

17

The Consolidation of Suprahuman Transformers with OpenAI's ChatGPT and GPT-4

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. In March 2023, OpenAI released its latest APIs for Whisper and GPT-4, which created another media storm. Also, in March 2023, code generation took a new turn when OpenAI went from Codex to GPT-3.5-turbo and GPT-4.

AI history is on a roll!

However, those who are reading this book, far from the AI frenzy, understood that GPT-3.5 (ChatGPT) is the logical evolution of GPT-3. The determined reader of this book knows the potential of prompt engineering, generating completions, and multimodal vision transformers.

This chapter continues on 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 provides enough transformer model expertise to enjoy the ride through the cutting-edge models in this chapter.

For the determined reader of this book, this chapter will be an experience of exploration and discovery. The understanding and skills acquired up to this chapter will lead you to the very heart of artificial intelligence research in the field of transformer model technology.

We will focus on the novelties and not the functionality experience with which the reader has already attained proficiency.

We will see how OpenAI succeeded in consolidating the performance of GPT-3 with that of GPT-3.5 legacy (ChatGPT), GPT-3.5-turbo, GPT-4, and DALL-E.

Then we will get started with looking at these cutting-edge models' Python notebooks to jump-start the ChatGPT API, ask ChatGPT Plus to generate code, get started with GPT-4, design advanced prompts, create images with the DALL-E 2 API, add audio to conversational AI, and run a Whisper function.

This chapter covers the following topics:

- Getting started with the ChatGPT API
- ChatGPT and ChatGPT Plus as a copilot
- Asking ChatGPT to write and comment on a classification program
- Running GPT-3.5 legacy (ChatGPT)
- GPT-3.5-turbo default (faster version)
- Getting started with the GPT-4 API
- Advanced prompt engineering
- Running OpenAI's moderation transformer
- Adding audio to ChatGPT dialogues
- Becoming an AI artist with the DALL-E 2 API

Let's begin our journey by exploring the most recent suprahuman transformer models.

Consolidating suprahuman NLP with ChatGPT and GPT-4 transformer models

This section first describes two important aspects of the transformer environment we will be looking into:

- How to get the most out of this chapter
- Opportunities

We will start by seeing how to get the most out of this chapter.

How to get the most out of this chapter

The previous chapters covered the key aspects of transformer models. Exploring OpenAI's recent models, such as ChatGPT and GPT-4, will be a step forward, not a gap to fill.

Build on your knowledge

We have covered the key aspects of transformer models from the beginning of this book up to this chapter. Thus, there is no need to start from scratch for each of the notebooks we will go through.

Focus on the innovations

The reminder icon will make the sections more concise so you can focus on the innovations. Build on the knowledge and expertise acquired in this book and enjoy the ride!

GitHub repository

Every notebook described in this chapter is in the Chapter17 directory of the GitHub repository of the book:

<https://github.com/Denis2054/Transformers-for-NLP-2nd-Edition>

In each section, when a notebook name is provided, you will know that it is in the Chapter17 directory of the repository.

For example:

Open `Jump_Starting_ChatGPT_with_the_OpenAI_API.ipynb`.

The notebooks were developed using the free version of Google Colab but can naturally run in other environments (Kaggle, Gradient, Sagemaker, etc.).

Some sections refer to notebooks that are in the Bonus directory of the GitHub repository.

Accepting the limitations of a disruptive innovation

Exploring the cutting edge means reaching the limits of existing technology. OpenAI models are no exception. Accepting the limitations and finding ways to circumvent them is part of discovering new horizons.

Let's list some of the limitations we will face:

- OpenAI clearly expresses this when we connect to ChatGPT: sometimes the system will generate erroneous, offensive, and biased outputs.

- The cutoff dates can sometimes produce erroneous information. For example, ChatGPT was launched in November 2022 and uses data no later than 2021.
- OpenAI models progress rapidly. Model upgrades and deprecation are frequent.
- You may or may not have access to ChatGPT Plus. ChatGPT Plus runs with GPT-3.5 legacy (original ChatGPT), GPT-3.5 default, and GPT-4. However, access might be limited because of OpenAI scaling issues or selective waitlists.
- Cost. You may or may not wish to pay for access. Even if you do, be careful! There is a maximum budget alert in your OpenAI account. Activate it and watch your budget.

You can read the notebooks to discover and explore ChatGPT until you obtain access to ChatGPT Plus and the APIs to run the notebooks. You acquired sufficient theoretical and practice expertise in the previous chapters to understand the notebooks in this chapter.

The potential of AI-driven content generation is unlimited. However, before we continue, we need to understand the copyright issues.

Who owns AI-created content?

Who owns AI-generated content of any type: art, code, language, and general-purpose completions? This is a billion-dollar litigation issue! The following three articles highlight the still-murky AI-generated content copyright issues.

In December 2022, Forbes published “*Who Ultimately Owns Content Generated By ChatGPT And Other AI Platforms?*” by Joe McKendrick, <https://www.forbes.com/sites/joemckendrick/2022/12/21/who-ultimately-owns-content-generated-by-chatgpt-and-other-ai-platforms/>.

In February 2023, Reuters published “*AI-created images lose U.S. copyrights in test for new technology*” by Blake Brittain: <https://www.reuters.com/legal/ai-created-images-lose-us-copyrights-test-new-technology-2023-02-22/>.

In early April 2023, Reuters published “*Humans vs. machines: the fight to copyright AI art*” by Tom Hals and Blake Brittain: <https://www.reuters.com/default/humans-vs-machines-fight-copyright-ai-art-2023-04-01/>.

Go through the articles and have a think. We will see how this turns out.

In the meantime, here are some recommendations when using generative AI:

- Author your content for publications. You can use AI as a helper just as you would a search engine or a dictionary to verify what you are writing. For example, Denis Rothman created the text and wrote this chapter.
- If you use ChatGPT completion in your work, make sure to cite ChatGPT. For example, the notebooks GitHub repository of this book cites AI-generated completions. Also, the notebooks are open-sourced under the MIT License.
- Before publishing AI-generated content, seek legal advice. Make sure to verify the laws and regulations of the locations that will access your applications.

With these pending issues in mind, let's examine the opportunities.

Opportunities

The limitations should not prevent you from grasping the opportunities that await you in your AI projects by using OpenAI's latest models.

Let's go through some of the opportunities:

- GPT-3.5 legacy (the original ChatGPT), GPT-3.5 default (faster than the legacy version), and GPT-4 will take you to the next level of AI. You can implement these transformer models in a wide range of NLP problems you need to solve.
- Those who know how to increase productivity with GPT-4 models will deliver company goals/work faster than their competition.
- In an article submitted on March 23, 2023, OpenAI examined the impact of **LLMs (Large Language Models)** such as GPT-4, and states the following:

"Our analysis suggests that, with access to an LLM, about 15% of all worker tasks in the US could be completed significantly faster at the same level of quality."

Reference: <https://openai.com/research/gpts-are-gpts>

Generative Pre-trained Transformer (GPT) models are **General-Purpose Technologies**. Transformers can perform tasks that go beyond NLP and visual/audio applications. Domains beyond NLP can benefit from GPT models, as discussed in the *General-purpose sequences* and *Embedding2ML* sections in the previous chapter.

Now, let's go through the models we will implement in this chapter.

List of models in this chapter

In this chapter, we will focus on the following models as described by OpenAI, <https://platform.openai.com/docs/models/>:

- GPT-4 limited beta: A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code
- GPT-3.5: A set of models that improve on GPT-3 and can understand as well as generate natural language or code
- DALL-E beta: A model that can generate and edit images given a natural language prompt
- Whisper beta: A model that can convert audio into text: <https://openai.com/blog/introducing-chatgpt-and-whisper-apis>
- Moderation: A fine-tuned model that can detect whether text may be sensitive or unsafe
- GPT-3: A set of models that can understand and generate natural language

The list of models seems quite limited. But if you consider the variations of these models, you will see that there are many possible engines (variations of a model).

There is a multitude of subsets of OpenAI’s main models, as shown in the following list:

0 babbage	18 ada-code-search-code	36 text-search-curie-query-001
1 davinci	19 ada-similarity	37 text-search-babbage-doc-001
2 text-davinci-edit-001	20 text-davinci-003	38 gpt-3.5-turbo
3 babbage-code-search-code	21 code-search-ada-text-001	39 curie-search-document
4 text-similarity-babbage-001	22 text-search-ada-query-001	40 text-search-curie-doc-001
5 code-davinci-edit-001	23 davinci-search-document	41 babbage-search-query
6 text-davinci-001	24 ada-code-search-text	42 text-babbage-001
7 ada	25 text-search-ada-doc-001	43 gpt-4
8 curie-instruct-beta	26 davinci-instruct-beta	44 text-search-davinci-doc-001
9 babbage-code-search-text	27 text-similarity-curie-001	45 gpt-4-0314
10 babbage-similarity	28 code-search-ada-code-001	46 text-search-babbage-query-001

11 whisper-1	29 ada-search-query	47 curie-similarity
12 code-search-babbage-text-001	30 text-search-davinci-query-001	48 curie
13 text-curie-001	31 curie-search-query	49 text-similarity-davinci-001
14 code-search-babbage-code-001	32 gpt-3.5-turbo-0301	50 text-davinci-002
15 text-ada-001	33 davinci-search-query	51 davinci-similarity
16 text-embedding-ada-002	34 babbage-search-document	
17 text-similarity-ada-001	35 ada-search-document	

Table 17.1: A list of OpenAI's engines

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.

OpenAI has announced that some models will soon be abandoned:

- gpt-3.5-turbo-0301 on June 1st, 2023
- gpt-4-0314 on June 14th, 2023
- gpt-4-32k-0314 on June 14th, 2023

You can investigate OpenAI models with the `Exploring_GPT_4_API.ipynb` notebook in the Bonus directory of the GitHub repository. It contains a toolkit that:

- Lists the engines as in the table above.
- Compares GPT-4, ChatGPT-3.5-turbo, and GPT-3 models when performing the same task. You will get a few interesting surprises when running the comparisons.
- Runs a variety of comparison tasks: dialogues, math problems, instruct series, movie to emoji, general-knowledge questions, unsafe content, undetected negative Tweets, failed translation, and search engine prompt optimized by specifying the audience of the topic requested.
- Provides a link to OpenAI's comparison tool.

This investigation will take you deep into OpenAI's research. You can run it now or wait until you have gone through a few sections of this chapter before examining OpenAI's models.

We are now ready to dive into the state-of-the-art world of transformer models.

Let's begin by jump-starting the ChatGPT API.

Jump-starting the ChatGPT API

The original ChatGPT is a GPT-3.5 model that includes GPT-3.5-turbo. These models are an evolution of the GPT-3 model. You are just a step away from home. But what a step!

Open `Jump_Starting_ChatGPT_with_the_OpenAI_API.ipynb`.

Install and import OpenAI and enter the API key.

This is the same process applied in *Chapter 7, The Rise of Suprahuman Transformers with GPT-3 Engines*. Review this chapter if necessary.

Content generation

OpenAI has introduced an innovative request approach:

1. The request is formulated with a **ChatCompletion** call.
2. The model called is `gpt-3.5-turbo`, a powerful conversational AI better known to the public as ChatGPT.
3. Each object of a message has a role: system, assistant, and user. The system role can be a general message, the assistant can provide factual information for the model, and the user role contains the core of the request. Once our prompt is designed, we run it:

```
response=openai.ChatCompletion.create(  
    model="gpt-3.5-turbo",  
    messages=[  
        {"role": "system", "content": "You are a helpful  
assistant."}, #setting the behaviour of the assistant  
        {"role": "user", "content": "What web services do you  
offer?"}, #user or developer instruction  
        {"role": "assistant", "content": "We provide web  
designers, developers and web templates."}, #assistant stores prior  
messages you can insert  
        {"role": "user", "content": "Do you have a starter  
package?"}]) #user follow-up question or suggestion by the developer
```


Response

We can have a look at the complete response:

```
#the complete response
response
```

The complete response contains response text and multiple outputs you can extract:

```
<OpenAIObject chat.completion id=chatcml-6p066dwZx05lwtRNJhJaFeWwo1pAR at
0x7fa9ee4cfb80> JSON: {
  "choices": [
    {
      "finish_reason": null,
      "index": 0,
      "message": {
        "content": "I'm sorry, but I'm just an AI language model and I
don't provide web services directly. If you're looking for a web design
and development package, there are several options available depending
on your needs, budget, and preferences. Many web design and development
companies offer starter packages that provide basic website features
and functionalities, such as a few pages, limited design options, and
basic SEO optimization. It's best to research different companies and
compare their packages and prices to choose the one that meets your
requirements.",
        "role": "assistant"
      }
    }
  ],
  "created": 1677705378,
  "id": "chatcml-6p066dwZx05lwtRNJhJaFeWwo1pAR",
  "model": "gpt-3.5-turbo-0301",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 104,
    "prompt_tokens": 52,
    "total_tokens": 156
  }
}
```

For more information on the objects in the response, see <https://platform.openai.com/docs/guides/chat/introduction>.

One of the main outputs you want to watch for is the number of tokens to monitor the cost of your requests! The best practice is to extract the information from the response and have a classical software function to monitor the cost of your application.

For more on pricing, see <https://openai.com/pricing#language-models>.

Now, let's extract the text of the response.

Extracting the text of the response

The text of the response is extracted from the json object in one line:

```
#extracting the generative AI response  
response["choices"][0]["message"]["content"]
```

The output is some well-constructed generated text:

```
I'm sorry, but I'm just an AI language model and I don't provide web  
services directly. If you're looking for a web design and development  
package, there are several options available depending on your needs,  
budget, and preferences. Many web design and development companies offer  
starter packages that provide basic website features and functionalities,  
such as a few pages, limited design options, and basic SEO optimization.  
It's best to research different companies and compare their packages and  
prices to choose the one that meets your requirements.
```

That is it! You have jump-started the OpenAI ChatGPT API. You can extend this function and add whatever you wish.



Prompt engineering has become the key to obtaining good responses, as we have seen throughout the book. You can build on your knowledge by going through the chapters and finding examples you can explore with ChatGPT.

We will now ask ChatGPT Plus to write and explain a program.

ChatGPT Plus writes and comments on a program

In this section, ChatGPT Plus will do all the work: writing the code, commenting on the code, and providing an explanation.

IBM SPSS Decision Trees is a classification and decision tree tool designed for a decision-making system: <https://www.ibm.com/products/spss-decision-trees>.

However, for some projects, we do not need a complex program but a compact function to get the job done.

Open the following notebook, Chapter17, which is in the GitHub repository:

ChatGPT_Plus_writes_and_explains_classification.ipynb

Install and import OpenAI and enter the API key before running the notebook.

Designing the prompt

After installing scikit-learn as suggested by ChatGPT Plus, we submit two requests in sequence to ChatGPT Plus:

1. Provide a scikit-learn classification of the Iris dataset with some kind of matplotlib graph to describe the result. Don't use OpenAI APIs.
2. Now write a detailed explanation for a beginner in Python for this code.

Now, ChatGPT Plus has written the explanations and the code!

ChatGPT Plus introduces the program:

```
"This program is a simple machine learning project that demonstrates how to use Scikitlearn, a popular machine learning library in Python, to build a decision tree classifier for the Iris dataset. The Iris dataset is a famous dataset in the machine learning community and is often used as a benchmark for classification algorithms.
```

```
In this program, you will first load the Iris dataset and split it into training and testing sets. Then, you will create a decision tree classifier model using Scikit-learn's DecisionTreeClassifier class and train it on the training data. Next, you will evaluate the accuracy of the model on the testing set and compute the confusion matrix, which is a table that summarizes the performance of the model. Finally, you will plot the confusion matrix using Matplotlib..."
```

After installing scikit-learn following the instructions provided by ChatGPT Plus, ChatGPT writes the program.

ChatGPT Plus writes the program

We can just sit back and watch:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.tree import plot_tree

# Load the Iris dataset
iris = load_iris()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.
target, test_size=0.2, random_state=42)

# Train a decision tree classifier model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Evaluate the model on the testing set
score = model.score(X_test, y_test)
print(f"Accuracy: {score:.2f}")

# Compute the confusion matrix
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
```

```

# Plot the confusion matrix
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation="nearest", cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=iris.target_names, yticklabels=iris.target_names,
       title="Confusion matrix",
       xlabel="Predicted label",
       ylabel="True label")
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_
mode="anchor")
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], "d"), ha="center", va="center",
                color="white" if cm[i, j] > cm.max() / 2 else "black")
fig.tight_layout()

# Plot the decision tree
plt.figure(figsize=(20,10))
plot_tree(model, feature_names=iris.feature_names, class_names=iris.
target_names, filled=True)
plt.show()

```

All you need to do is copy the code in ChatGPT:

python

 Copy code

```

import pandas as pd
import numpy as np

```

Figure 17.1: ChatGPT creating code

Then paste and run it in your notebook.

ChatGPT Plus plots the output

ChatGPT plots a confusion matrix:

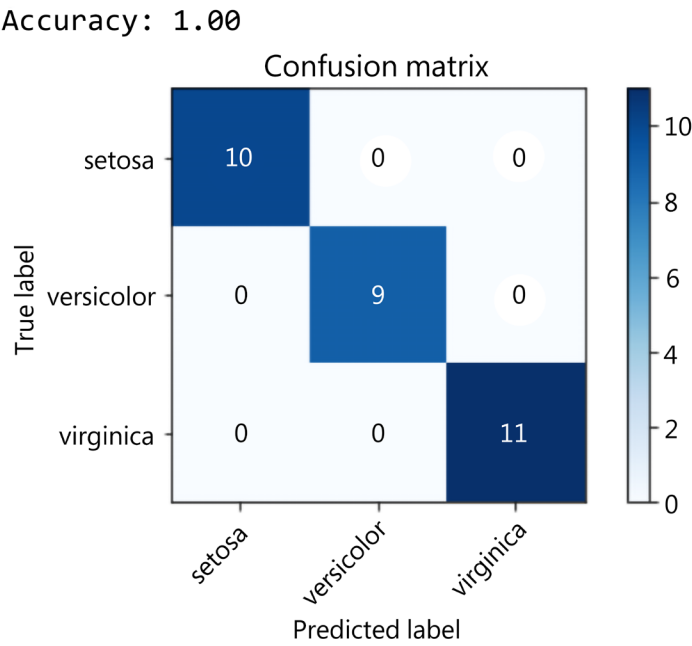


Figure 17.2: A confusion matrix created by ChatGPT

ChatGPT also plots a decision tree:

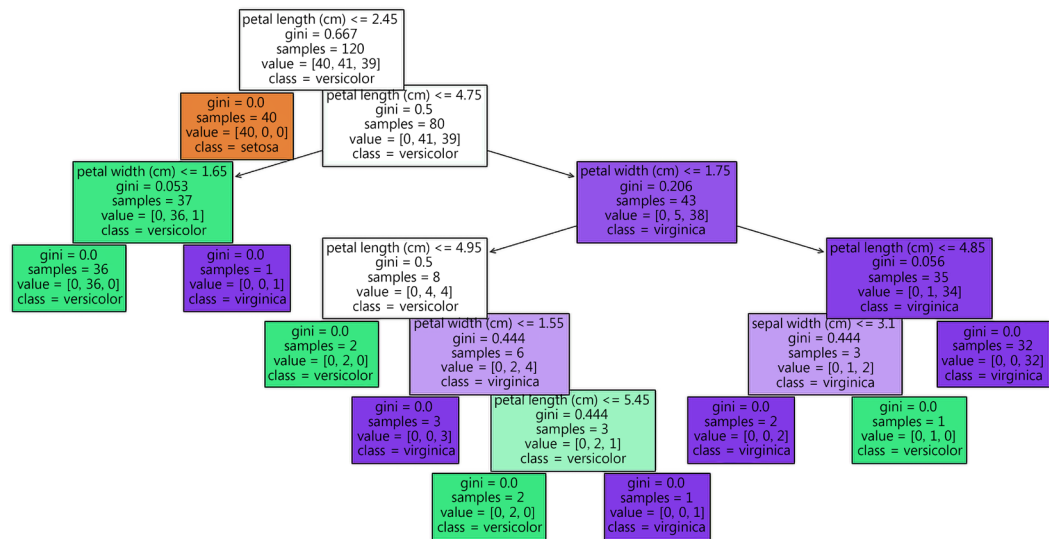


Figure 17.3: ChatGPT creates code to plot a decision tree

It takes less than 10 minutes to get the job done. Even if we develop quickly, using ChatGPT as a copilot is a real time-saver!

Now, let's get started with GPT-4.

Getting started with the GPT-4 API

In this section, we will get started with the GPT-4 API using ChatGPT Plus GPT-4 as a 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 `Getting_Started_with_GPT_4.ipynb`.

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**:

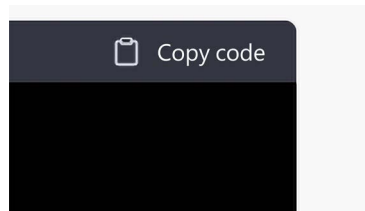


Figure 17.4: Copy code option

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 14th, 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
video_id = Greg_Brockman # Replace with the correct video ID of Greg
Brockman's GPT-4 presentation

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/
the-persistence-of-memory-1931.jpg"
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:

```
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.strip()
```

```

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 news feed

Humanity went from walking on foot to riding horses and from horses to trains, and from trains to cars to airplanes. Soon, humanity will travel with 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:

```

!pip install feedparser

import feedparser

news_feed_url = "http://feeds.bbc.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=KARANGA

../...
```

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 k-means clustering 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!

7. Talking with GPT-4 about GPT-4 and GPT model architecture

This notebook was written on March 16th, 2023, 48 hours after Greg Brockman, Chairman of OpenAI, announced the release of GPT-4.

It was interesting to interview GPT-4.

The interview began with the following interaction:

```
Denis Rothman: * What is the difference between GPT-3.5 turbo and GPT-4?
GPT-4:As an AI language model, I am based on GPT-4 architecture, so I can
provide you with a comparison between GPT-3.5 Turbo and GPT-4. GPT-3.5
Turbo is a version of the GPT-3 architecture. It was developed by OpenAI
and is an improved, more efficient version of GPT-3.
../...
The improvements in GPT-4 include:
1. Higher capacity: GPT-4 has a larger number of parameters than GPT-3.5
Turbo, enabling it to store more knowledge and generate more sophisticated
responses.
2. Better performance: GPT-4 can understand and generate text more
effectively, resulting in improved performance across a range of tasks,
including translation, summarization, question-answering, and more.
3. Enhanced fine-tuning capabilities: GPT-4 allows for better fine-tuning,
making it more customizable for specific applications, industries, or use
cases.
4. Newer knowledge base: Since GPT-4 was trained on more recent data, it
has access to information and events that occurred after GPT-3.5 Turbo's
training data was collected. However, it's important to note that GPT-
4 may be more resource-intensive and potentially more expensive to use,
depending on the specific application and API usage. In summary, GPT-4 is
a more advanced and capable version of the GPT series compared to GPT-
3.5 Turbo, with improvements in capacity, performance, fine-tuning, and
knowledge base."
```

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

Let's now explore some of OpenAI's key models.

Advanced prompt engineering

This notebook builds on all the knowledge and expertise you have acquired in this book in the previous chapters and previous sections. It is an advanced prompt engineering program. Take your time; this is a challenging notebook built on the philosophy of advanced AI-powered search engines.

For example, try using the new Bing app to see how Bing parses the keywords in your request, looks for web pages containing those keywords, and displays a response in conversational AI format. This will help you understand this notebook, which is a simplified version of the philosophy of search engine processes.

Up to now, when we implemented a transformer model, we only had a few options:

- Run the model as it was.

We have gone through many examples, in the previous chapters of this book, of translations, sentiment analysis, sentence completion, and more. However, sometimes, the transformer model will not fit our requirements.

- Fine-tune the model.

See *Chapter 3, Fine-Tuning BERT Models, and the Fine-tuning GPT-3* section in *Chapter 7, The Rise of Suprahuman Transformers with GPT-3 Engines*.

Fine-tuning the LLM requires reliable datasets. Designing high-quality datasets is time-consuming and requires a lot of effort. The challenge is creating training datasets that can help a model learn how to predict the correct outputs. This can be a costly and difficult endeavor.

- Train the model.

Training a model is possible, as we saw previously. For more details, see *Chapter 4, Pre-training a RoBERTa Model from Scratch*.

However, training an advanced model, such as OpenAI GPT-3, GPT-3.5-turbo, and GPT-4, requires resources that few organizations can afford..

- Prompt engineering.

We went through prompt engineering several times in the previous chapters. We also saw how to create OpenAI ChatGPT, GPT-3.5-turbo, and GPT-4 in the previous sections of this chapter. OpenAI's completion models offer the possibility to implement advanced prompt engineering through the informative prompts we can create. The introduction of the system, assistant, and user roles can help *pilot* a GPT-3.5-turbo and GPT-4 model.

This section will present prompt engineering as an alternative to fine-tuning a transformer model in some cases:

- Early in a project when we are searching for the best way to design the datasets
- When the standard model doesn't work but fine-tuning or training isn't necessary
- In any situation in which prompt engineering is sufficient

Open Prompt_Engineering_as_an_alternative_to_fine_tuning.ipynb.

The previous sections explained the basic functions to build an OpenAI completion and generation AI transformer model.

The notebook of this chapter is self-contained. There are four important features to focus on:

1. Building a knowledge base for ChatGPT
2. Building keywords and parsing user requests
3. Building a meaningful prompt that contains the information requested
4. Moderation, quality control

1. Building a knowledge base for ChatGPT and GPT-4

You can build a knowledge base with any tool you wish and then send it to the assistant object of the prompt:

```
assert1={'role': 'assistant', 'content': 'Opening hours of Rothman Consulting :Monday through Friday 9am to 5pm. Services :expert systems, rule-based systems, machine learning, deep learning, transformer models.'}
assert2={'role': 'assistant', 'content': 'Services :expert systems, rule-based systems, machine learning, deep learning, transformer models.'}
assert3={'role': 'assistant', 'content': 'Services :Fine-tuning OpenAI GPT-3 models, designing datasets, designing knowledge bases.'}
assertn={'role': 'assistant', 'content': 'Services:advanced prompt engineering using a knowledge base and SEO keyword methods.'}
```

```
#Using the knowledge base as a dataset:
kbt = []
kbt.append(assert1)
kbt.append(assert2)
kbt.append(assert3)
kbt.append(assertn)
```

2. Adding keywords and parsing user requests

You can also add keywords to the assertions:

```
assertkw1="open"
assertkw2="expert"
assertkw3="services"
assertkwn="prompt"
```

Then you can build a parser that will scan a user's request or design any other function to search for request-knowledge base similarity. The function in the notebook shows one way to parse a user's request:

```
# This is an example. You can customize this as you wish for your project
def parse_user(uprompt, kkw, kbt):
    i=0
    j=0
    for kw in kkw:
        #print(i, kw)
        rq=str(uprompt)
        k=str(kw)
        fi=rq.find(k)
        if fi>-1:
            print(kw, rq, kbt[i])
            j=i
        i+=1
    return kbt[j]
```

3. Building prompts that pilot ChatGPT

You have entered information retrieved from your knowledge base that contains keywords linked to the data.

You have parsed the user request to find keywords that fit the knowledge base assertions you made.

Now, you are ready to send a search engine-like prompt to GPT-3.5-turbo or GPT-4:

```
#convmodel="gpt-3.5-turbo"
convmodel="gpt-4"
def dialog(iprompt):
    response = openai.ChatCompletion.create(
        model=convmodel,
        messages=iprompt
    )
    return response
```

In this notebook, the user requests were loaded in a list to make requests in a batch that will call the `dialog` function.

The outputs of the batched requests are loaded in a pandas dataframe:

0	{'role': 'user', 'content': 'At what time does...	Rothman Consulting opens on Monday at 9am.	Total Tokens:83
1	{'role': 'user', 'content': 'At what time does...	Rothman Consulting is not open on Saturdays. O...	Total Tokens:97
2	{'role': 'user', 'content': 'Can you create an...	As an AI language model, I'm unable to create ...	Total Tokens:457
3	{'role': 'user', 'content': 'What services doe...	Rothman Consulting offers a wide range of serv...	Total Tokens:347

Figure 17.5: A list of user requests

Note that the number of total tokens is also displayed for budget monitoring. OpenAI will charge us for batches of 1k tokens. A token is about 75% of a word. You can thus build a cost-monitoring function.

To build an advanced prompt engineering model, a prerequisite is to implement a moderation function.

4. Moderation, quality control

Go to the **4. Moderation, quality control** cell and run the OpenAI moderation transformer model, which contains an example of how to implement the model:

```
text = "I apologize for the confusion in my previous message. Rothman Consulting is open only from Monday through Friday from 9 AM to 5 PM. We are closed on weekends, including Saturdays. If you have any further queries, please let us know."

response = openai.Moderation.create(input=text)
```

The output has many properties that are worth exploring.

```
<OpenAIObject at 0x7ff867ff7b30> JSON: {  
  "hate": false,  
  "hate/threatening": false,  
  "self-harm": false,  
  "sexual": false,  
  "sexual/minors": false,  
  "violence": false,  
  "violence/g
```

Take all the time you need to go through this notebook and understand how to implement next-generation searches with informative prompts that will pilot advanced transformer models.

Explainable AI (XAI)

You can add **XAI** to your programs if you are interested in implementing advanced prompt engineering with OpenAI's state-of-the-art models that can explain outputs. ChatGPT can explain source code. It can also explain its own outputs to a certain extent.



We went through some of the main aspects of explainable AI in *Chapter 14, Interpreting Black Box Transformer Models*.

To go further, you can try using ChatGPT to explain ChatGPT outputs and other tools by running `XAI_by_ChatGPT_for_ChatGPT.ipynb`, which is in the Bonus directory of the GitHub repository of this book. The program runs a ChatGPT XAI analysis of a ChatGPT output and also shows how to explain outputs with SHAP.

The notebook is self-contained and can help you, the advanced reader, build XAI on top of the tools in this notebook.

Let's add audio to our dialogue with ChatGPT.

Speech-to-text with Whisper

In this section, we will run a speech-to-text model using the **Whisper API** and **gTTS (Google Text-to-Speech)**. The notebook provides the basis for an oral dialogue with an OpenAI model or any type of software.

Open `Speaking_with_ChatGPT.ipynb`.

The OpenAI installation, import, and API key code is the same as for the other notebooks.

Let's begin with text-to-speech.

Text-to-speech with gTTS

Google Text-to-Speech (gTTS) can come in handy when we want our programs to “talk.” We will install and import gTTS to add text-to-speech capability.

Installing and importing gTTS

First, we install and import gTTS if necessary:

```
#3.Importing gTTS
try:
    from gtts import gTTS
except:
    !pip install gTTS
    from gtts import gTTS
    from IPython.display import Audio
Preparing the prompt
```

You can use Windows speech-to-text functionality or any other module you wish.

This notebook uses Windows. Try *Ctrl* + *H* with Windows or read the Microsoft documentation for more details: <https://support.microsoft.com/en-us/windows/dictate-text-using-speech-recognition-854ef1de-7041-9482-d755-8fdf2126ef27>.

Note: For this notebook, press *Enter* when you have finished asking for a request.

You can adapt the function to your application with a timed input function ([https://pypi.org/project/pytimedinput/#:~:text=timedInput\(\),if%20the%20user%20goes%20idle](https://pypi.org/project/pytimedinput/#:~:text=timedInput(),if%20the%20user%20goes%20idle)) or other methods.

In this case, the input message `prepare_message()` function waits for the *Enter* key to be pressed:

```
#Speech to text. Use OS speech-to-text app. For example, Windows: press
Windows Key+H
def prepare_message():
    #enter the request with a microphone or type it if you wish
    # example: "Where is Tahiti Located?"
    print("Enter a request and press ENTER:")
    uinput = input("")
```

In the notebook, the example was “Where is Tahiti located?”

The AI agent now prepares the prompt for the user object of the request API message:

```
#preparing the prompt for OpenAI
role="user"
#prompt="Where is Tahiti Located?" #maintenance or if you do not want to
use a microphone
line = {"role": role, "content": uinput}
line = {"role": role, "content": prompt}
```

You can create variables for each part of the OpenAI messages object.



For more on implementing each part of the messages object, run `Prompt_Engineering_as_an_alternative_to_fine_tuning.ipynb`, as described in the *Advanced prompt engineering* section of this chapter.

Now we can create the complete message object and return the prompt:

```
#creating the message
assert1={"role": "system", "content": "You are a helpful assistant."}
assert2={"role": "assistant", "content": "Geography is an important topic
if you are going on a once in a lifetime trip."}
assert3=line
iprompt = []
iprompt.append(assert1)
iprompt.append(assert2)
iprompt.append(assert3)

return iprompt
```

Running the GPT-3.5-turbo request

The message object of the prompt is ready. We can now make the full request with the conversational model of our choice. In this case, we will run GPT-3.5-turbo:

```
response=openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=iprompt)
```

Now we extract the response from using response:

```
#extracting the generative AI response
text=response["choices"][0]["message"]["content"]
text
```

The output for the example request (the output may change) in the notebook is:

```
Tahiti is an island located in the South Pacific Ocean. It is part of
French Polynesia and is located approximately 4,000 kilometers (2,500
miles) south of Hawaii, 7,000 kilometers (4,300 miles) east of Australia,
and 5,700 kilometers (3,500 miles) northwest of New Zealand.
```

Text-to-speech with gTTS and IPython

We have now obtained the text output and can ask Whisper to convert the text to speech. We save the text output with gTTS in a WAV file and play it with IPython audio:

```
from gtts import gTTS
from IPython.display import Audio
tts = gTTS(text)
tts.save('chatgpt.wav')
sound_file = 'chatgpt.wav'
Audio(sound_file, autoplay=True)
```

The file will be played automatically but you can listen to it again with the output's interface:



Figure 17.6: Playing the sound file

Transcribing audio files with Whisper

Speech-to-text with Whisper can allow you to transcribe audio files into text for many languages and usages: meetings, conferences, interviews, etc.

We first install the audio processing module, ffmpeg:

```
!pip install ffmpeg
```

Then, we transcribe the audio file. Whisper will automatically detect the language:

```
!whisper 1.wav
```

The output shows the language detected, the transcription, and the audio processing timeline:

```
Detected language: English
[00:00.000 --> 00:04.440] Tahiti is located in the South Pacific Ocean.
[00:04.440 --> 00:06.320] Specifically in French Polynesia.
[00:06.320 --> 00:10.720] It is the largest island in the Windward Group
of Islands in French Polynesi
```

The output is also written in several file formats (json, srt, tsv, txt, and vtt).

Let's extract the text object in json:

```
import json

with open('1.json') as f:
    data = json.load(f)

text = data['text']
print(text)
```

For the notebook example, the text is:

```
Tahiti is located in the South Pacific Ocean, specifically in the
archipelago of society islands, and is part of French Polynesia. It is
approximately 4,000 miles, 6,400 km, south of Hawaii and 5,700 miles,
9,200 km, west of Santiago, Chile.
```

You are now ready to expand this notebook to any project you wish.

Let's create images with the DALL-E API.

Getting started with the DALL-E 2 API

DALL-E's text-to-image functionality has dramatically improved in a short time. Transformers are task-agnostic and can perform a variety of tasks. Transformers are now also multimodal (audio, image, other signals).

We previously went through DALL-E's architecture in the *DALL-E* section of *Chapter 15, From NLP to Task-Agnostic Transformer Models*.

You can try DALL-E 2 online: <https://openai.com/product/dall-e-2>.

However, this section goes further using the second-generation API of the DALL-E 2 API so that we can write a program. The API allows us to create, modify, and generate variations of an image.

Open `Getting_Started_with_the_DALL_E_2_API.ipynb`.

You can implement the DALL-E 2 API as you see fit for your project.

The notebook presents one way of running the DALL-E 2 API.

You can run it cell by cell to grasp the functionality provided or run the whole notebook for one scenario.

This section is divided into two parts:

- Creating a new image
- Creating a variation of an image

Let's first create an image.

Creating a new image

The first cell sets the goal of the notebook. You can use it for all scenarios.

In this example, we will create a new image of a person talking to a chatbot in a restaurant near the moon and save the image to a file:

```
#prompt
sequence="Creating an image of a person using a chatbot in a restaurant on
a spaceship near the moon."
```

`sequence` is the text prompt to instruct DALL-E 2 to create an image.

Run the OpenAI install, import, and API key cells in the notebook.

Now go to the Generation cell and run the DALL-E 2 API:

```
#creating an image or images
response = openai.Image.create(
    prompt=sequence,
    n=2, #number of images to produce
    size="1024x1024")
image_url = response['data'][0]['url']
```

The request begins with `openai.Image.create`. The prompt is the sequence we defined following the documentation cited in the first cells of this notebook.

The image is generated. Skip the following cells and go to the `Displaying Generation` cell, which uses a standard PIL function:

```
# displaying the image
url = image_url
image = Image.open(requests.get(url, stream=True).raw)
image.save("c_image.png", "PNG")
c_image = Image.open(requests.get(url, stream=True).raw)
c_image
```

Your image is now displayed:

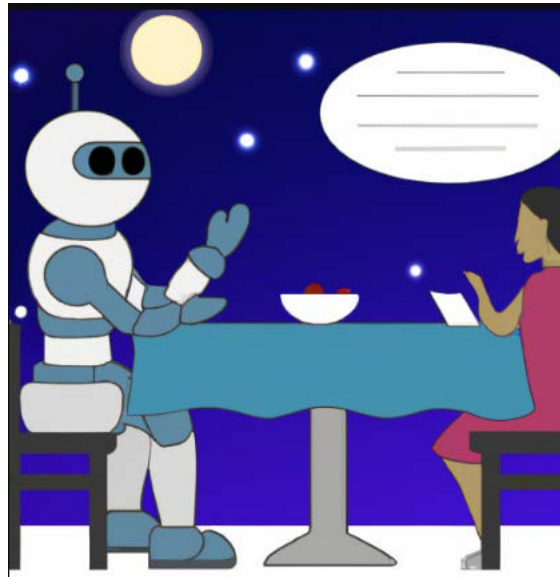


Figure 17.7: Image created by the DALL-2 API

Each time you run the API for the same sequence, you might obtain a different version of the same image.

The image is also saved:

```
image.save("c_image.png", "PNG")
```

Now, let's build a variation of an image.

Creating a variation of an image

Let's see how creative DALL-E 2 can be and ask it to create a variation for the image that was created and saved as `c_image.png`.

The following is an example of the Variation cell. We select an image and ask DALL-E to create a variation of it, similar to the original one but more creative:

```
#creating a variation of an image
response = openai.Image.create_variation(
    image=open("c_image.png", "rb"),
    n=1,
    size="1024x1024")
image_url = response['data'][0]['url']
```

The image is displayed:

```
image_url = response['data'][0]['url']
image = Image.open(requests.get(url, stream=True).raw)
v_image = Image.open(requests.get(url, stream=True).raw)
v_image
```

The output can be a nice variation, a slight variation, or no variation. It will depend on our prompt and the image we ask DALL-E 2 to process. It also depends on the advancement of OpenAI in this field.

You now know how to create and generate variations of images with the DALL-E 2 API.

You are on the road to becoming an AI image artist!

The excitement is there. Let's put it all together.

Putting it all together

Before moving on to the content summary section, let's sum up the models we discovered in one notebook. This chapter took you to the cutting edge of artificial intelligence, to a place where everything is dynamic, constantly evolving like the birth of a galaxy.

We went through OpenAI's many innovations with ChatGPT and a variety of new models: GPT-3.5-turbo, GPT-4, moderation, Whisper, and DALL-E. We also implemented gTTS.

You have seen how to build programs with these models. But it's a lot to digest.

To summarize the chapter notebooks and take you further, run the `ALL-in-ONE.ipynb` notebook. It contains an entertaining scenario that will help you review all the models in a nutshell:

1. Install OpenAI and the modules for this notebook.
2. Enter a request.
3. Check if the content is safe with the Moderation model.
4. Prepare the prompt for ChatGPT 5. ChatGPT 3.5-turbo tells a story.
5. GPT-4 writes a poem based on the story.
6. DALL-E 2 illustrates the poem.
7. Listen to the poem with gTTS.
8. Whisper transcribes the audio.

Run this notebook to review the key cutting-edge OpenAI models in a few minutes. You'll always be on top of the innovation curve!



Figure 17.8: DALL-E 2 illustrates the poem

Click on **Run all** under the **Runtime** menu in Google Colab and enjoy the magic! We are now ready to summarize what we explored in the chapter.

Summary

In this chapter, we built on the knowledge and expertise you acquired in the previous chapters to tackle OpenAI's state-of-the-art transformer models.

The knowledge you now possess enables you to make an incremental step forward from the GPT-3 model you already discovered in *Chapter 7, The Rise of Suprahuman Transformers with GPT-3 Engines*. For those who are beginning to learn transformers now, the path will be quite long.

We first jump-started ChatGPT using the same approach as in *Chapter 7*. The new step was to implement a conversational AI prompt. You saw how ChatGPT Plus can generate a k-means clustering classification program, plot the outputs, and provide explanations.

Getting started with GPT-4 was an incremental step forward in implementing a powerful general-purpose transformer model.

Model exploration took you into the world of 50+ transformer models, including davinci, GPT-3.5-turbo, and GPT-4.

Explainable AI by ChatGPT opened the OpenAI toolbox for ChatGPT Plus, GPT-3.5, GPT-4, and davinci to explain transformer outputs

Advanced prompt engineering showed you how to start building a knowledge base using prompts that add data to requests. This data can progressively become an organization's knowledge base.

We then added speech to text with OpenAI's transformer models.

You saw how DALL-E provides a text-to-image model that can create, modify, and produce variations of an image.

Finally, we summed up the models in an all-in-one notebook so you can review them in a nutshell.

It's now up to you to keep up to speed with the LLM and generative AI movement. You can explore the Bonus directory for more advanced AI notebooks. You are also all set to create transformer programs.

Use your knowledge and expertise to accomplish wonderful projects!

Questions

1. GPT-4 is sentient. (True/False)
2. ChatGPT can replace a human expert. (True/False)
3. GPT-4 can generate source code for any task. (True/False)
4. Advanced prompt engineering is intuitive. (True/False)
5. The most advanced transformer, that is, GPT-4, is the best model to use. (True/False)
6. Developing applications with transformers will require no training since copilots such as GPT-4 can do the job. (True/False)
7. GPT-4 will be the last OpenAI transformer model because it has reached the limit of AI. (True/False)

References

- OpenAI ChatGPT blog: <https://openai.com/blog/chatgpt>
- OpenAI blog introducing the ChatGPT and Whisper APIs: <https://openai.com/blog/introducing-chatgpt-and-whisper-apis>
- OpenAI blog introducing ChatGPT Plus: <https://openai.com/blog/chatgpt-plus>
- OpenAI models: <https://platform.openai.com/docs/models/>
- OpenAI GPT-4: <https://openai.com/product/gpt-4>
- OpenAI Codex deprecated: <https://openai.com/blog/openai-codex>
- Microsoft GitHub Copilot with Codex (March 2023): <https://github.com/features/copilot>
- Google Text-to-Speech: <https://pypi.org/project/gTTS/>

Join our book's Discord space

Join the book's Discord workspace:

<https://www.packt.link/Transformers>

