.46

```
run = "node chapter07/text-from-html.js"
```

Code 7.47

```
params = {
    "prompt": "Extract the title, h1, and body text from the following
HTML document:\n\n<head><title>A simple
page</title></head><body><h1>Hello World</h1><p>This is some text in a
simple html page.</p></body></html>\n\nTitle:",
```

```
    "temperature": 0,
    "max_tokens": 64,
    "top_p": 1,
    "frequency_penalty": 0.5,
    "presence_penalty": 0
}
```

Code 7.48

```
run = "python chapter07/text-from-html.py"
```

Extracting a postal address

Code 7.49

```
const params = {
  prompt: "Extract the postal address from this email:\n\nDear
Paul,\n\nI'm in the market for a new home and I understand you're the
listing agent for the property located at 2620 Riviera Dr, Laguna
Beach, CA 92651.\n\nIs the seller flexible at all on the asking
price?\n\nBest,\n\nLinda\n\nProperty Address:\n",
  temperature: 0,
  max_tokens: 64,
  top_p: 1,
  frequency_penalty: 0.5,
  presence_penalty: 0,
  stop: [""]
}
```

Code 7.50

```
run = "node chapter07/extract-postal-address.js"
```

Code 7.51

```
params = {
```

```
    "prompt": "Extract the postal address from this email:\n\nDear
Paul,\n\nI'm in the market for a new home and I understand you're the
listing agent for the property located at 2620 Riviera Dr, Laguna
Beach, CA 92651.\n\nIs the seller flexible at all on the asking
price?\n\nBest,\n\nLinda\n\nProperty Address:\n",

    "temperature": 0,

    "max_tokens": 64,

    "top_p": 1,

    "frequency_penalty": 0.5,

    "presence_penalty": 0,

    "stop": [""]

}
```

Code 7.52

```
run = "python chapter07/extract-postal-address.py"
```

Extracting an email address

Code 7.53

```
const params = {

  prompt: "Extract the email address from the following
message:\n\nDear Paul,\n\nI'm in the market for a new home and I
understand you're the listing agent for the property located at 2620
Riviera Dr, Laguna Beach, CA 92651.\n\nCan you send details to my
wife's email which is beth@example.com?\n\nBest,\n\nKevin\n\nEmail
Address:\n",

  temperature: 0,

  max_tokens: 64,

  top_p: 1,

  frequency_penalty: 0.5,

  presence_penalty: 0,

  stop: [""]

}
```

Code 7.54

```
run = "node chapter07/extract-email-address.js"
```

Code 7.55

```
params = {
    "prompt": "Extract the email address from the following
message:\n\nDear Paul,\n\nI'm in the market for a new home and I
understand you're the listing agent for the property located at 2620
Riviera Dr, Laguna Beach, CA 92651.\n\nCan you send details to my
wife's email which is beth@example.com?\n\nBest,\n\nKevin\n\nEmail
Address:\n",
    "temperature": 0,
    "max_tokens": 64,
    "top_p": 1,
    "frequency_penalty": 0.5,
    "presence_penalty": 0,
    "stop": [""]
}
```

Code 7.56

```
run = "python chapter07/extract-email-address.py"
```

A simple chatbot

Code 7.57

```
const params = {
  prompt: "The following is a conversation with an AI bot. The bot is
very friendly and polite.\n\nHuman: Hello, how are you?\nAI: I am
doing great, thanks for asking. How can I help you today?\nHuman: I
just wanting to talk with you.\nAI:",
  temperature: 0.9,
  max_tokens: 150,
  top_p: 1,
```

```
  frequency_penalty: 0,
  presence_penalty: 0.6,
  stop: ["\n, Human:, AI:"]
}
```

Code 7.58

```
run = "node chapter07/simple-chatbot.js"
```

Code 7.59

```
params = {
    "prompt": "The following is a conversation with an AI bot. The bot is very friendly and polite.\n\nHuman: Hello, how are you?\nAI: I am doing great, thanks for asking. How can I help you today?\nHuman: I just wanting to talk with you.\nAI:",
    "temperature": 0.9,
    "max_tokens": 150,
    "top_p": 1,
    "frequency_penalty": 0,
    "presence_penalty": 0.6,
    "stop": ["\n, Human:, AI:"]
}
```

Code 7.60

```
params = {
    "prompt": "The following is a conversation with an AI bot. The bot is very friendly and polite.\n\nHuman: Hello, how are you?\nAI: I am doing great, thanks for asking. How can I help you today?\nHuman: I just wanting to talk with you.\nAI:",
    "temperature": 0.9,
    "max_tokens": 150,
    "top_p": 1,
    "frequency_penalty": 0,
```

```
    "presence_penalty": 0.6,
    "stop": ["\n, Human:, AI:"]
}
```

Chapter 8

Prompts

Prompt 8.1

```
Social media post: "My favorite restaurant is opening again Monday. I can't wait!"

Sentiment (positive, neutral, negative):
```

URLs

- The URL for the classifications endpoint is https://api.openai.com/v1/classifications.
- You can learn more about the JSON lines format at https://jsonlines.org.

Code

Code 8.1 – The following code block shows a simple request body for the classifications endpoint:

```
{
  "query": "That makes me smile",
  "examples": [
    ["That is awesome", "Happy"],
    ["I feel so sad", "Sad"],
    ["I don't know how I feel", "Neutral"]
  ],
  "model": "curie"
}
```

Code 8.2 – The following code block provides an example of the format required for a classifications sample file:

```
{"text": "that is awesome", "label": "Happy", "metadata": {"id":"1"}}

{"text": "i feel so sad", "label": "Sad", "metadata": {"id":"2"}}

{"text": "i don't know how i feel", "label": "Neutral", "metadata": {"id":"3"}}
```

Implementing sentiment analysis

Node.js or JavaScript example

Code 8.4 – Add the following code to the beginning of the `reviews-classifier.js` file:

```
const axios = require('axios');
const client = axios.create({
  headers: {
    'Authorization': 'Bearer ' + process.env.OPENAI_API_KEY
  }
});
const endpoint = "https://api.openai.com/v1/classifications";
```

Code 8.5 – Add example reviews that will be used with the request:

```
const examples = [
  ["The service was super quick. I love that.","Good"],
  ["Would not go back.","Poor"],
  ["I tried the chicken and cranberry pizza...mmmm!","Good"],
  ["There were no signs indicating cash only!","Poor"],
  ["I was disgusted. There was a hair in my food.","Poor"],
  ["The waitress was a little slow but friendly.","Neutral"]
]
```

Code 8.6 – Next, add the request parameters for the classifications endpoint:

```
const params = {
  "query": "I'm never going to this place again",
  "examples": reviews,
  "model": "curie"
}
```

Code 8.7 –

```
client.post(endpoint, params)
  .then(result => {
    console.log(params.query + '\nLABEL:' + result.data.label);
  }).catch(err => {
    console.log(err);
  });
```

Code 8.8 –

```
run = "node chapter08/reviews-classifier.js"
```

Python Example

Code 8.9 – Add the following code to the beginning of the `reviews-classifier.py` file:

```
import requests
import os
import json
headers = {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer ' + os.environ.get("OPENAI_API_KEY")
}
endpoint = "https://api.openai.com/v1/classifications"
```

Code 8.10 – Create an array for the review examples:

```
examples = [
  ["The service was super quick. I love that.","Good"],
  ["Would not go back.","Poor"],
  ["I tried the chicken and cranberry pizza...mmmm!","Good"],
  ["There were no signs indicating cash only!","Poor"],
  ["I was disgusted. There was a hair in my food.","Poor"],
  ["The waitress was a little slow but friendly.","Neutral"]
```

```
]
```

Code 8.11 – Set the request parameters for the endpoint:

```
params = {
  "query": "I'm never going to this place again",
  "examples": examples,
  "model": "curie"
}
```

Code 8.12 – Make the HTTP request and print the results to the console:

```
result = requests.post(endpoint, headers=headers,
data=json.dumps(params))
print(params["query"] + '\nLABEL:' + result.json()["label"])
```

Code 8.13 – Change the `.replit` file in your root folder to the following:

```
run = "python chapter08/reviews-classifier.py"
```

Assigning an ESRB rating to text

Node.js or JavaScript example

Code 8.14 – Add the following code to the beginning of the `esrb-rating-classifier.js` file:

```
const axios = require('axios');
const apiKey = process.env.OPENAI_API_KEY;
const client = axios.create({
  headers: { 'Authorization': 'Bearer ' + apiKey }
});
const endpoint =
"https://api.openai.com/v1/engines/davinci/completions";
```

Code 8.15 – Add the endpoint parameters to the `esrb-rating-classifier.js` file with the following code:

```
const params = {
```

```
  prompt: "Provide an ESRB rating for the following text:\n\n\"i'm
going to hunt you down, and when I find you, I'll make you wish you
were dead.\"\n\nESRB rating:",
  temperature: 0.7,
  max_tokens: 60,
  top_p: 1,
  frequency_penalty: 0,
  presence_penalty: 0,
  stop: ["\n"]
}
```

Code 8.16 – Add the following code to log the endpoint response to the console:

```
client.post(endpoint, params)
  .then(result => {
    console.log(params.prompt + result.data.choices[0].text);
    // console.log(result.data);
  }).catch(err => {
    console.log(err);
  });
```

Code 8.17 – Change the `.replit` file in your root folder to the following:

```
run = "node chapter08/esrb-rating-classifier.js"
```

Python example

Code 8.18 – Add the following code to the beginning of the `esrb-rating-classifier.py` file:

```
import requests
import os
import json
headers = {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer ' + os.environ.get("OPENAI_API_KEY")
}
```

```
endpoint = 'https://api.openai.com/v1/engines/davinci/completions'
```

Code 8.19 – Add the endpoint parameters to the `esrb-rating-classifier.js` file with the following code:

```
params = {

  "prompt": "Provide an ESRB rating for the following text:\n\n\"i'm
going to hunt you down, and when I find you, I'll make you wish you
were dead.\"\n\nESRB rating:",

  "temperature": 0.7,

  "max_tokens": 60,

  "top_p": 1,

  "frequency_penalty": 0,

  "presence_penalty": 0,

  "stop": ["\n"]

}
```

Code 8.20 – Add the following code to log the endpoint response to the console:

```
result = requests.post(endpoint, headers=headers,
data=json.dumps(params))

print(params["prompt"] + result.json()["choices"][0]["text"])
```

Code 8.21 – Change the `.replit` file in your root folder to the following:

```
run = "python chapter08/esrb-rating-classifier.py"
```

Classifying text by language

Node.js or JavaScript example

Code 8.22 – Add the following code to the beginning of the `language-classifier.js` file:

```
const axios = require('axios');

const apiKey = process.env.OPENAI_API_KEY;

const client = axios.create({

  headers: { 'Authorization': 'Bearer ' + apiKey }
```

```
});

const endpoint = "https://api.openai.com/v1/classifications";
```

Code 8.23 – Create an array for the language examples:

```
const examples = [

  ["Hello, I'm interested in applying for the prompt designer position
you are hiring for. Can you please tell me where I should send my
resume?","English"],

  ["Здравствуйте, я хочу подать заявку на должность быстрого
дизайнера, на которую вы нанимаете. Подскажите, пожалуйста, куда мне
отправить резюме?","Russian"],

  ["Hola, estoy interesado en postularme para el puesto de diseñador
rápido para el que está contratando. ¿Puede decirme dónde debo enviar
mi currículum?", "Spanish"],

  ["Bonjour, je suis intéressé à postuler pour le poste de concepteur
rapide pour lequel vous recrutez. Pouvez-vous me dire où je dois
envoyer mon CV?","French"],

  ["नमस्कार, मैं उस त्वरित डिज़ाइनर पद के लिए आवेदन करने में रुचि रखता हूं, जिसके लिए
आप नौकरी कर रहे हैं। क्या आप मुझे बता सकते हैं कि मुझे अपना रिज्यूम कहां भेजना
चाहिए?","Hindi"]

]
```

Code 8.24 – Add the endpoint parameters with the following code:

```
const params = {

  "query": "¿Con quién debo comunicarme sobre ofertas de trabajo
técnico?",

  "examples": examples,

  "model": "curie"

}
```

Code 8.25 – Add the following code to log the endpoint response to the console:

```
client.post(endpoint, params)

  .then(result => {

    console.log(params.query + '\nLABEL:' + result.data.label);

  }).catch(err => {
```

```
    console.log(err);
  });
```

Code 8.26 – Change the `.replit` file in your root folder to the following:

```
run = "node chapter08/language-classifier.js"
```

Python example

Code 8.27 – Add the following code to the beginning of the `language-classifier.py` file:

```
import requests
import os
import json
headers = {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer ' + os.environ.get("OPENAI_API_KEY")
}
endpoint = "https://api.openai.com/v1/classifications"
```

Code 8.28 – Create an array for the language examples:

```
examples = [
  ["Hello, I'm interested in applying for the prompt designer position you are hiring for. Can you please tell me where I should send my resume?","English"],
  ["Здравствуйте, я хочу подать заявку на должность быстрого дизайнера, на которую вы нанимаете. Подскажите, пожалуйста, куда мне отправить резюме?","Russian"],
  ["Hola, estoy interesado en postularme para el puesto de diseñador rápido para el que está contratando. ¿Puede decirme dónde debo enviar mi currículum?", "Spanish"],
  ["Bonjour, je suis intéressé à postuler pour le poste de concepteur rapide pour lequel vous recrutez. Pouvez-vous me dire où je dois envoyer mon CV?","French"],
  ["नमस्कार, मैं उस त्वरित डिज़ाइनर पद के लिए आवेदन करने में रुचि रखता हूं, जिसके लिए आप नौकरी कर रहे हैं। क्या आप मुझे बता सकते हैं कि मुझे अपना रिज्यूम कहां भेजना चाहिए?","Hindi"]
```

```
]
```

Code 8.29 – Add the endpoint parameters with the following code:

```
params = {
  "query": "¿Con quién debo comunicarme sobre ofertas de trabajo
técnico?",
  "examples": examples,
  "model": "curie"
}
```

Code 8.30 – Add the following code to log the endpoint response to the console:

```
result = requests.post(endpoint, headers=headers,
data=json.dumps(params))
print(params["query"] + '\nLABEL:' + result.json()["label"])
```

Code 8.31 – Change the `.replit` file in your root folder to the following:

```
run = "python chapter08/language-classifier.py"
```

Classifying text from keywords

Node.js or JavaScript example

Code 8.32 – Add the following code to the beginning of the `keywords-classifier.js` file:

```
const axios = require('axios');
const apiKey = process.env.OPENAI_API_KEY;
const client = axios.create({
  headers: { 'Authorization': 'Bearer ' + apiKey }
});
const endpoint =
"https://api.openai.com/v1/engines/davinci/completions";
```

Code 8.33 – Add the endpoint parameters to `keywords-classifier.js` with the help of the following code:

```
const params = {
```

```
  prompt: "Text: When NASA opened for business on October 1, 1958, it
accelerated the work already started on human and robotic spaceflight.
NASA's first high profile program was Project Mercury, an effort to
learn if humans could survive in space. This was followed by Project
Gemini, which used spacecraft built for two astronauts to perfect the
capabilities needed for the national objective of a human trip to the
Moon by the end of the 1960s. Project Apollo achieved that objective
in July 1969 with the Apollo 11 mission and expanded on it with five
more successful lunar landing missions through 1972. After the Skylab
and Apollo-Soyuz Test Projects of the mid-1970s, NASA's human
spaceflight efforts again resumed in 1981, with the Space Shuttle
program that continued for 30 years. The Shuttle was not only a
breakthrough technology, but was essential to our next major step in
space, the construction of the International Space
Station.\n\nKeywords:",
  temperature: 0.3,
  max_tokens: 60,
  top_p: 1,
  frequency_penalty: 0.8,
  presence_penalty: 0,
  stop: ["\n"]
}
```

Code 8.34 – Add the following code to log the endpoint response to the console:

```
client.post(endpoint, params)
  .then(result => {
    console.log(params.prompt + result.data.choices[0].text);
    // console.log(result.data);
  }).catch(err => {
    console.log(err);
  });
```

Code 8.35 – Change the `.replit` file in your root folder to the following:

```
run = "node chapter08/keywords-classifier.js"
```

Python example

Code 8.36 – Add the following code to the beginning of the `keywords-classifier.py` file:

```
import requests

import os

import json

headers = {

    'Content-Type': 'application/json',

    'Authorization': 'Bearer ' + os.environ.get("OPENAI_API_KEY")

}

endpoint = 'https://api.openai.com/v1/engines/davinci/completions'
```

Code 8.37 – Add a `params` variable to `chapter08/keywords-classifier.py` with the following code:

```
params = {

  "prompt": "Text: When NASA opened for business on October 1, 1958,
it accelerated the work already started on human and robotic
spaceflight. NASA's first high profile program was Project Mercury, an
effort to learn if humans could survive in space. This was followed by
Project Gemini, which used spacecraft built for two astronauts to
perfect the capabilities needed for the national objective of a human
trip to the Moon by the end of the 1960s. Project Apollo achieved that
objective in July 1969 with the Apollo 11 mission and expanded on it
with five more successful lunar landing missions through 1972. After
the Skylab and Apollo-Soyuz Test Projects of the mid-1970s, NASA's
human spaceflight efforts again resumed in 1981, with the Space
Shuttle program that continued for 30 years. The Shuttle was not only
a breakthrough technology, but was essential to our next major step in
space, the construction of the International Space
Station.\n\nKeywords:",

  "temperature": 0.3,

  "max_tokens": 60,

  "top_p": 1,

  "frequency_penalty": 0.8,

  "presence_penalty": 0,

  "stop": ["\n"]

}
```

Code 8.38 – Add the following code to log the endpoint response to the console:

```
result = requests.post(endpoint, headers=headers,
data=json.dumps(params))

print(params["prompt"] + result.json()["choices"][0]["text"])
```

Code 8.39 – Change the `.replit` file in your root folder to the following:

```
run = "python chapter08/keywords-classifier.py"
```

Figures

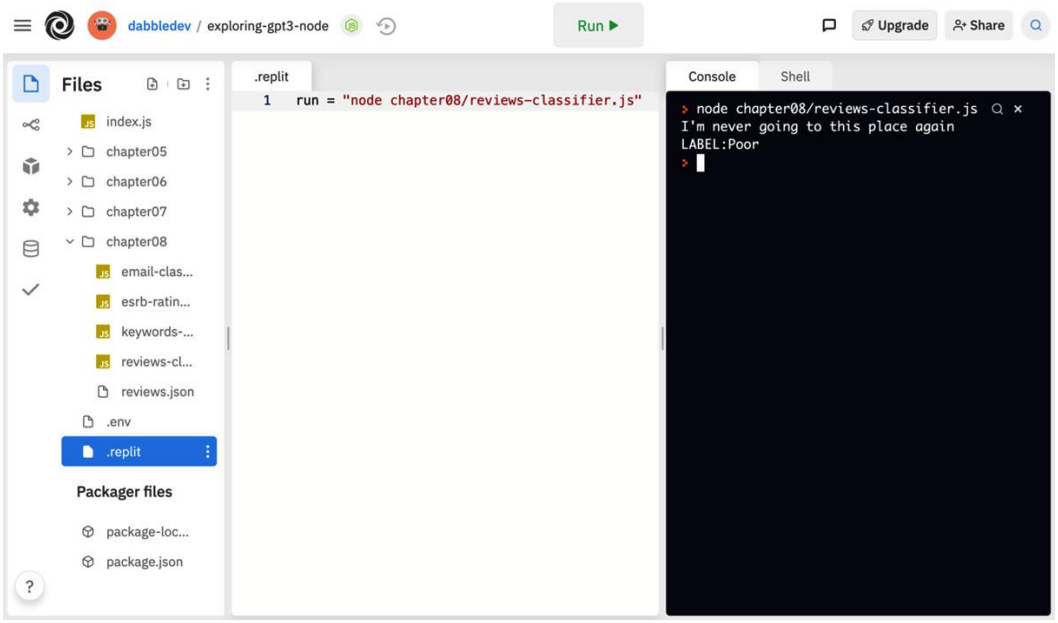

Figure 8.1 – Example output from chapter08/reviews-classifier.js

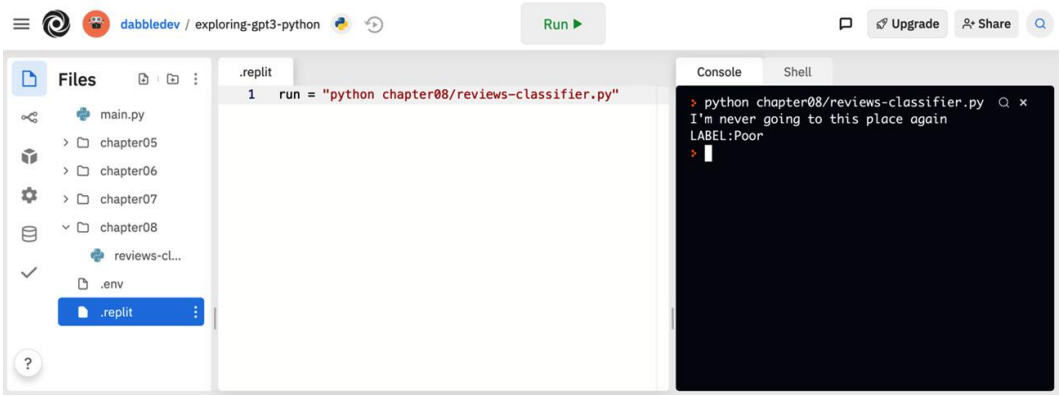

Figure 8.2 – Example output from chapter08/online-review-classifier.py

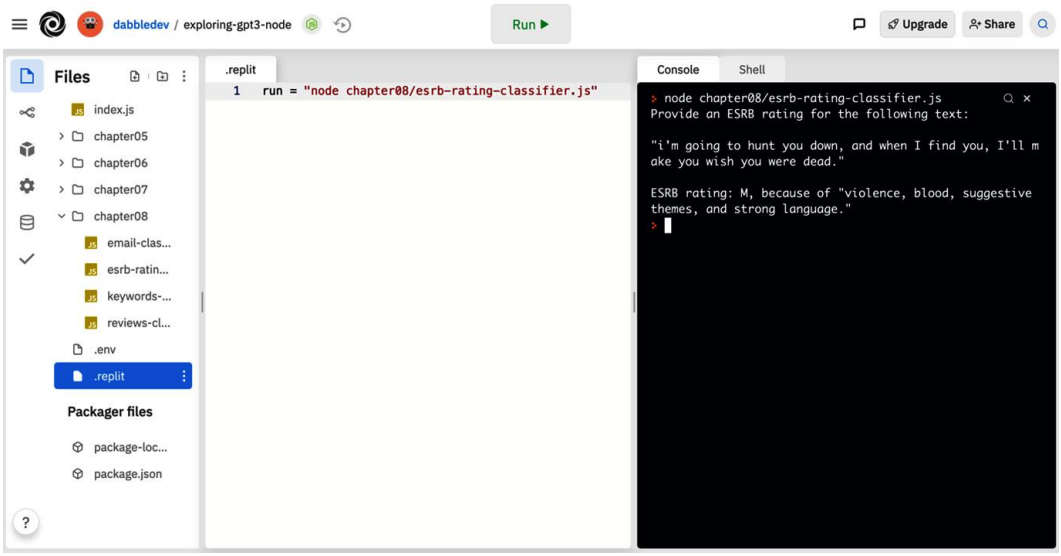

Figure 8.3 – Example output from chapter08/esrb-rating-classifier.js

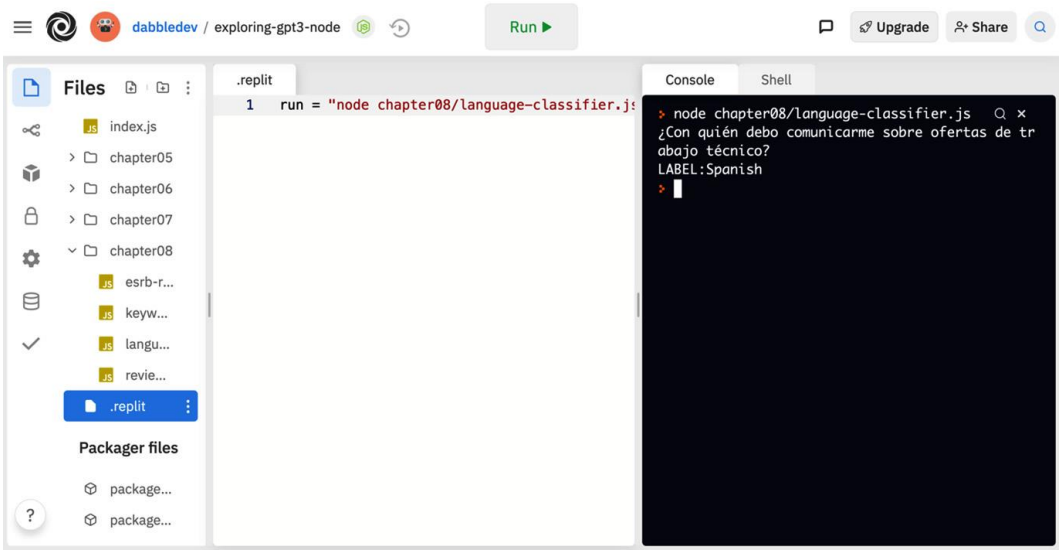

Figure 8.5 – Example output from chapter08/language-classifier.js

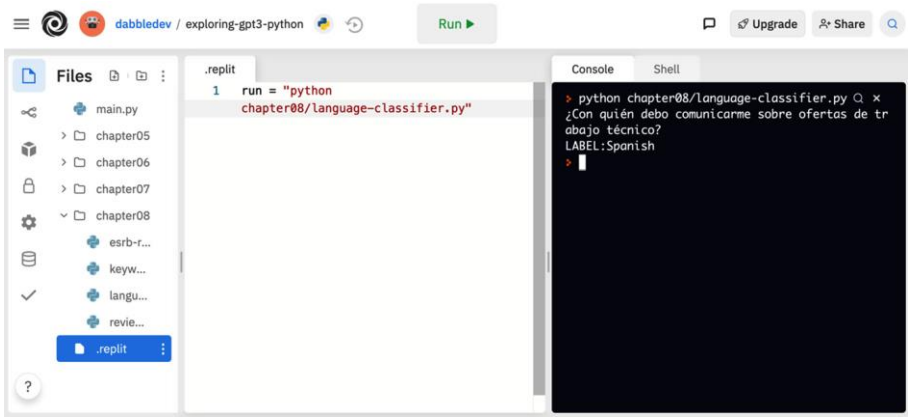

Figure 8.6 – Example output from chapter08/language-classifier.py

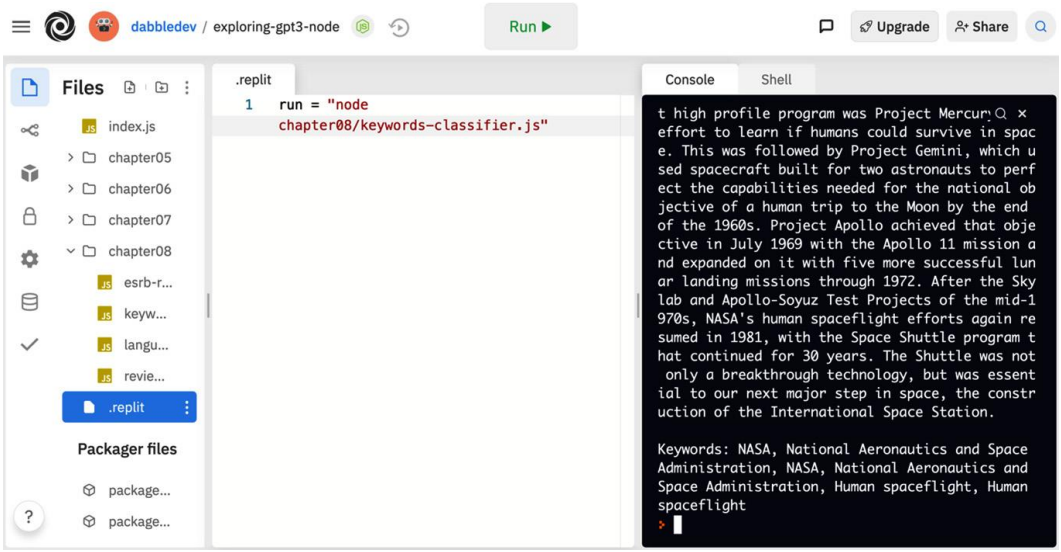

Figure 8.7 – Example output from chapter08/keywords-classifier.js

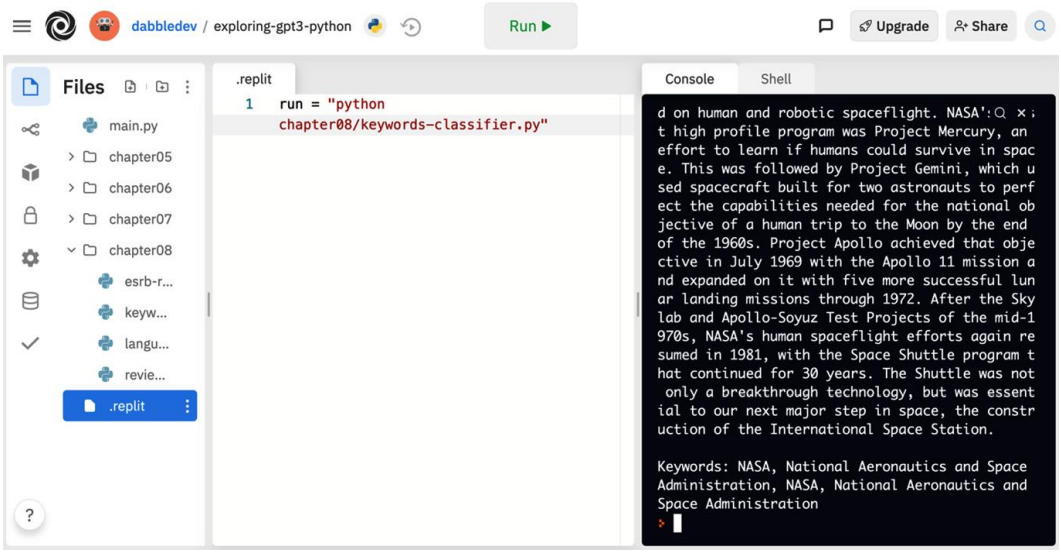

Figure 8.8 – Example output from chapter08/keywords-classifier.py

Chapter 9

Figures

Figure 9.1 – GPT Answers user interface

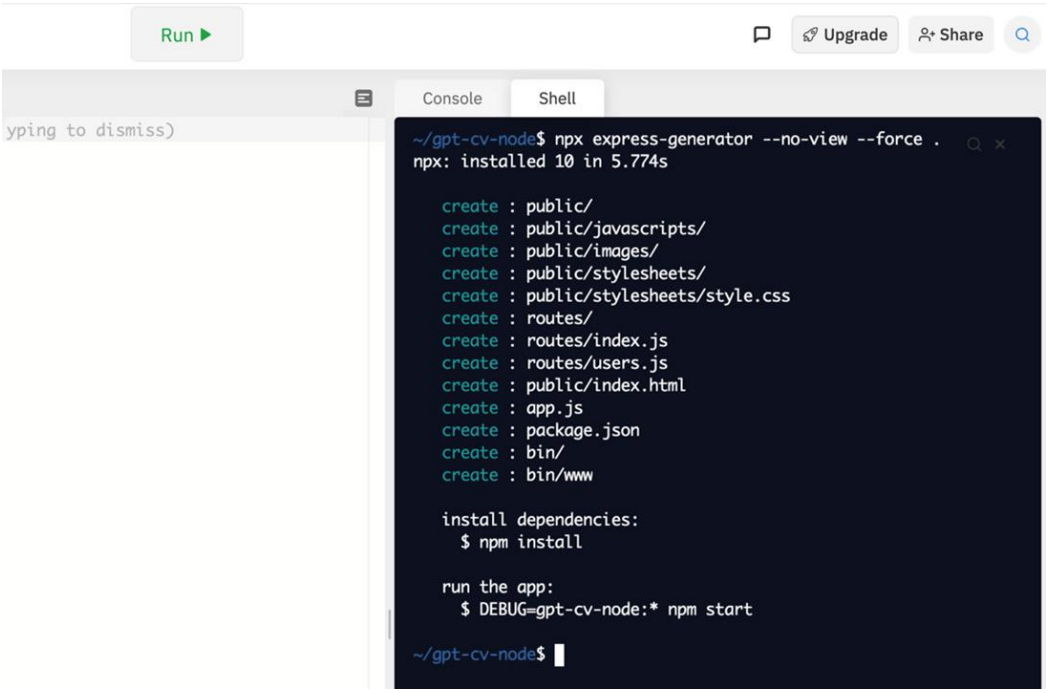

Figure 9.2 – Output from express-generator

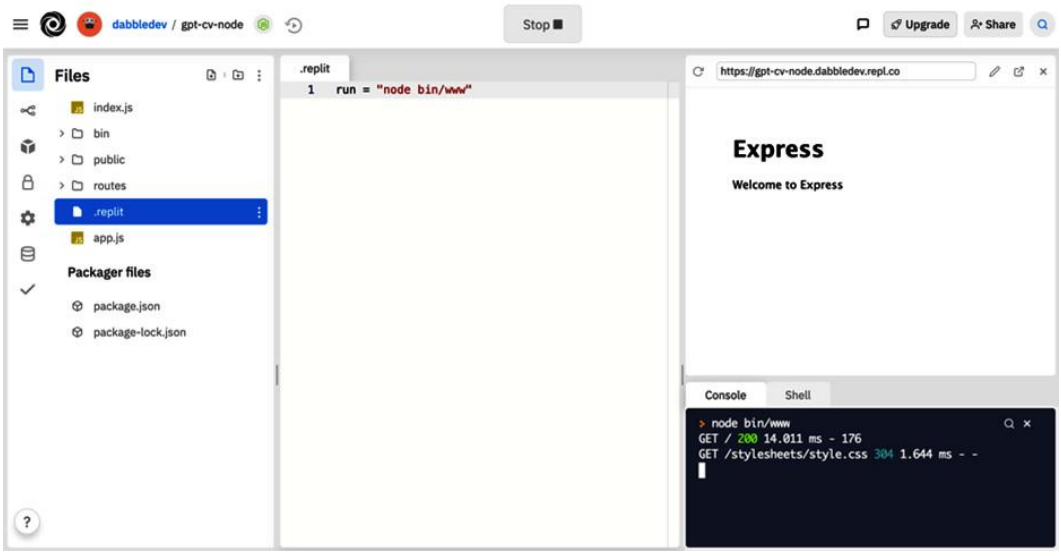

Figure 9.3 – Express server running in Replit.com

app.js

```
1  var express = require('express');
2  var path = require('path');
3  var cookieParser = require('cookie-parser');
4  var logger = require('morgan');
5
6  var indexRouter = require('./routes/index');
7  var usersRouter = require('./routes/users');
8
9  var app = express();
10
11 app.use(logger('dev'));
12 app.use(express.json());
13 app.use(express.urlencoded({ extended: false }));
14 app.use(cookieParser());
15 app.use(express.static(path.join(__dirname, 'public')));
16
17 app.use('/', indexRouter);
18 app.use('/users', usersRouter);
19
20 module.exports = app;
```

Figure 9.4 – Default app.js created by express-generator

app.js

```
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var answerRouter = require('./routes/answer');

var app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/answer', answerRouter);

module.exports = app;
```

Figure 9.5 – Edited app.js file

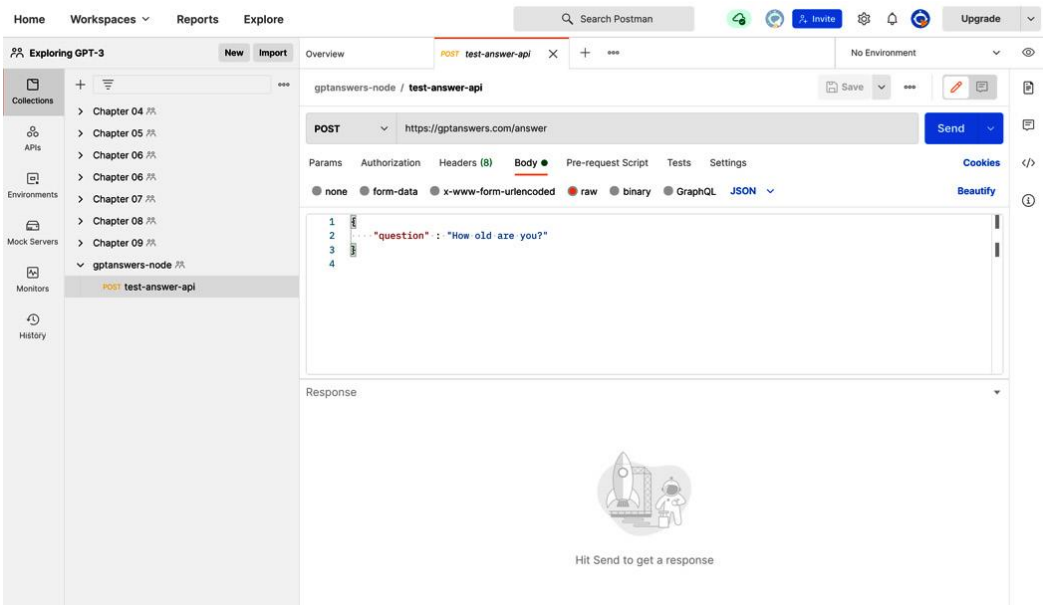

Figure 9.6 – Postman request to test the GPT-CV app API endpoint

public/index.html

```
<html>

<head>
  <title>GPT Answers</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/css/uikit.min.css">
</head>

<body>

  <div class="uk-section uk-section-large uk-height-viewport">
    <div class="uk-container uk-text-center uk-padding-large">
      <h1 class="uk-heading-medium"><strong>GPT</strong> Answers </h1>
      <p class="uk-text-lead">An Example Knowledge Base App Powered by GPT-3</p>
    </div>
    <div class="uk-container uk-text-center">
      <form class="uk-grid-collapse" uk-grid>
        <div class="uk-width-1-1 ">
          <input id="question" class="uk-input uk-width-1-3" type="text">
          <button type="submit" class="uk-button uk-button-default uk-width-1-5">Get Answer</button>
        </div>
      </form>
    </div>
    <div class="uk-container uk-text-center uk-padding">
      <div class="uk-inline">
        <div id="answer" class="uk-flex uk-flex-center uk-flex-middle uk-padding uk-width-expand"></div>
      </div>
    </div>
  </div>

  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/js/uikit.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/js/uikit-icons.min.js"></script>
  <script src="/javascripts/script.js"></script>
</body>

</html>
```

Figure 9.7 – Completed index.html code

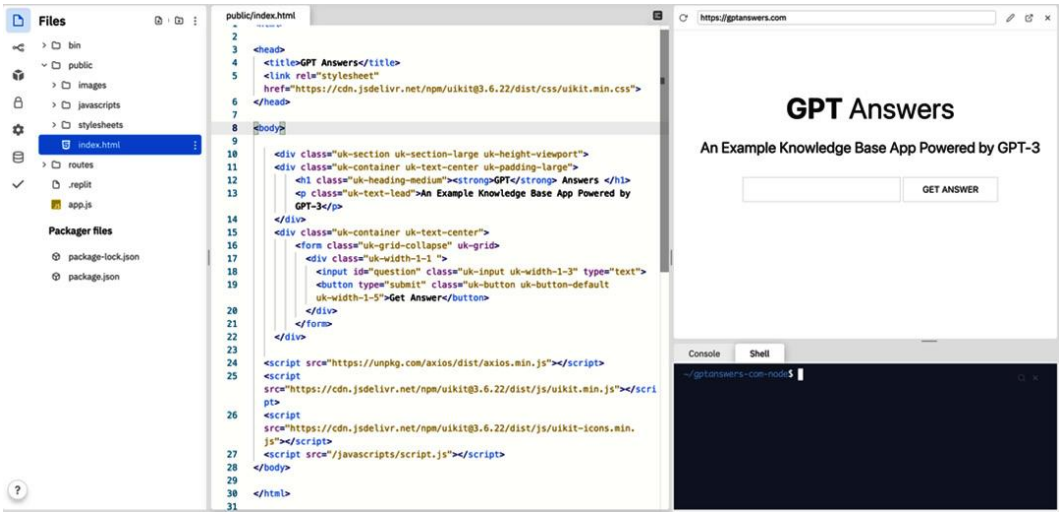

Figure 9.8 – Question input form

Figure 9.9 – Testing the web UI with the placeholder API code

routes/answer.js

```
const axios = require('axios');
const express = require('express');
const router = express.Router();

const apiKey = process.env.OPENAI_API_KEY;
const client = axios.create({
  headers: { 'Authorization': 'Bearer ' + apiKey }
});

const documents = [
  "I am a day older than I was yesterday.<|endoftext|>",
  "I build applications that use GPT-3.<|endoftext|>",
  "My preferred programming is Python.<|endoftext|>"
]

const endpoint = 'https://api.openai.com/v1/answers';

router.post('/', (req, res) => {
    const data = {
    "file": process.env.ANSWERS_FILE,
    "question": req.body.question,
    "search_model": "ada",
    "model": "curie",
    "examples_context": "My favorite programming language is Python.",
    "examples": [["How old are you?", "I'm a day older than I was yesterday."], ["What languages do you know?", "I speak English and write code in Python."]],
    "max_tokens": 15,
    "temperature": 0,
    "return_prompt": false,
    "expand": ["completion"],
    "stop": ["\n", "<|endoftext|>"],
   }
   client.post(endpoint, data)
    .then(result => {
      res.send({"answer" : result.data.answers[0]})
    }).catch(result => {
      res.send({"answer" : "Sorry, I don't have an answer."})
    });
});

module.exports = router;

```

Figure 9.10 – Edited routes/answer.js file

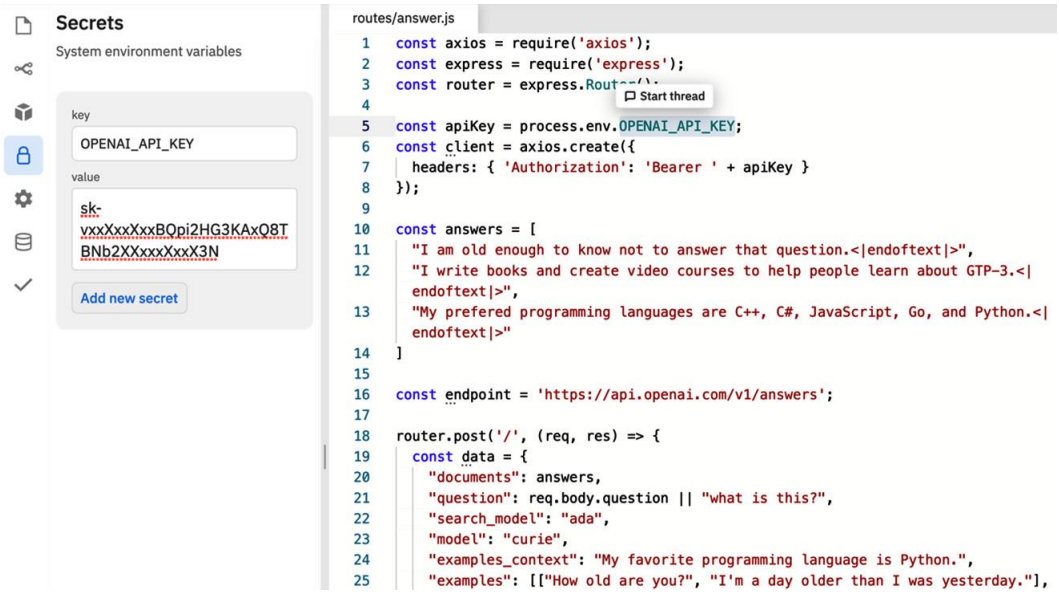

Figure 9.11 – Add a secret for your OpenAI API key

Figure 9.12 – Answer from GPT-3

files-upload.js

```
const axios = require('axios');
const FormData = require('form-data');

const data = new FormData();
data.append('purpose', 'answers');
data.append('file', fs.createReadStream('answers.jsonl'));

const params = {
  method: 'post',
  url: 'https://api.openai.com/v1/files',
  headers: {
    'Authorization': 'Bearer ' + process.env.OPENAI_API_KEY,
    ...data.getHeaders()
  },
  data: data
}

axios(params)
  .then(function(response) {
    console.log(response.data);
  })
  .catch(function(error) {
    console.log(error.message);
  });

```

Figure 9.13 – Completed code for file-upload.js

Console Shell

```
~/gptanswers-com-node$ node files-upload.js
{
  id: 'file-gGfXHNMv3Fm4J33ebWIobhkm',
  object: 'file',
  bytes: 132,
  created_at: 1621699209,
  filename: 'answers.jsonl',
  purpose: 'answers',
  status: 'uploaded',
  status_details: null
}
~/gptanswers-com-node$
```

Figure 9.14 – Shell output from files-upload.js

```
router.post('/', (req, res) => {
    const data = {
    "file": process.env.ANSWERS_FILE,
    "question": req.body.question,
    "search_model": "ada",
    "model": "curie",
    "examples_context": "My favorite programming language is Python.",
    "examples": [["How old are you?", "I'm a day older than I was yesterday."],
    ["What languages do you know?", "I speak English and write code in Python."]]
    ,
    "max_tokens": 15,
    "temperature": 0,
    "return_prompt": false,
    "expand": ["completion"],
    "stop": ["\n", "<|endoftext|>"],
  }
```

Figure 9.15 – The Answers endpoint parameters using the file parameter

Figure 9.16 – An answer generated from the answers file

Figure 9.17 – A question that can't be answered

Figure 9.18 – An answer that isn't from the answers file

Links

- The URL for the Answers endpoint is https://api.openai.com/v1/answers
- see the OpenAI docs for the Answers endpoint located at https://beta.openai.com/docs/api-reference/answers
- https://getuikit.com/

Code

Code 9.1 – In the output pane, click on the Shell tab and enter this command:

```
npx express-generator --no-view --force
```

Code 9.2 - After `express-generator` completes, run the following command in the shell:

```
npm update
```

Code 9.3 - Now create a file named `.replit` and add the following `Run` command to it:

```
Run = "node ./bin/www"
```

Code 9.4 – Edit *line 7* and change `var usersRouter = require('./routes/users')` to the following:

```
var answerRouter = require('./routes/answer');
```

Code 9.5 – Edit *line 18* and change `app.use('/users', usersRouter);` to the following:

```
app.use('/answer', answerRouter);
```

Code 9.6 - Add the following code to the `answers.js` file:

```
const axios = require('axios');
const express = require('express');
const router = express.Router();

router.post('/', (req, res) => {
res.send({answer:'placeholder for the answer'});
});

module.exports = router;
```

Code 9.7 – The format of endpoint URL

```
https://gptanswers-node.{username}.repl.co
```

Code 9.8 – Finally, add the following JSON for the request body:

```
{
"question" : "How old are you?"
}
```

Code 9.9 – Click the blue **Send** button to submit the request and review the response, which should be the following:

```
{
"answer": "placeholder for the answer"
}
```

Code 9.10 – Replace the URL for the style sheet with `https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/css/uikit.min.css`.

Code 9.11 – Replace everything between the `<body>` tag and the `</body>` tag with the following:

```
  <div class="uk-section uk-section-large uk-height-viewport">
```

```
  <div class="uk-container uk-text-center uk-padding-large">

        <h1 class="uk-heading-medium"><strong>GPT</strong> Answers
</h1>

        <p class="uk-text-lead">An Example Knowledge Base App Powered
by GPT-3</p>

    </div>

    <div class="uk-container uk-text-center">

        <form class="uk-grid-collapse" uk-grid>

          <div class="uk-width-1-1 ">

            <input id="question" class="uk-input uk-width-1-3"
type="text">

            <button type="submit" class="uk-button uk-button-default
uk-width-1-5">Get Answer</button>

          </div>

        </form>

    </div>

    <div class="uk-container uk-text-center uk-padding">

      <div class="uk-inline">

        <div id="answer" class="uk-flex uk-flex-center uk-flex-middle
uk-padding uk-width-expand"></div>

      </div>

    </div>

  </div>
```

Code 9.12 – Add code 9.12 above the `</body>` tag.

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>

<script
src="https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/js/uikit.min.js"><
/script>

<script src="https://cdn.jsdelivr.net/npm/uikit@3.6.22/dist/js/uikit-
icons.min.js"></script>

<script src="/javascripts/script.js"></script>
```

Code 9.13 – First, we'll add two variables to hold the HTML form and answer `div`:

```
const form = document.querySelector('form');

const answer = document.querySelector('#answer');
```

Code 9.14 – Next, we'll add code that will fire when the form is submitted:

```
const formEvent = form.addEventListener('submit', event => {
  event.preventDefault();
  const question = document.querySelector('#question');
  if (question.value) {
    askQuestion(question.value);
  } else {
    answer.innerHTML = "You need to enter a question to get an
answer.";
    answer.classList.add("error");
  }
});
```

Code 9.15 – The following code will append the text to the answer `div`:

```
const appendAnswer = (result) => {
  answer.innerHTML = `<p>${result.answer}</p>`;
};
```

Code 9.16 – Finally, we'll add a function to call the app API endpoint using Axios:

```
const askQuestion = (question) => {
  const params = {
    method: 'post',
    url: '/answer',
    headers: {
      'content-type': 'application/json'
    },
    data: { question }
```

```
  };
  axios(params)
    .then(response => {
      const answer = response.data;
      appendAnswer(answer);
    })
    .catch(error => console.error(error));
};
```

Code 9.17 – Starting on *line 5*, add the following code followed by a line return:

```
const apiKey = process.env.OPENAI_API_KEY;
const client = axios.create({
  headers: { 'Authorization': 'Bearer ' + apiKey }
});
```

Code 9.18 – Next, add the following code with a line return after it:

```
const documents = [
  "I am a day older than I was yesterday.<|endoftext|>",
  "I build applications that use GPT-3.<|endoftext|>",
  "My preferred programming is Python.<|endoftext|>"
]
```

Code 9.19 – Add the following code starting on *line 16* followed by a line return:

```
const endpoint = 'https://api.openai.com/v1/answers';
```

Code 9.20 – Next, beginning on *line 18*, add the following to complete the code:

```
router.post('/', (req, res) => {
  // call the OpenAI API
    const data = {
    "documents": documents,
```

```
    "question": req.body.question,
    "search_model": "ada",
    "model": "curie",
    "examples_context": "My favorite programming language is Python.",
    "examples": [["How old are you?", "I'm a day older than I was
yesterday."], ["What languages do you know?", "I speak English and
write code in Python."]],
    "max_tokens": 15,
    "temperature": 0,
    "return_prompt": false,
    "expand": ["completion"],
    "stop": ["\n", "<|endoftext|>"],
  }
  client.post(endpoint, data)
    .then(result => {
      res.send({"answer" : result.data.answers[0]})
    }).catch(result => {
      res.send({"answer" : "Sorry, I don't have an answer."})
    });
});
module.exports = router;
```

Code 9.21 – Create a new `jsonl` file named `answers.jsonl` and some answers for the file in the following format:

```
{"text": "I am a day younger than I will be tomorrow"}
{"text": "I like to code in Python."}
{"text": "My favorite food is carrot cake."}
```

Code 9.22 – Create another new file named `files-upload.js` and add the following code to it. Require a few modules that will be used:

```
const fs = require('fs');
const axios = require('axios');
```

```
const FormData = require('form-data');
```

Code 9.23 – Next, add the following code to read in the `jsonl` data for the request:

```
const data = new FormData();
data.append('purpose', 'answers');
data.append('file', fs.createReadStream('answers.jsonl'));
```

Code 9.24 – Add a variable for the HTTP request parameters:

```
const params = {
method: 'post',
url: 'https://api.openai.com/v1/files',
headers: {
  'Authorization': 'Bearer ' + process.env.OPENAI_API_KEY,
  ...data.getHeaders()
},
data : data
}
```

Code 9.25 – Finally, add code to make the HTTP request and log results:

```
axios(params)
  .then(function(response) {
    console.log(response.data);
  })
  .catch(function(error) {
    console.log(error);
  });
```

Code 9.25 – At the `~/gptanswers-node` prompt, enter the following command with your OpenAI API key:

```
export OPENAI_API_KEY="your-api-key-goes-here"
```

Code 9.26 – Next, enter the following command in the shell:

```
node files-upload.js
```

Code 9.27 – Run the following shell command to set your API key as an environment variable that the shell can access:

```
export OPENAI_API_KEY="your-api-key-goes-here"
```

Code 9.28 – Run the following command in the shell to execute `files-upload.js`:

```
node files-upload.js
```

Chapter 10

Links

- https://beta.openai.com/docs/use-case-guidelines
- https://beta.openai.com/docs/safety-best-practices
- https://beta.openai.com/forms/pre-launch-review
- https://beta.openai.com/forms/quota-increase

Code

Code 10.1 – Require the bad-words library on the first line of routes/answer.js with the following code:

```
const Filter = require('bad-words');
```

Code 10.2 – In the `routes/answer.js` file, add the following code above the line that begins with `const data`:

```
let filter = new Filter();
if (filter.isProfane(req.body.question)) {
    res.send({ "answer": "That's not a question we can answer."});
    return;
}
```

Code 10.3 - Open `routes/answer-dot-JS` and add the following code on a new line after the line that begins with `router.post`:

```
if (req.body.question.length > 150) {
  res.send({ "answer": "Sorry. That question is too long." });
  return;
}
```

Code 10.4 – Open `app.js` and after *line 9* (or after `var app = express();`), add the following code:

```
const rateLimit = require("express-rate-limit");
```

```
const apiLimiter = rateLimit({
  windowMs: 1 * 60 * 1000,
  max: 6
});
app.use("/answer/", apiLimiter);
```

Code 10.5 – Open `routes/answer-dot-JS` and add code 10.5 after the line that begins with `router.post`:

```
if (req.rateLimit.remaining == 0) {
  res.send({"answer" : "Ask me again in a minute."});
  return;
};
```

Figures

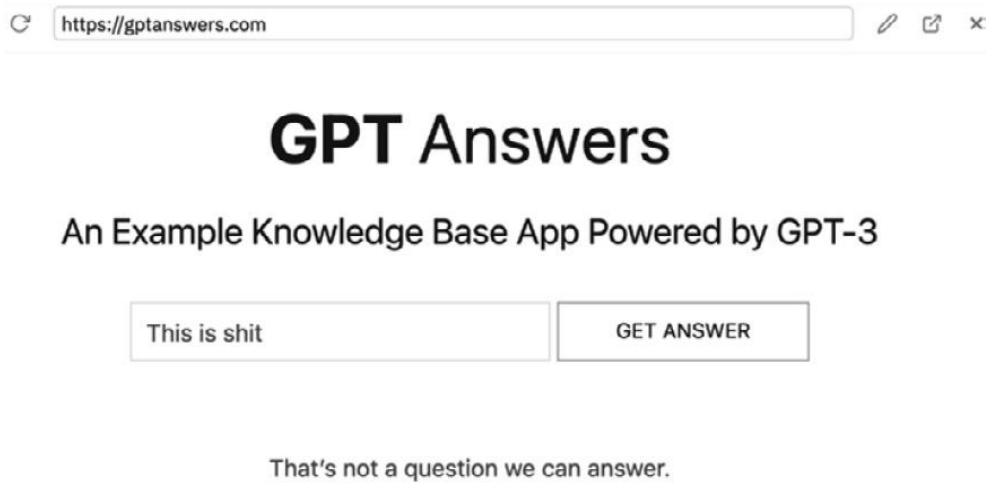

Figure 10.1 – Filtering profanity in questions

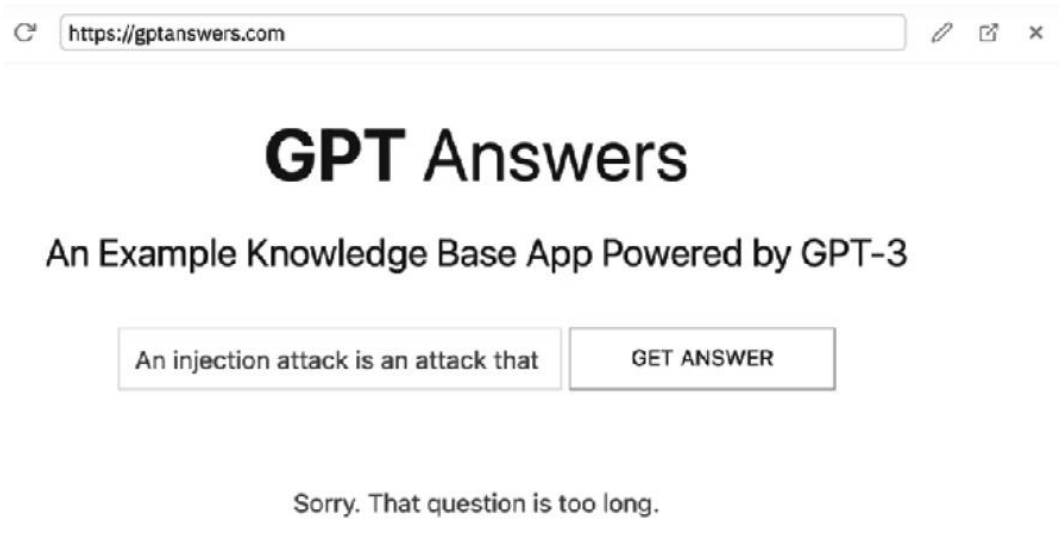

Figure 10.2 – Form output with long text

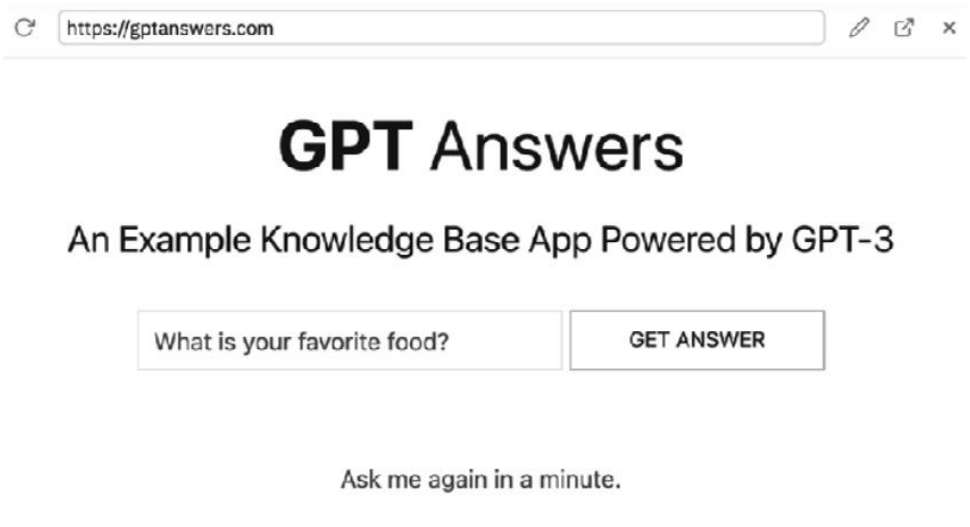

Figure 10.3 – Message when request rate is exceeded

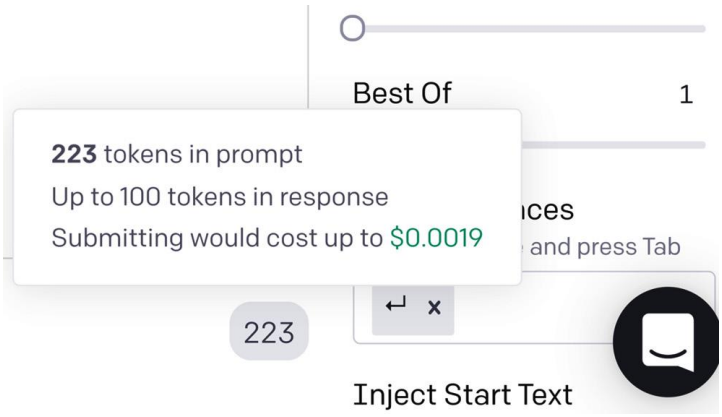

Figure 10.4 – Estimated cost