7

The Generative AI Revolution with ChatGPT

In November 2022, OpenAI ChatGPT entered mainstream media with a resonating big bang. Newspapers, television channels, and social media raced to OpenAI's ChatGPT website. Rumors of what ChatGPT could do spread like wildfire. Since then, OpenAI has continually updated its platform.

AI history is on a roll!

This chapter continues the journey of this book and takes the reader further down the path of learning about the ever-growing power of transformers. The theory and practical knowledge acquired through the previous chapters provide enough transformer model expertise to enjoy the ride through the cutting-edge models in this chapter.

We will first go through the tremendous improvements and diffusion of ChatGPT models in the everyday lives of developers and end users. We will see that GPT is a General Purpose Technology: GPTs are GPTs that can be applied to a wide range of domains.

This chapter will then examine the improvements in the architecture of the Generative AI GPT models. We will investigate the zero-shot challenge of using trained transformer models with little to no fine-tuning of the model's parameters for downstream tasks.

The tremendous impact of generative transformer assistants will be explored. We will then get started with OpenAI models as assistants.

Then, we will get started with the GPT-4 API, the hyperparameters, and implement several NLP examples.

Finally, we will learn how to obtain better results with **Retrieval Augmented Generation** (**RAG**). We will implement an example of automated RAG with GPT-4.

By the end of the chapter, you will have fully understood the many perspectives OpenAI's transformer models open for you.

This chapter covers the following topics:

- The rise and diffusion of GPT models as a General Purpose Technology
- The architecture of OpenAI GPT models
- The path from few-shot to one-shot models
- Getting started with ChatGPT as an assistant
- Getting started with the OpenAI GPT-4 API
- Implementing an introductory automated RAG process for GPT-4

> With all the innovations and library updates in this cutting-edge field, packages and models change regularly. Please go to the GitHub repository for the latest installation and code examples:

> `https://github.com/Denis2054/Transformers-for-NLP-and-Computer-Vision-3rd-Edition/tree/main/Chapter07`.
>
> You can also post a message in our Discord community (`https://www.packt.link/Transformers`) if you have any trouble running the code in this or any chapter.

Let's begin our journey by exploring GPTs as **GPTs** (**General Purpose Technologies**).

GPTs as GPTs

This chapter unveils the power of generative artificial intelligence models. **GPT** (**Generative Pre-trained Transformers**) reveals its meaning with ChatGPT models, such as GPT 3.5-Turbo and GPT-4. There are 50+ variants of the main OpenAI GPT models. Among those variants, some will be regularly deprecated, and new ones will appear.

The evolution of OpenAI models has confirmed that Generative AI transformers, such as GPTs, are GPTs as defined by *Eloundou et al.* (2023).

Once an innovation becomes a general-purpose technology, it spreads to thousands of applications, like the invention of electricity, combustion engines, and computers. As such, Generative Pre-trained Transformers technology, often referred to as Generative

AI, has begun to spread to thousands of applications in many domains.

In this section, we will approach the maze of OpenAI models, Generative AI, and applications from the perspective of a general-purpose technology. We will focus on two main aspects: *improvement* and *diffusion*.

Let's begin with the improvements.

Improvement

In this chapter, the focus of improvement will be on the architecture of OpenAI transformer models. The focus will be on four improvements: decoder only, scale, task generalization, and new terminology:

- **Decoder only**

  The Original Transformer described in *Chapter 2, Getting Started with the Architecture of the Transformer Model*, contained an encoder and decoder stack. *Chapter 5, Diving into Fine-Tuning through BERT*, introduced BERT, an encoder-only stack. This chapter will present a decoder-only stack. You may ask yourself what mathematical chain of logic or proof leads to choosing one of these configurations. There is none!

  The development of transformer models involves empirical data-driven insights, hardware constraints, and evaluations. This explains why they will constantly evolve through the intuitive and creative minds of their architects.

- **Scale**

Scale remains a critical feature in transformers. GPT models have increased in size, as you will discover in this section. Why? The goal is to capture the many dependencies between words and contexts. One word can have many different meanings depending on the context. For example, the verb *eat* seems simple. Right? But we quickly discover that somebody can *eat* something, or something can be *eaten*, that somebody might want *to eat*, *not eat*, or *maybe eat*. The list is nearly endless! We can create many parameters to express these subtleties.

The issue then becomes finding the right number of parameters. Too many parameters might be costly and useless. Too few parameters might reduce accuracy. The right number of parameters will be obtained through trial and error.

- **Task generalization**

  If a model is trained to a specific task, it's task-specific, such as the model we fine-tuned and trained in *Chapter 5, Diving into Fine-Tuning through BERT.* We provide an input and get output such as a word, a classification, an answer to a question, and other NLP tasks.

  However, when faced with potentially hundreds of tasks, we can't imagine creating hundreds of task-specific models! That is where Generative AI models such as OpenAI GPT models fit in.

  You can sum this up as, "We start a sentence, and the model finishes it." For example, you might say to a friend, "Give me the recipe for your delicious apple pie, please."

  Think about it. What do you expect? A list of cars? Instructions to build a house? No, you expect a list of ingredients and instructions to cook an apple pie. Transformers have been trained on huge datasets and will statistically determine the following word(s).

You started the sentence by designing the "prompt," the other person or GPT will just continue the sentence by producing the "response."

You can easily imagine thousands of tasks you could create by simply beginning a sentence with a clear context and waiting for GPT to continue it! You just entered the world of general-purpose, Generative AI.

- **New terminology**

  New words appear with the latest technology, such as LLMs, Generative AI, and Foundation Models.

  Do not let yourself be overwhelmed by these terms. Just get used to them and the concepts they represent like any other new words. For example, OpenAI GPT models now have billions of parameters to process natural language. They are thus "Large Language Models."

  A GPT model can continue a sentence, a "prompt," which explains why they are Generative AI models. GPT models can process words, images, and sounds. We can build hundreds of tasks on their capabilities. This makes them Foundation Models that we can build other systems with.

Keep these takeaways in mind when you work your way through this chapter.

Orientating ourselves in the *diffusion* of GPTs is equally essential.

Diffusion

The diffusion of ChatGPT models can be explained by the new application sectors and their pervasiveness.

New application sectors

New application sectors are typical of general purpose technology. There is no need to feel overtaken or overwhelmed. Once electricity was invented, thousands of products and services appeared on the market. As these applications spread, general purpose technology is continually evolving and improving.

This chapter will review some of the main products and services powered by OpenAI models.

We can break this list down into self-service assistant and development assistant applications.

Self-service assistants

Self-service interfaces require no development and no machine learning knowledge at all:

- ChatGPT online to chat, create images, analyze data, and more.
- ChatGPT online to provide text instructions.
- OpenAI Playground: for general-purpose NLP tasks.
- Microsoft Office 365 Copilot spreading to many applications.
- New Bing to chat, give instructions, obtain responses, or use as a search engine.

Development assistants

Development assistants have become unavoidable game changers, such as the following tools:

- ChatGPT and OpenAI Playground as a code assistant.
- GitHub Copilot: code assistant (in an application).
- New Bing as a code assistant.

We can further state that the new application sectors are *assistants* for any user in many applications, whether to develop AI applications, classical applications, or for everyday usage.

Pervasiveness

Do not underestimate pervasiveness. Pervasiveness pushes innovations out of the shadows into everyday life. An invention only becomes an innovation when it exerts an impact on society. Then, it often spills over into many domains at an inevitable exponential speed under the pressure of demand and competition.

In the case of transformers, *early adopters* are the ones who started working on transformers as early as November 2017 when transformers emerged, as explained in *Chapter 1, What Are Transformers?*

Those who accept a technological advance through imitation (media, social media, mainstream resources) are *new adopters*.

The big bang of *new adopters* began in November 2022 when OpenAI made ChatGPT available to mainstream end users and AI professionals. The number of new adopters increases through a classical imitation effect of general purpose technologies.

If we break down the number of new adopters from the imitation effect, we get the following:

- New adopters who are just curious enough to persuade others to join the trend.
- New adopters who use Generative AI, including ChatGPT, for the fun of it and keep up with the pace of technology.
- New adopters who begin to create prompts that generate content that boosts their productivity.
- New adopters who know nothing about ML but are creating functions and applications that compete with AI professionals.
- New adopters who are AI professionals boost their productivity with LLMs.

We are living in a typical acceleration of the diffusion of AI as general-purpose technology through Generative AI such as GPTs.

Companies will not take long to realize they do not need a data scientist or an AI specialist to start an NLP project with the tools available in every key application.

So why bother?

The answer to these questions is quite simple. It’s easy to start a GPT-3 or a GPT-4 engine, just like starting a Formula 1 or Indy 500 race car. No problem. But then, driving such a car is nearly impossible without months of training! Generative AI engines, such as GPT-3 and GPT-4 are powerful AI race cars. You can get them to run in a few clicks. However, leveraging their incredible horsepower requires the knowledge you have acquired from the beginning of this book and what you will discover in the following chapters!

We will now go through the architecture that made transformers so popular.

The architecture of OpenAI GPT transformer models

In 2020, *Brown et al.* (2020) described the training of an OpenAI GPT-3 model containing 175 billion parameters that was trained on huge datasets, such as the 400 billion byte-pair-encoded tokens extracted from Common Crawl data. OpenAI ran the training on a Microsoft Azure supercomputer with 285,00 CPUs and 10,000 GPUs.

The machine intelligence of OpenAI's GPT-3 models and their supercomputer led *Brown et al*. (2020) to zero-shot experiments. The idea was to use a trained model for downstream tasks without further training the parameters. The goal would be for a trained model to go directly into multi-task production with an API that could even perform tasks it wasn't trained for.

The era of suprahuman cloud AI models was born. OpenAI's API requires no high-level software skills or AI knowledge. You might wonder why I use the term "suprahuman." GPT-3 and the GPT-4 model (and soon more powerful ones) can perform many tasks at least as well as a human in many cases. For the moment, *it is essential to understand how GPT models are built and run to appreciate the magic*.

GPT-4 is built on GPT-3, which in turn is built on the GPT-2 architecture. However, a fully trained GPT-3 transformer is a Foundation Model:

- A Foundation Model can do many tasks it wasn't trained for through emergence.
- Through *homogenization*, GPT-3/GPT-4 generative abilities through a unified architecture apply to many NLP tasks, including programming tasks.

Transformers went from training to fine-tuning and finally to zero-shot models in less than three years between the end of 2017 and the first part of 2020. A zero-shot GPT-3 transformer model requires no fine-tuning. The trained model parameters are not updated for downstream multi-tasks, which opens a new era for NLP/NLU tasks.

In this section, we will first learn about the OpenAI team's motivation for designing GPT models.

We will first go through the creation process of the OpenAI team.

The rise of billion-parameter transformer models

The speed at which transformers went from small models trained for NLP tasks to models that require little to no fine-tuning is staggering.

Vaswani et al. (2017) introduced the Original Transformer, which surpassed CNNs and RNNs on BLEU tasks. *Radford et al.* (2018) introduced the GPT model, which could perform downstream tasks with fine-tuning. *Devlin et al.* (2019) perfected fine-tuning with the BERT model. *Radford et al.* (2019) went further with GPT-2 models. *Brown et al.* (2020) defined a GPT-3 zero-shot approach to transformers that does not require fine-tuning!

At the same time, *Wang et al*. (2019) created GLUE to benchmark NLP models. But transformer models evolved so quickly that they surpassed human baselines!

Wang et al. (2019^2) rapidly created SuperGLUE, set the human baselines much higher, and made the NLU/NLP tasks more challenging. Transformers are rapidly progressing, and some have already surpassed Human Baselines on the SuperGLUE leaderboards at the time of writing.

How did this happen so quickly?

We will look at one aspect, the models' sizes, to understand how this evolution happened.

The increasing size of transformer models

From 2017 to 2020 alone, the number of parameters increased from 65M parameters in the Original Transformer model to 175B parameters in the GPT-3 model, as shown in *Table 7.1*:

Transformer Model	**Paper**	**Parameters**
Transformer Base	*Vaswani et al*. (2017)	65M
Transformer Big	*Vaswani et al*. (2017)	213M
BERT-Base	*Devlin et al*. (2019)	110M

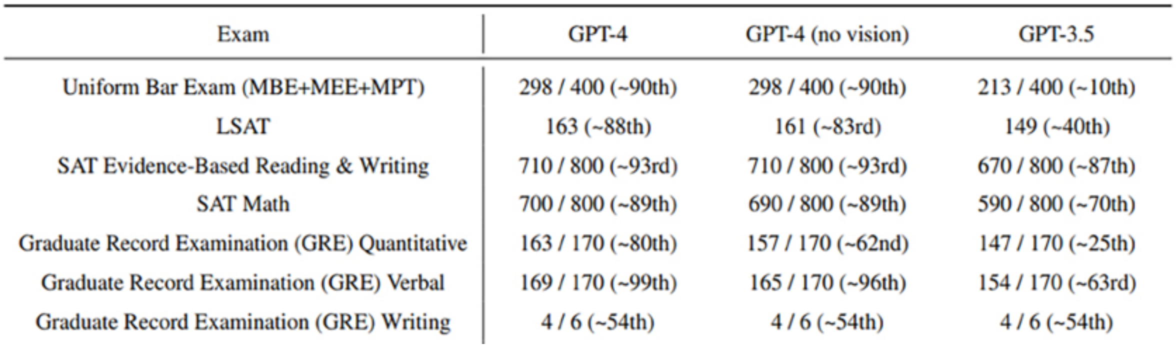
BERT-Large	*Devlin et al.* (2019)	340M
GPT-2 Small	*Radford et al.* (2019)	117M
GPT-2 Medium	*Radford et al.* (2019)	345M
GPT-2 Large	*Radford et al.* (2019)	1.5B
GPT-3	*Brown et al.* (2020)	175B
GPT-3.5	OpenAI (March 2022)	175B
GPT-4	OpenAI (March 2023)	-
GPT-4V	OpenAI (September 2023)	

Table 7.1: The evolution of the number of transformer parameters

Table 7.1 only contains the main models designed during that short time. The dates of the publications come after the date the models were actually designed. Also, the authors updated the papers. For example, once the Original Transformer set the market in motion, transformers emerged from Google Brain and Research, OpenAI, and Facebook AI, which all produced new models in parallel.

The number of parameters has increased significantly in a relatively short time, reaching 175 billion parameters for GPT-3. GPT-3.5 and GPT-3.5-turbo have 175B parameters.

Information on the architecture of GPT-4 models is scarce. OpenAI has not officially disclosed the details of GPT-4's architecture. However, they optimized the system and obtained high scores on well-known exams and evaluations, as shown in the following excerpt of the *GPT-4 Technical Report, March 23, 2023, page 5*:



Exam	GPT-4	GPT-4 (no vision)	GPT-3.5
Uniform Bar Exam (MBE+MEE+MPT)	298 / 400 (~90th)	298 / 400 (~90th)	213 / 400 (~10th)
LSAT	163 (~88th)	161 (~83rd)	149 (~40th)
SAT Evidence-Based Reading & Writing	710 / 800 (~93rd)	710 / 800 (~93rd)	670 / 800 (~87th)
SAT Math	700 / 800 (~89th)	690 / 800 (~89th)	590 / 800 (~70th)
Graduate Record Examination (GRE) Quantitative	163 / 170 (~80th)	157 / 170 (~62nd)	147 / 170 (~25th)
Graduate Record Examination (GRE) Verbal	169 / 170 (~99th)	165 / 170 (~96th)	154 / 170 (~63rd)
Graduate Record Examination (GRE) Writing	4 / 6 (~54th)	4 / 6 (~54th)	4 / 6 (~54th)

Figure 7.1: GPT model performance on specific exams

GPT-4 (expanded into GPT-4V) is muti-modal (language and vision). We will discover transformers for computer vision starting in *Chapter 16, Beyond Text: Vision Transformers in the Dawn of Revolutionary AI*.

ChatGPT is an umbrella term that covers GPT-3.5-turbo, GPT-4, GPT-4V, and possible future improvements.

The size of the architecture evolved at the same time:

- The number of layers of a model went from 6 layers in the Original Transformer model to 96 layers in the GPT-3 model.
- The number of heads of a layer went from 8 in the Original Transformer model to 96 in the GPT-3 model.
- The context size went from 512 tokens in the Original Transformer model to 12,288 in the davinci version of the GPT-3

model.

The architecture's size explains why GPT-3 175B, with its 96 layers, produces more impressive results than GPT-2 1,542M, with only 40 layers. The parameters of both models are comparable, but the number of layers has doubled.

Let's focus on the context size to understand another aspect of the rapid evolution of transformers.

Context size and maximum path length

The cornerstone of transformer models resides in the attention sub-layers. In turn, the key property of attention sub-layers is the method used to process context size.

Context size is one of the main ways humans and machines can learn languages. The larger the context size, the more we can understand a sequence.

However, the drawback of context size is the distance it takes to understand what a word refers to. The path taken to analyze long-term dependencies requires changing from recurrent to attention layers.

The following sentence requires a long path to find what pronoun "it" refers to:

"Our *house* was too small to fit a big couch, a large table, and other furniture we would have liked in such a tiny space. We thought about staying for some time, but finally, we decided to sell *it*."

The meaning of "it" can only be explained if we take a long path back to the word "house" at the beginning of the sentence. That's quite a path for a machine!

Vaswani et al. (2017) optimized the design of context analysis in the Original Transformer model. Attention brings the operations down to a one-to-one token operation. All the layers are identical, making it much easier to scale up the size of transformer models.

The flexible and optimized architecture of transformers has led to an impact on several other factors:

- *Vaswani et al.* (2017) trained a state-of-the-art transformer model with 36M sentences. *Brown et al.* (2020) trained a GPT-3 model with 400 billion byte-pair-encoded tokens extracted from Common Crawl data.
- Training large transformer models requires machine power that is only available to a few teams worldwide. It took a total of $2.14*10^{23}$ FLOPS for *Brown et al.* (2020) to train GPT-3 175B.
- Designing the architecture of transformers requires highly qualified teams that can only be funded by a small number of organizations worldwide.

The size and architecture will continue to evolve. Supercomputers will continue to provide the necessary resources to train transformers.

We will now see how zero-shot models were achieved.

From fine-tuning to zero-shot models

From the start, OpenAI's research teams, led by *Radford et al.* (2018), wanted to take transformers from former trained models to GPT models. The goal was to train transformers on unlabeled data.

Letting attention layers learn a language from unsupervised data was a smart move. Instead of teaching transformers to do specific NLP tasks, OpenAI trained transformers to learn a language.

> Note: The term *unsupervised* defines training with no labels. In that sense, GPT models go through *unsupervised* training. However, when it predicts a token during training, the output is compared to the actual complete input sequence to calculate the loss, find the gradients, and perform backpropagation. In this sense, GPT is *rather self-supervised* than totally *unsupervised*. Keep this in mind when you see the term unsupervised associated with generative models.

OpenAI wanted to create a task-agnostic model. So, they began to train transformer models on raw data instead of relying on labeled data by specialists. Labeling data is time-consuming and considerably slows down the transformer's training process.

The first step was to start with unsupervised training in a transformer model. Then, they would only fine-tune the model's supervised learning.

OpenAI opted for a decoder-only transformer described in the stacking decoder layers section. The metrics of the results were convincing and quickly reached the level of the best NLP models of fellow NLP research labs.

The promising results of the first version of GPT transformer models soon led *Radford* et al. (2019) to come up with zero-shot transfer models. The core of their philosophy was to continue training GPT models to learn from raw text. They then took their research a step further, focusing on language modeling through examples of unsupervised distributions.

The goal was to generalize this concept to any downstream task once the trained GPT model understands a language through intensive training.

The GPT models rapidly evolved from 117M parameters to 345M parameters, to other sizes, and then to 1,542M parameters. 1,000,000,000+ parameter transformers were born. The amount of fine-tuning was sharply reduced.

This encouraged OpenAI to go further, much further. *Brown et al.* (2020) went on the assumption that conditional probability transformer models could be trained in depth and were able to produce excellent results with little to no fine-tuning for downstream tasks.

OpenAI was reaching its goal of training a model and running downstream tasks directly without further fine-tuning. This phenomenal progress can be described in four phases:

- **Fine-Tuning** (**FT**) is meant to be performed in the sense we have been exploring in previous chapters. A transformer model is trained and then fine-tuned on downstream tasks. *Radford et al.* (2018) designed many fine-tuning tasks. The OpenAI team then progressively reduced the number of tasks to `0` in the following steps.
- **Few-Shot** (**FS**) represents a huge step forward. The GPT is trained. When the model needs to make inferences, it is

presented with demonstrations of the task to perform as conditioning. Conditioning replaces weight updating, which the GPT team excluded from the process. We will apply conditioning to our model through the context we provide to obtain text completion in the notebooks we will go through in this chapter.

- **One-Shot** (**1S**) takes the process further. The trained GPT model is presented with only one demonstration of the downstream task to perform. No weight updating is permitted, either.
- **Zero-Shot** (**ZS**) is the ultimate goal. The trained GPT model is presented with no demonstration of the downstream task to perform.

Each of these approaches has various levels of efficiency. The OpenAI GPT team has worked hard to produce these state-of-the-art transformer models.

We can now explain the motivations that led to the architecture of the GPT models:

- Teaching transformer models how to learn a language through extensive training.
- Focusing on language modeling through context conditioning.
- The transformer takes the context and generates text completion in a novel way. Instead of consuming resources on learning downstream tasks, it works on understanding the input and making inferences no matter what the task is.
- Finding efficient ways to train models by masking portions of the input sequences forces the transformer to think with

> machine intelligence. Thus, machine intelligence, though not human, is efficient.

We understand the motivations that led to the architecture of GPT models. Let's now have a look at the decoder-layer-only GPT model.

Stacking decoder layers

We now understand that the OpenAI team focused on language modeling. Therefore, it makes sense to keep the masked attention sublayer. *Brown et al.* (2020) dramatically increased the size of the decoder-only transformer models to get excellent results.

GPT models have the same structure as the decoder stacks of the Original Transformer designed by *Vaswani et al.* (2017). We described the decoder stacks in *Chapter 2, Getting Started with the Architecture of the Transformer Model*. If necessary, take a few minutes to go back through the architecture of the Original Transformer.

The GPT model has a decoder-only architecture, as shown in *Figure 7.2*:

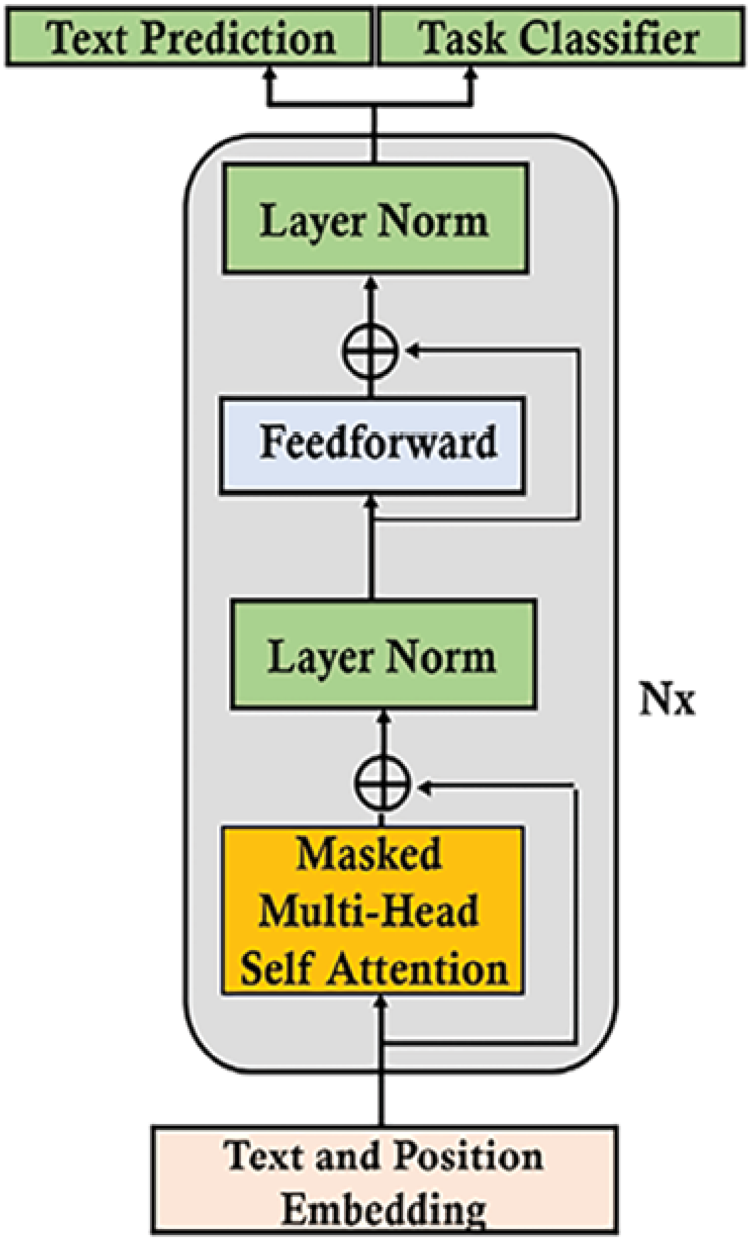


Figure 7.2: GPT decoder-only architecture

We can recognize the text and position embedding sub-layer, the masked multi-head self-attention layer, the normalization sub-layers, the feedforward sub-layer, and the outputs.

The OpenAI team customized and tweaked the decoder model by model. *Radford* et al. (2019) presented no fewer than four GPT models, and *Brown* et al. (2020) described no fewer than eight models.

The GPT-3 175B model has reached a unique size that requires computer resources that few teams in the world can access:

$$n_{params} = 175.0\text{B},\ n_{layers} = 96,\ d_{model} = 12288,\ n_{heads} = 96$$

OpenAI produces a significant number of models and variations.

GPT models

OpenAI is continually evolving. New models appear while others become deprecated. Some are upgraded, function calls change, API interfaces evolve, and the documentation goes through many updates. This is normal for a general-purpose technology, as explained in the *GPTs are GPTs* section of this chapter. We have to keep up with this fast-paced technology. Fortunately, OpenAI has online resources that inform us of the evolution of their models.

OpenAI's model documentation page that details the categories and lists of models: `https://platform.openai.com/docs/models`

The list contains the main domains of OpenAI models, such as language, vision, embedding, and moderation. We will implement these domains in this chapter and in the following chapters.

For example, we will implement the embeddings model in *Chapter 11, Leveraging LLM Embeddings as an Alternative to Fine-Tuning*. We will implement the moderation and whisper models in *Chapter 15, Guarding the Giants: Mitigating Risks in Large Language Models*. We begin implementing vision models (OpenAI and others) in *Chapter 16, Beyond Text: Vision Transformers in the Dawn of Revolutionary AI*.

OpenAI also provides a model deprecation page that details the retirement dates of the “older” models in this fast-moving market: `https://platform.openai.com/docs/deprecations`.

Let’s now open the hood of OpenAI and look at the models and variants available. This notebook will help you see the list of models and variants while consulting OpenAI’s documentation.

Open `OpenAI_Models.ipynb`. We will first install OpenAI:

```
try:
  import openai
except:
  !pip install openai
  import openai
```

Then retrieve the OpenAI `API_KEY` from a file:

```
from google.colab import drive
drive.mount('/content/drive')
f = open("drive/MyDrive/files/api_key.txt", "r")
API_KEY=f.readline()
f.close()
```

You can load the `API_KEY` from a file or enter it directly if you don’t wish to hide it.

Now, we can set an environment variable with the `API_KEY`:

```
import os
os.environ['OPENAI_API_KEY'] =API_KEY
openai.api_key = os.getenv("OPENAI_API_KEY")
```

We now create a list of models and engines:

```
elist=openai.models.list()
print(elist)
```

We want to know how many models and engines there are:

```
count = 0
for model in elist:
    count += 1
print("Number of models:", count)
```

The output shows the number of models and engines:

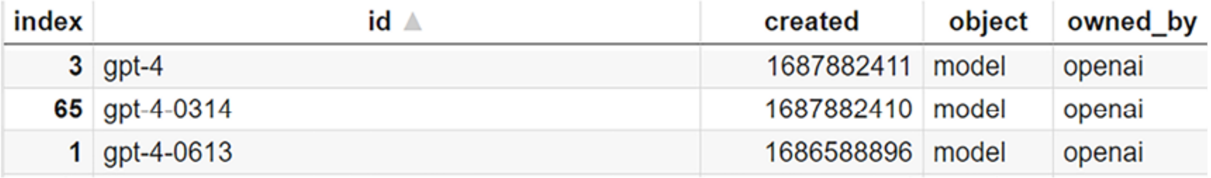
```
Number of models: 81
```

It is good practice to verify the list regularly along with OpenAI’s documentation (model and deprecation pages) as this list and number continually evolves.

The following cell displays a sorted list of `models` in `pandas` (note that the elist object itself can evolve as well):

```
import pandas as pd
model_data = []
# Iterate through each model in elist and collect the required informa
for model in elist:
    model_info = {
        'id': model.id,
        'created': model.created,
        'object': model.object,
        'owned_by': model.owned_by
    }
    model_data.append(model_info)
# Create a DataFrame from the collected data
df = pd.DataFrame(model_data)
# Sort the DataFrame by the 'id' column
df_sorted = df.sort_values(by='id')
# Display the sorted DataFrame
df_sorted
```

We can now see the list of models in the **id** column, as shown in the excerpt in *Figure 7.3*:



index	id ▲	created	object	owned_by
3	gpt-4	1687882411	model	openai
65	gpt-4-0314	1687882410	model	openai
1	gpt-4-0613	1686588896	model	openai

Figure 7.3: An excerpt from the list of GPT-based models

OpenAI's library is quite full of opportunities and paths to explore!

Keep in mind that OpenAI advances at full speed and regularly replaces some models with cutting-edge models.

We will now start with the magic of using ChatGPT models as assistants.

OpenAI models as assistants

Generative AI and GPT assistants will pervade everyday applications. From a software development perspective, nothing will ever return to the old days. ChatGPT-like models will boost the productivity of everyday software development.

In this section, we will unleash the power of GPTs to use them as assistants to explain OpenAI models and engines.

Go to this link to access ChatGPT Plus: https://chat.openai.com/.

If you don't wish to subscribe, you can try OpenAI's free version: https://openai.com/chatgpt.

ChatGPT Plus offers services such as GPT-3.5, GPT-4, and plugins.

The cutoff date of GPT-4 training might limit its responses since the available data is in the past. Therefore, plugins can come in handy, such as the Bing feature that has been implemented in OpenAI. However, the plugins are continually updated, and some are retired.

ChatGPT Plus models will evolve, and you will grow with them!

You can also go directly to New Bing, which is powered by OpenAI's state-of-the-art models: https://bing.com/?link=new.

You can open the New Bing chatbot in Microsoft Edge while you are searching for information.

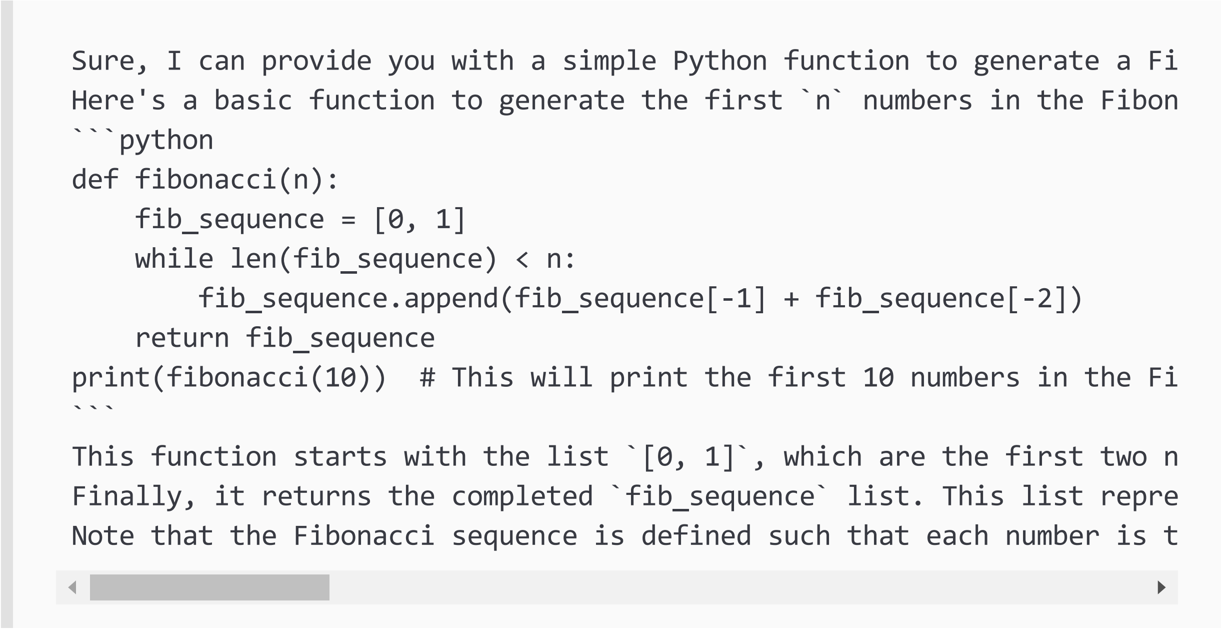
You can also ask ChatGPT to provide source code.

ChatGPT provides source code

ChatGPT can generate source code in multiple programming languages:

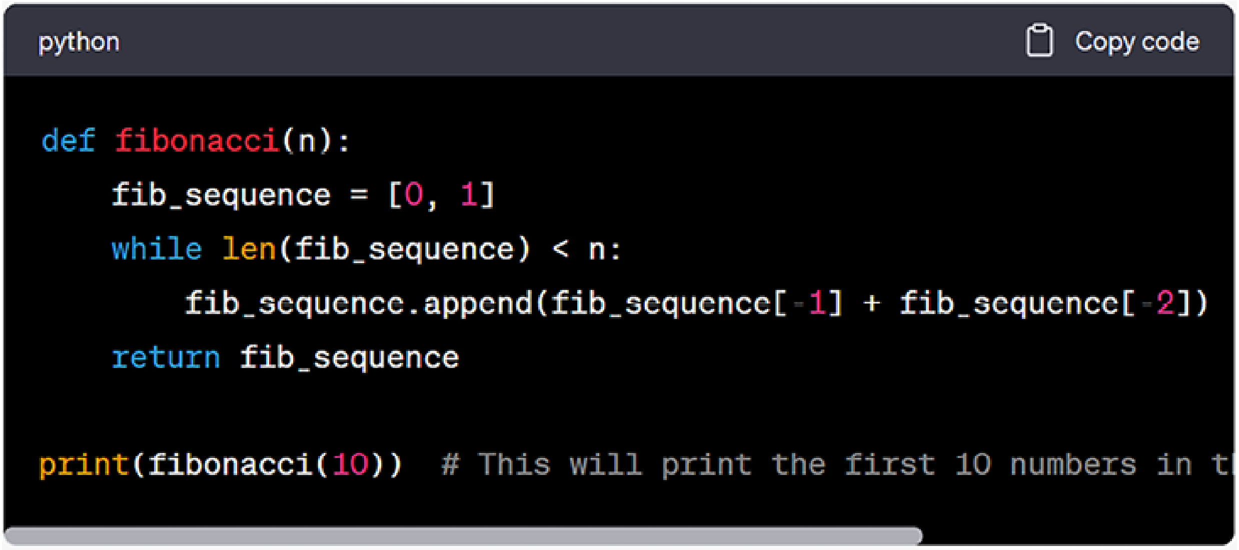
```
Create a Fibonacci sequence in Python
```

ChatGPT will provide the code in Python and explain it:



~~~~
Sure, I can provide you with a simple Python function to generate a Fi
Here's a basic function to generate the first `n` numbers in the Fibon
```python
def fibonacci(n):
    fib_sequence = [0, 1]
    while len(fib_sequence) < n:
        fib_sequence.append(fib_sequence[-1] + fib_sequence[-2])
    return fib_sequence
print(fibonacci(10))  # This will print the first 10 numbers in the Fi
```
This function starts with the list `[0, 1]`, which are the first two n
Finally, it returns the completed `fib_sequence` list. This list repre
Note that the Fibonacci sequence is defined such that each number is t
~~~~

Once you have read the answer, you can copy the code and paste it directly into your Python environment:



```python
def fibonacci(n):
    fib_sequence = [0, 1]
    while len(fib_sequence) < n:
        fib_sequence.append(fib_sequence[-1] + fib_sequence[-2])
    return fib_sequence

print(fibonacci(10))  # This will print the first 10 numbers in t
```

Figure 7.4: Click on "Copy code" in the top-right corner and paste it into your Python editor

You can ask ChatGPT to provide the code in other programming languages.

You can also run GPT-4 as an assistant in your development environment with GitHub Copilot.

GitHub Copilot code assistant

You can use the generative functionality of OpenAI's state-of-the-art models in your development environment. In this section, we will run GitHub Copilot in JetBrains PyCharm.

First, go to GitHub Copilot: https://github.com/features/copilot.

You can read the documentation to get started: https://docs.github.com/en/copilot/getting-started-with-github-copilot.

In this section, JetBrains PyCharm was chosen: https://www.jetbrains.com/products/compare/?product=pycharm&product=pycharm-ce.

JetBrains provides the documentation to install PyCharm and GitHub Copilot. PyCharm contains a function to install GitHub Copilot:

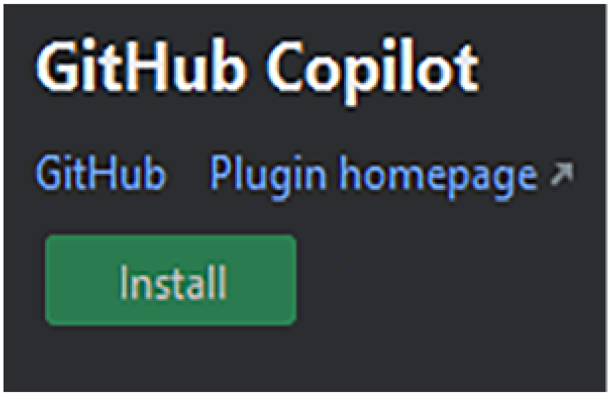
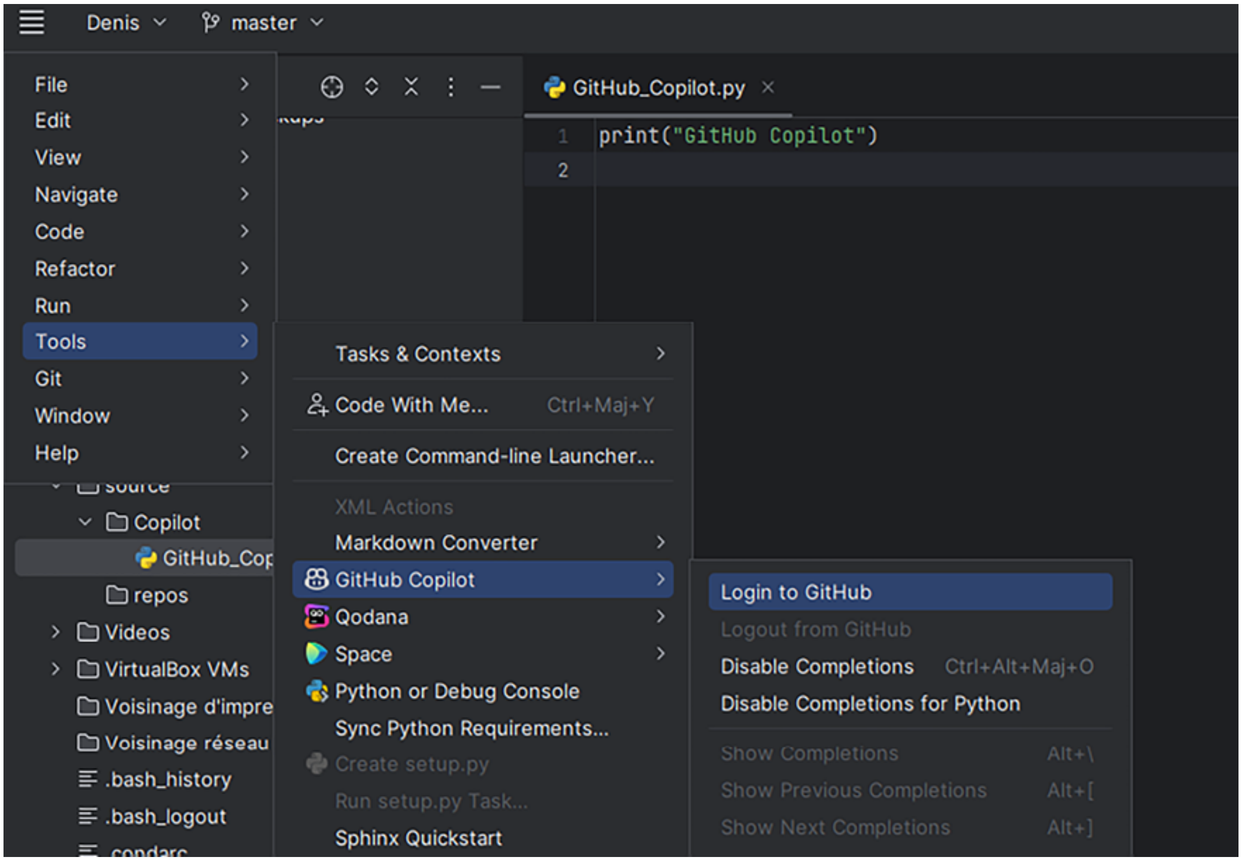


Figure 7.5: One step away from installing GitHub Copilot on PyCharm

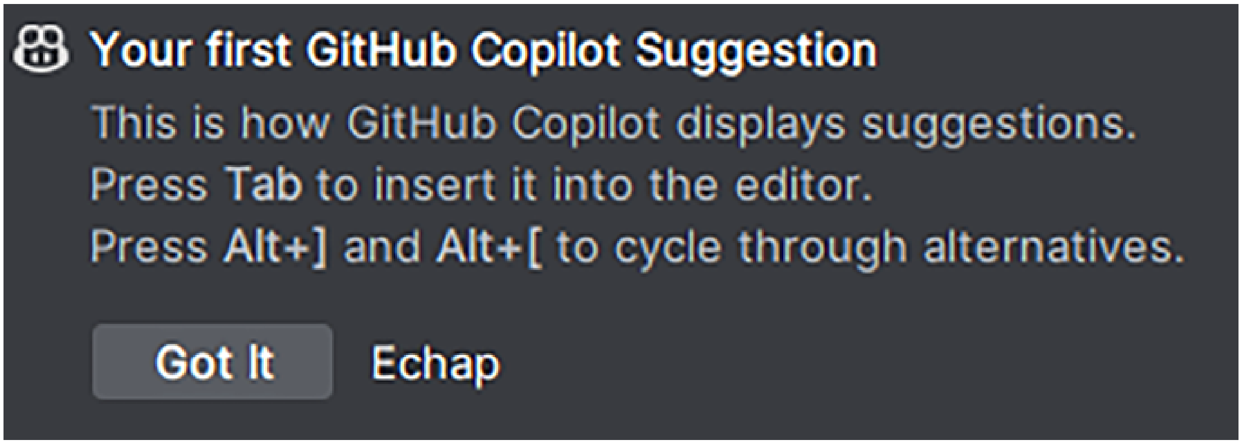
You can activate GitHub in PyCharm and get to work in a few clicks, as shown in *Figure 7.6*:



```
print("GitHub Copilot")
```

Figure 7.6: GitHub Copilot activation path on PyCharm

PyCharm has an intuitive interface and will guide you through activating code generation:



Your first GitHub Copilot Suggestion

This is how GitHub Copilot displays suggestions.
Press Tab to insert it into the editor.
Press Alt+] and Alt+[to cycle through alternatives.

Figure 7.7: The GitHub Copilot plugin walk-through

You can begin by describing a function you would like, such as a Fibonacci sequence, in a comment (the prompt):

```
#create a Fibonacci sequence
```

The code will be generated automatically, and you can select it by pressing *Tab*:

```
def fib(n):
    if n == 0:
        return 0 elif n == 1:
        return 1 else:
        return fib(n-2) + fib(n-1)  # recursive call
```

Then you continue by entering other comments, which act as a prompt, and press *Tab* when the code appears:

```
# calculate the 10th Fibonacci number
fib(10)
#print output
print(fib(10))
```

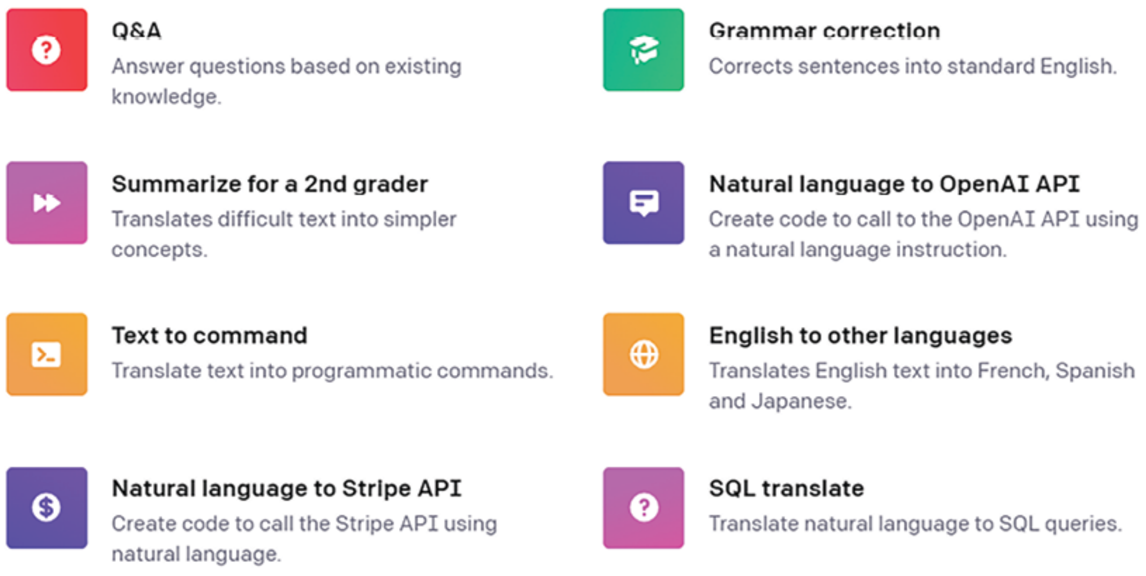
You can then run the code in your environment.

GitHub Copilot, as a coding assistant, will boost your productivity. Just think of the number of assistants we have already gone through! And they only represent the tip of the iceberg of the number of assistants propagating on the market.

General-purpose examples will take us to another NLP task and coding assistant.

General-purpose prompt examples

Go to https://platform.openai.com/examples. The page shows many examples we can look into, as shown in *Figure 7.8*:



Q&A
Answer questions based on existing knowledge.

Grammar correction
Corrects sentences into standard English.

Summarize for a 2nd grader
Translates difficult text into simpler concepts.

Natural language to OpenAI API
Create code to call to the OpenAI API using a natural language instruction.

Text to command
Translate text into programmatic commands.

English to other languages
Translates English text into French, Spanish and Japanese.

Natural language to Stripe API
Create code to call the Stripe API using natural language.

SQL translate
Translate natural language to SQL queries.

Figure 7.8: A few examples from the prompt examples provided by OpenAI

Note that OpenAI GPT models were not pretrained to perform these tasks. The system's magic is that GPT was not trained for these tasks.

If we click on **Grammar correction**, we can access an example in Python:

```
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
response = openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[],
  temperature=0,
  max_tokens=256
)
```

We can also go to the OpenAI Playground to run an example such as Q&A, ask questions, and modify the parameters: https://platform.openai.com/playground/.

You can run many NLP tasks, modify the parameters, and view the code.

ChatGPT for Microsoft Office 365 Copilot opens the door of generative transformer technology to millions of end users, including us.

Getting started with ChatGPT - GPT-4 as an assistant

In this section, we will get started using ChatGPT Plus with GPT-4 as an assistant and copilot. You will see how a cutting-edge developer can reduce the time to market with GPT-4 as a copilot.

Most of this section was written in the notebook by the author but also with the support of GPT-4. GPT-4's comments are preceded by `GPT-4:`, which is a standard ethical procedure.

First, open `Open_AI_GPT_4_Assistant.ipynb` in the chapter directory of the GitHub repository.

This section follows the structure and comments of the notebook. Working at this speed and comfort is an exhilarating experience.

1. GPT-4 helps to explain how to write source code

Give GPT-4 instructions with well-crafted prompts to steer the model.

It will return the source code. This notebook was designed with Python but you can try other languages.

In the top-right corner of the source code frame, click on **Copy code**:

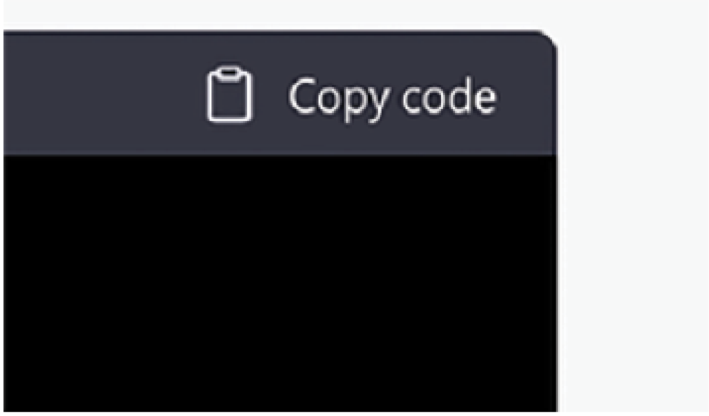


Figure 7.9: Copy code option

Paste the code in a code cell in your notebook and run it! Note that there is a limitation: some debugging might be necessary. In any case, the status of each test is provided at the end of each section.

2. GPT-4 creates a function to show the YouTube presentation of GPT-4 by Greg Brockman on March 14, 2023

`Denis Rothman: I would like to write a program in Python in Google Colab to display a YouTube video in a frame in Google Colab cell. The video is a presentation of GPT-4 by Greg Brockman. How I can write this code?`

`GPT-4: To display a YouTube video in a Google Colab notebook…`

Read the rest of GPT -4's instructions in the notebook. We give GPT-4 the instructions for the task we wish to obtain. GPT-4 responds with the instructions for us to follow:

```
from IPython.display import YouTubeVideo
Greg_Brockman="hdhZwyf24mE"
# Replace the video_id below with the YouTube video ID of the desired
video_id = Greg_Brockman  # Replace with the correct video ID of Greg
YouTubeVideo(video_id)
```

3. GPT-4 creates an application for WikiArt to display images

The developer provides the prompt, and GPT-4 responds with the instructions:

`Denis Rothman: I would like to write a Python program for Google Colab that can display images from wikiart. How do I start?`

`GPT-4:To display images from Wikiart in a Google Colab notebook using Python, you can follow these steps…`

Read the rest of GPT-4's instructions in the notebook and run the code:

```
# Import required libraries
import requests
from IPython.display import Image, display
# Function to display an image from Wikiart
def display_wikiart_image(url):
    response = requests.get(url)
    if response.status_code == 200:
        img = Image(data=response.content)
        display(img)
    else:
        print("Unable to fetch image")
# Replace the URL below with the desired Wikiart image URL
wikiart_image_url = "https://uploads7.wikiart.org/images/salvador-dali
display_wikiart_image(wikiart_image_url)
```

Bear in mind that it takes less than a minute to get each function running.

4. GPT-4 creates an application to display IMDb reviews

We are now settled in our routine and getting used to the comfort of piloting a racing car!

We provide the prompt and obtain instructions:

`Denis Rothman: Ok. I want to create another program on Google Colab in Python. This time I want to write a program that displays movie reviews from IMDB and displays them in the Google Colab notebook. How do I do this?`

`GPT-4: To display movie reviews from IMDb in a Google Colab notebook using Python, you can use the requests library to fetch the HTML …`

Read the rest of GPT-4's instructions in the notebook and run the code.

We first install `beautifulsoup4` to scrape web pages:

```
pip install beautifulsoup4 lxml
```

Then we run the code provided by GPT-4:

```python
import requests
from bs4 import BeautifulSoup
from IPython.display import display, Markdown
def display_imdb_reviews(movie_id, num_reviews=5):
    url = f"https://www.imdb.com/title/{movie_id}/reviews"
    response = requests.get(url)
    if response.status_code != 200:
        print("Unable to fetch IMDb reviews")
        return
    soup = BeautifulSoup(response.text, "lxml")
    reviews = soup.find_all("div", class_="imdb-user-review")
    for idx, review in enumerate(reviews[:num_reviews]):
        title = review.find("a", class_="title").text.strip()
        author = review.find("span", class_="display-name-link").text.
        date = review.find("span", class_="review-date").text.strip()
        content = review.find("div", class_="text").text.strip()
        display(Markdown(f"**Review {idx + 1}: {title}**"))
        display(Markdown(f"_by {author} on {date}_"))
        display(Markdown(f"{content}\n\n---"))
# Replace the movie_id below with the IMDb ID of the desired movie
movie_id = "tt1375666" # Inception (2010)
display_imdb_reviews(movie_id)
```

The output contains the requested reviews:

```
Review 1: A one-of-a-kind mind-blowing masterpiece!
by adrien_ngoc_1701 on 1 March 2019
My 3rd time watching this movie! Yet, it still stunned my mind,…
```

5. GPT-4 creates an application to display a newsfeed

Humanity went from walking on foot to riding horses, from horses to trains, and from trains to cars to airplanes. Soon, humanity will travel on shuttles in space. In the 21st century, few people want to go from New York to Los Angeles on horseback. Some will prefer driving or taking a train. Most people will fly.

Once we get used to the speed a GPT-4 copilot provides, there is no turning back!

For this task, a simple one-sentence prompt does the job:

`Denis Rothman I want to write a program in Python in Google Colab that reads a news feed. How do I do that?`

`GPT-4: To write a Python program in Google Colab that reads a news feed, …`

Read the rest of GPT-4's instructions in the notebook and run the code:

```python
!pip install feedparser
import feedparser
news_feed_url = "http://feeds.bbci.co.uk/news/rss.xml"
feed = feedparser.parse(news_feed_url)
for entry in feed.entries:
    print(entry.title)
    print(entry.link)
    print()
```

The output provides news headlines and websites to consult:

```
NHS 5% pay offer may end bitter dispute in England
https://www.bbc.co.uk/news/health-64977269?at_medium=RSS&at_campaign=KA
../…
```

6. GPT-4 creates a k-means clustering (KMC) algorithm

This example shows an iterative dialogue to obtain the right source code.

First, GPT-4 does not provide the exact answer. We provide the error message, and GPT-4 corrects the code.

You also provide the code you have written directly and share an error you are encountering.

In this case, GPT-4 provided the code, analyzed the error, and regenerated a correct response.

This prompt requests a KMC program:

`Denis Rothman Create a program in Google Colab in Python that can generate 1000 random numbers between 1 and 1000 and then run a k-means clustering algorithm on this data and finally display the result in matplotlib.`

`GPT-4: To create a program in Google Colab that generates 1000 random numbers between 1 and 1000, runs a k-means clustering algorithm on the data, and displays the result using matplotlib, follow these steps:`

Read the rest of GPT-4’s instructions in the notebook and run the code.

However, this time, the GPT-4 code will not work. Don't be surprised when you get an error. Read GPT-4's reaction and run the new, corrected code. It runs perfectly.

You can ask GPT-4 to correct or explain its code or your code. 21st-century productivity in development has reached another level!

This notebook showed the flexibility and copilot potential of GPT-4.

Now, we will look into the OpenAI GPT-4 API.

Getting started with the GPT-4 API

OpenAI has some of the most powerful transformer engines in the world. One GPT-4 model can perform hundreds of tasks. GPT-3 can do many tasks it wasn't trained for.

This section will use the API in `Getting_Started_GPT_4_API.ipynb`.

To use GPT-3, go to OpenAI's website, https://openai.com/, and sign up.

We can run the examples provided by OpenAI to get started. We are once again relying on assistants.

Running our first NLP task with GPT-4

Let's start using GPT-4 in a few steps.

Go to Google Colab and open `Getting_Started_GPT_4_API.ipynb`, which is the chapter directory of the book on GitHub.

You do not need to change the hardware settings of the notebook. We are using an API, so we will not need much local computing power for the tasks in this section.

The steps of this section are the same ones as in the notebook.

Running an NLP task is done in three simple steps:

Steps 1: Installing OpenAI and Step 2: Entering the API key

Steps 1 and 2 are the same as we went through in the *GPT models* section of this chapter.

Let's now run an NLP task.

Step 3: Running an NLP task with GPT-4

We copy and paste an OpenAI example for a **grammar correction** task:

```
from openai import OpenAI
client = OpenAI()
response = client.chat.completions.create(
  model="gpt-4",
  messages=[
    {
      "role": "system",
      "content": "You will be provided with statements, and your task
    },
    {
      "role": "user",
      "content": "She no went to the market."
    }
  ],
  temperature=0,
  max_tokens=256,
  top_p=1,
  frequency_penalty=0,
    presence_penalty=0
)
```

The task is to correct this grammar mistake: `She no went to the market`.

We can process the response as we wish by parsing it. OpenAI's response is a dictionary object. The OpenAI object contains detailed information on the task. We can ask the object to be displayed:

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

The output of "`text`" in the dictionary is the grammatically correct sentence:

```
She didn't go to the market.
```

You can steer the outputs with several parameters.

Key hyperparameters

The term "parameters" generally refers to the trained parameters, such as weights and biases. The term "hyperparameters" refers to parameters that can configure the model (layers, dimensions) or its behavior when it processes a request.

A request begins with a prompt and is completed by the model or engine. "Completion" refers to the model's task of completing a prompt with its Generative AI capabilities. The model predicts a response token by token.

You can try each example in the notebook with the different models or engines listed in the *GPT models* section of this chapter and modify the behavior of the completion task with the following parameters:

- `model="gpt-4"`: The choice of the OpenAI GPT-3 model to use and possibly other models in the future.
- `temperature=0`: A higher value, such as `0.9,` will force the model to take more risks. Do not modify the temperature and `top_p` at the same time.
- `max_tokens=256`: The maximum number of tokens of the response.
- `top_p=1.0`: Top-P or nucleus sampling selects the top probabilities until the sum of the probabilities reaches the `top_p=1.0` value. Then, one of the probabilities in this set is chosen randomly.
- `frequency_penalty=0.0`: A value between `0` and `1` limits the frequency of tokens in a given response.
- `presence_penalty=0.0`: A value between `0` and `1` forces the system to use new tokens and produce new ideas.

Each pipeline on each platform for each model has its hyperparameters that will influence the sampling process and control outputs.

For more on the sampling process involving the temperature Top-P hyperparameters, see the *Vertex AI PaLM 2 interface* section of *Chapter 14, Exploring Cutting-Edge LLMs with Vertex AI and PaLM 2*.

In this case, the prompt has two main parts: the system and user roles.

The system role

The roles are preceded by `"role":`. The system role is defined by:

```
"role": "system",
```

Then, the content of the role is defined by the term `"content":`, which contains the instructions for the model:

```
 "content": "You will be provided with statements, and your task is to
```

The user role

The user role is defined by:

```
"role": "user",
```

Then, as for the role, the content of the user role is defined by the term `"content":`, which contains the user request:

```
"content": "She no went to the market."
```

Many more options are possible. OpenAI documentation is constantly evolving and offering new functionality: https://platform.openai.com/docs/overview.

Your imagination is the limit!

You can now continue the notebook for the other tasks.

Running multiple NLP tasks

`Getting_Started_GPT_4_API.ipynb` contains examples you can run to practice implementing OpenAI GPT-4. You can refer to the description of the content of the prompt in *Step 3, Running an NLP task with GPT-4* section above.

Run these examples in the notebook. We ran the grammar correction and translation task. You can rerun the tasks with different engines and parameters. Then continue exploring and running the remaining examples with the output processing code provided when required:

- Example 1: Grammar correction
- Example 2: English-to-French translation
- Example 3: Calculate time complexity
- Example 4: Movie to emoji
- Example 5: Spreadsheet creator
- Example 6: Advanced tweet classifier
- Example 7: Natural Language to SQL

You can go further and run many other tasks on the Examples page: `https://beta.openai.com/examples`.

You can also take GPT-4 to the next level with **Retrieval Augmented Generation** (**RAG**).

Retrieval Augmented Generation (RAG) with GPT-4

In this section, we will build an introductory program that implements **RAG**. Document retrieval is not new. Knowledge bases have been around since the arrival of queries on databases decades ago. Generative AI isn't new either. RNNs were AI-driven text generators years ago.

Taking these factors into account, we can say that RAG is not an innovation but an improvement that compensates for the lack of precision, training data, and responses of Generative AI models. It can also avoid fine-tuning a model in some instances.

There are also different ways of performing augmented generation, as we will see in the following chapters:

- *Chapter 11, Leveraging LLM Embeddings as an Alternative to Fine-Tuning*, is where we will implement embedded data.
- *Chapter 15, Guarding the Giants: Mitigating Risks in Large Language Models*, in which one of the mitigating solutions is to implement knowledge bases.

OpenAI mentions document retrieval as one of the methods to improve results in their documentation: `https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results`.

In this section, we will implement an improvement of GPT's Generative AI GPT-4 functionality with RAG.

Open `GPT_4_RAG.ipynb` in the directory of this chapter.

The notebook is divided into three parts: *Installation*, *Document retrieval*, and *Retrieval augmented generation*.

Installation

The following packages are installed:

- `tiktoken`, a BPE tokenizer for OpenAI's models
- `cohere`, a text-generation LLM platform
- `openai`, the API functionality we will use for GPT-4
- `ipwidgets` to create HMTL widgets
- `transformers`, packages for transformer functionality
- `requests` to make HTTP requests in Python
- `beautifulsoup4` for web scraping

Once the libraries and modules are imported, we can create a document retrieval function.

Document retrieval

The document retrieval section of the code contains two functions: URL retrieval and URL processing based on the user request.

The first function is a URL retrieval function based on the user request:

```
def select_urls_based_on_query(user_query):
```

The URLs are selected by using a keyword in the user's request. Any other method can be implemented. The following code selects URLs in a knowledge base based on keywords detected in the user's request:

```
def select_urls_based_on_query(user_query):
    # URLs related to 'climate'
    climate_urls = [
        "https://en.wikipedia.org/wiki/Climate_change",  # Replace wit
"https://en.wikipedia.org/wiki/Effects_of_climate_change"
    ]
    # URLs related to 'RAG'
    rag_urls = [
        "https://en.wikipedia.org/wiki/Large_language_model",  # Repla
"https://huggingface.co/blog/ray-rag"
    ]
    # Check if 'climate' is in the user query
if "climate" in user_query.lower():
        return climate_urls
    # Check if 'RAG' is in the user query
elif "RAG" in user_query:
        return rag_urls
    # Default return if no keyword matches
return []
```

You can use any other method you wish, such as database queries and lists. You can also implement any form of automation you want. *The main concept to keep in mind is that the search should be based on the user's request and adapted to it.*

The second function scrapes and summarizes the documents.

First, a `summarizer` is defined:

```
def fetch_and_summarize(user_query):
    urls = select_urls_based_on_query(user_query)
    summarizer = pipeline("summarization", model="sshleifer/distilbart
```

The content retrieved is processed:

```
    summaries = []
    for url in urls:
        page = requests.get(url)
        soup = BeautifulSoup(page.content, 'html.parser')
        # Try to extract the main article text more accurately
# This is a generic example and might need to be adjusted for specific
        article = soup.find('article')
        if article:
            article_text = article.get_text()
        else:
            paragraphs = soup.find_all('p')
            article_text = ' '.join([para.get_text() for para in parag
        # Truncate if too long for the model
if len(article_text) > 1024:
            article_text = article_text[:1024]
```

The `summarizer` is then activated to summarize the content retrieved:

```
   summary = summarizer(article_text, max_length=130, min_length=30, d
        summaries.append(summary)
```

Finally, the summary is returned to the GPT-4 function:

```
 return summaries
```

We have seen one of the many ways to retrieve data based on a user’s request. Now, let’s see how to integrate the information in a GPT-4 input.

Augmented retrieval generation

We first import the modules and classes we need:

```
from openai import OpenAI
import ipywidgets as widgets
from IPython.display import display
```

Then, we define the client and a model:

```
client = OpenAI()
AImodel = "gpt-4" # or select another model
```

We create the API function:

```
# Function to interact with OpenAI's model
def openai_chat(input_text, document_excerpt, web_article_summary):
    # Start the OpenAI API call to generate a chat response
    response = client.chat.completions.create(
        model=AImodel,
```

The prompt message contains three roles:

- A document excerpt, which can come from any source (database, web, text file, JSON, or hard-coded):

  ```
  "role": "system",  # "system" role for providing contextual
  "content": f"The following is an excerpt from a document about cl
  # The document excerpt is now a variable passed to the function
  ```

- A web excerpt, which comes from the summarizing function we defined:

```
 "role": "system",  # Another "system" role message
"content": f"The following is a summary of a web article on renew
# The web article summary is now a variable passed to the functio
```

- The user’s input text:

```
 "role": "user",  # "user" role for the actual user query
"content": input_text
      # The user's query or input that the model will respond to
```

We now define the parameters as we explored in the *Getting started with the GPT-4 API* section of this chapter:

```
        temperature=0.1,  # Controls randomness. Lower values make res
        max_tokens=150,   # Sets the maximum length of the response in
        top_p=0.9,        # Nucleus sampling: A higher value increases
        frequency_penalty=0.5,  # Reduces repetition of the same text.
        presence_penalty=0.5 # Reduces repetition of similar topics. H
```

The response is not a dictionary but an instance of a class:

```
# the response object is not a dictionary. It is an instance of the Ch
# to access the content property, use dot notation instead of bracket
return response.choices[0].message.content
```

Note that the code excerpts do not contain brackets. Consult the notebook for the complete code.

We can now define the user request:

```
input_text = "What are the impacts of climate change?"
#input_text = "What is RAG"
```

In this case, the input text is in the code. When you deploy an application, the user request will be submitted from the interface you have designed.

The example here is climate change, or it could be RAG. To go further, you will have to enhance the document retrieval functionality. Keep in mind that this requires resources for design, development, copyright management, and all the standard procedures that go with document retrieval and usage.

We now parse the user request to decide what type of document excerpt we need to use. This introductory educational example uses keywords. More sophisticated parsers can be implemented. Also, the document is hard-coded in this example. You could retrieve and summarize a PDF, insert a complete text, or any other content you wish to implement. In this case, we are focusing on climate change but leave it up to the code to choose a document excerpt:

```
# 1. you can create a function specifically for your domain with diffe
# Check if 'climate' is in the user query
if "climate" in input_text.lower():
        document_excerpt = "Climate change refers to significant chang
# Check if 'RAG' is in the user query
if "RAG" in input_text.lower():
        document_excerpt = "OpenAI documentation states that RAG or re
```

You can hard-code the information if it is for customer support on a product that will be stable for its lifecycle, or you can automate the process by activating the scraping and summarizing functionality we implemented:

```
#2. and/or you can automate the retrieval
summaries = fetch_and_summarize(input_text)
#print(summaries)
web_article_summary = summaries
```

We are now ready to send the document excerpt and summary to OpenAI along with the user request. GPT-4 will have a significantly enhanced context to respond:

```
iresponse = openai_chat(input_text, document_excerpt, web_article_summ
formatted_response = iresponse.replace('\n', '<br>')  # Replace \n wit
display(widgets.HTML(value=formatted_response))  # Display response as
```

The response and output are interesting:

```
Climate change impacts the physical environment, ecosystems, and human
```

We can see that RAG can be quite a way to improve the quality of Generative AI models and also control the outputs through guided enhanced input requests.

Summary

We began the chapter by seeing how OpenAI **Generative Pre-trained Transformer** (**GPT**) models are **General Purpose**

Technologies (**GPTs**). As such, they improve rapidly, and their diffusion is highly pervasive. The Generative AI functionality of OpenAI's models has broadened the horizons for mainstream applications.

We continued by examining the improvements in the architecture of GPTs through decoder stacks, scaling, and machine power. These improvements led to the creation of ChatGPT, which spread into mainstream everyday lives, pervading applications such as search engines, office tools, and more.

We started with some of the many generative transformer assistants, including ChatGPT, New Bing, GitHub Copilot, Microsoft 365 Office Copilot, and OpenAI's Playground.

Our journey then led to building several examples with the OpenAI GPT-4 API, such as grammar corrections, translations, and more.

Finally, we built an example of how RAG can improve the performances of Generative AI models, such as GPT-4.

In the next chapter, *Chapter 8*, *Fine-Tuning OpenAI GPT Models*, we will fine-tune an OpenAI GPT model.

Questions

1. A zero-shot method trains the parameters once. (True/False)
2. Gradient updates are performed when running zero-shot models. (True/False)
3. GPT models only have a decoder stack. (True/False)
4. OpenAI GPT models are not GPTs. (True/False)

5. The diffusion of generative transformer models is very slow in everyday applications. (True/False)
6. GPT-3 models have been useless since GPT-4 was made public. (True/False)
7. ChatGPT models are not completion models. (True/False)
8. Gradio is a transformer model. (True/False)
9. Supercomputers with 285,000 CPUs do not exist. (True/False)
10. Supercomputers with thousands of GPUs are game changers in AI. (True/False)

References

- OpenAI and GPT-3 engines: https://beta.openai.com/docs/engines/engines
- BertViz GitHub Repository by *Jesse Vig*: https://github.com/jessevig/bertviz
- OpenAI's supercomputer: https://blogs.microsoft.com/ai/openai-azure-supercomputer/
- *Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever,* 2018, *Improving Language Understanding by Generative Pre-Training*: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
- *Alec Radford et al.*, 2019, *Language Models are Unsupervised Multitask Learners*: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

- Common Crawl data: https://commoncrawl.org/overview/
- GPT-4 Technical Report, OpenAI 2023: https://arxiv.org/pdf/2303.08774.pdf

Further reading

- *Alex Wang et.al*, 2019, *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*: https://arxiv.org/pdf/1804.07461.pdf
- *Alex Wang et al.*, 2019[2], *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems*: https://w4ngatang.github.io/static/papers/superglue.pdf
- *Tom B. Brown et al.*, 2020, *Language Models are Few-Shot Learners*: https://arxiv.org/abs/2005.14165
- *Chi Wang et al.*, 2023, *Cost-Effective Hyperparameter Optimization for Large Language Model Generation Inference*: https://arxiv.org/abs/2303.04673
- *Vaswani et al.*, 2017, *Attention Is All You Need*: https://arxiv.org/abs/1706.03762

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

https://www.packt.link/Transformers