Join our book community on Discord

https://packt.link/EarlyAccessCommunity

Chapter 7, From Basics to Disruption with ChatGPT, took us into the world of GPT assistants and APIs. We examined the Generative AI Transformer, General Purpose Technology, through interactions and code. If all the technology we explored is so effective, why bother? Why not just use the ready-made solutions and forget about learning, preparing data, configuring parameters, and writing documentation?The answer is quite simple. What can we do when a general-purpose model such as GPT-3 or ChatGPT doesn't reach the accuracy threshold we need for our project? We have to do something. But what? We'll start the chapter by fitting fine-tuning in the landscape of *Tracing transformer paths in GPT* and make sense of the choices we can make for a project to fine-tune or not.In this chapter, we will decide to fine-tune an OpenAI GPT-3 model. We will fine-tune an Ada model for a completion task and a classification task. We will prepare the datasets in the format required by OpenAI. This step seems deceivingly easy. However, preparing datasets can prove quite challenging.We will then launch OpenAI's fine-tuning service. The fine-tuning service can take a few minutes or longer if we are in a queue. We can view our fine-tuned models, which are stored on OpenAI. We will run a completion task with Immanuel Kant's work. We will also run a classification task. The validation process will also consist in running a classification task to determine if the content of a text is related to hockey or baseball. What if a standard pretrained model can do the job? We need to verify that and will with a `davinci` engine.Finally, we explore the capabilities of Weight & Biases' Wandb, providing insights, graphs, logs, and more on fine-tuning.By the end of the chapter, you will have built a complete fine-tuning process.This chapter covers the following topics:

The improvements and limitations of fine-tuning models on OpenAI

Dataset preparation

OpenAI's file format

Converting JSON files to JSONL

Fine-tuning an original OpenAI GPT-3 model

Running the fine-tuned model

Running a completion task(generative)

Running a classification task(discriminative)

Managing fine-tuned models

Implementing Weights & Biases' Wandb

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/Chapter08.

The journey begins with tracing the GPT path.

Tracing transformer paths in GPT

From an emerging General Purpose Technology perspective, fine-tuning OpenAI models can be summed as productivity improvements and diffusion limitations:

Improvements

Automated dataset control

OpenAI provides a data-preparation tool that will accept the data, try to correct the errors, and explain the potential issues.

Intuitive fine-tuning

OpenAI models can be fine-tuned with a few instructions.

Synchronized metrics

OpenAI can be synchronized with Weights & Biases' Wandb, to produce information on the fine-tuning process, guaranteeing a productive level of traceability.

A Generative model can be fine-tuned for a completion (generative) task and a classification (discriminative) task, as shown in *Figure 8.1*:

Figure 8.1: A generative model can be generative and discriminative. Source: Tracing_Transformer_Paths_in_GPT.ipynb in Appendix1, Terminology

Diffusion

ChatGPT reached mainstream users as ready-to-use assistants. The diffusion of fine-tuning functionality is limited to the constraints involved:

Developer tools for initial adopters (m).

Fine-tuning requires dataset preparation and knowing how to use developer-level tools and APIs.

Possible limitations for security, privacy, and compliance requirements.

OpenAI has a wonderful platform. However, data privacy can be a concern for confidential information, for example. In that case, fine-tuning might not be possible on the OpenAI platform.A project manager will have to take other considerations into account, as shown in *Figure 8.2*:

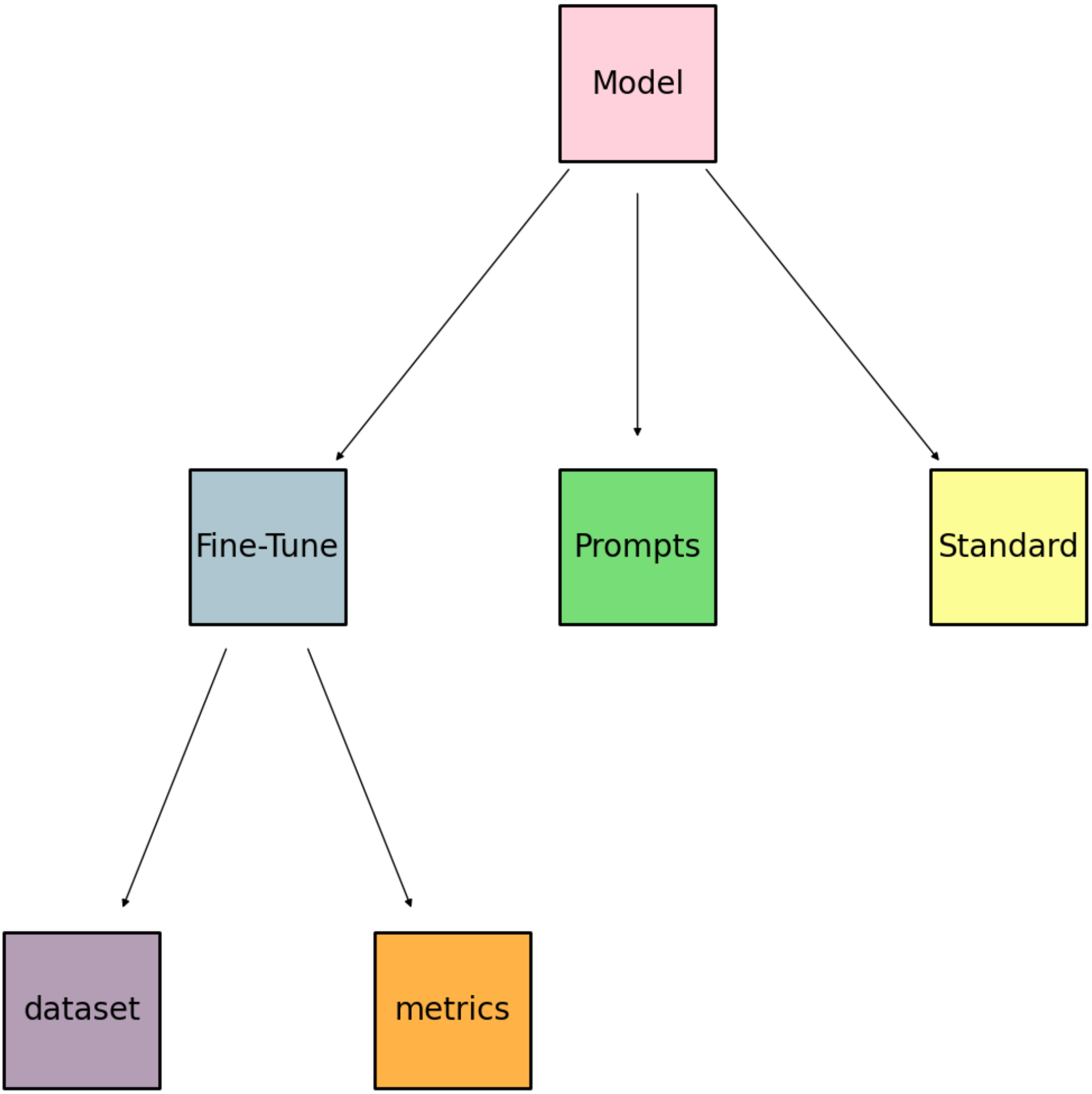

Figure 8.2: Deciding to customize a model or not. Source: Tracing_Transformer_Paths_in_GPT.ipynb in Appendix1, Terminology

Figure 8.2 shows that we can fine-tune with a dataset and metrics. However, some projects might require prompt engineering instead of fine-tuning. In some cases, the standard model might not require fine-tuning or prompt engineering.In *Chapter 15, Guarding the Giants: Mitigating Risks in Large Language Models*, we will implement advanced prompt engineering as an alternative to fine-tuning an OpenAI model.The takeaway is that choosing between using a standard pretrained model, fine-tuning a model, and advanced prompt engineering will depend on each project. Fine-tuning might provide more precise results, process more training data, and save tokens through shorter prompts. Prompt engineering might provide more customized implementations and control. There is no miracle method or tool to customize

transformer models. Each project will require a careful study to balance the resources, costs, and results. We can now examine how fine-tuning GPT-3 works for a completion (generative) task.

Fine-tuning GPT-3 for completion (generative)

OpenAI (at the time of the writing of this book) has a service to fine-tune the following original GPT-3 models: davinci, curie, babbage, and ada. They are original models and, as such, have no suffixes. GPT-4 models are not available for fine-tuning at the time of the writing of this book. However, if GPT-4 models become available for fine-tuning, the same or similar process as for GPT-3 will apply.A fine-tuned model can perform data exploration, classification, question answering, and other NLP tasks like the original models. As such, the fined-tuned model might produce acceptable or inaccurate results. Quality control remains essential. Make sure to go through OpenAI's documentation before beginning a project: https://platform.openai.com/docs/guides/fine-tuning/This section aims to implement the fine-tuning process of a model in a notebook, cell by cell, so you can apply fine-tuning to your specific domain.Fine-tuning GPT-3 models involves four phases that we will implement in this section:

1. 1. Preparing the data
2. 2. Fine-tuning a GPT-3 model for a generative task (completion)
3. 3. Running the fine-tuned GPT-3 model
4. 4. Managing the models

Once you have learned to fine-tune a GPT-3 model, you can use other types of data to teach it specific domains, knowledge graphs, and texts.

Evaluate the cost to fine-tune an OpenAI model before running the program in this section.

In the *Fine-tuned for classification (discriminative)* section, we will fine-tune a model for a classification task. To get started, open `Fine_Tuning_GPT_3.ipynb` in Google Colab in the GitHub chapter directory and first install the OpenAI library and enter your key:

Copy Explain

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

You can also install wandb to visualize the logs:

Copy Explain

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

Now retrieve your key from a file or comment the code below and enter your key manually.To read the key from the file, you can use Google Drive or any filesystem of your choice:

Copy Explain

```
#You can retrieve your API key from a file(1)
# or enter it manually(2)
#Comment this cell if you want to enter your key manually.
#(1)Retrieve the API Key from a file
#Store you key in a file and read it(you can type it directly in the notebook but it
will be visible for somebody next to you)
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 let the program read your key from a file or comment the code above and enter your key manually:

Copy Explain

```
#(2) Enter your manually by
# replacing API_KEY by your key.
#The OpenAI Key
import os
os.environ['OPENAI_API_KEY'] =API_KEY
openai.api_key = os.getenv("OPENAI_API_KEY")
```

We'll also install `wandb` to track and visualize the fine-tuning process

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

You can also synchronize `openai` and `wandb` to access your tracking data. You will need a `wandb` key when prompted:

```
!openai wandb sync
```

You can also run `Wandb.ipynb` in this section's *Wandb – Weights & Biases* subsection.You are now ready to prepare the dataset and fine-tune an OpenAI model.

1. Preparing the dataset

OpenAI has documented the data preparation process in detail:https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset For this fine-tuning session, we will download and process the *Critique of Pure Reason by Immanuel Kant* for the Project Gutenberg website. The content of this book is challenging for machines as well as humans and, thus, exciting to use as a dataset.The first step is the prepare a dataset.

1.1. Preparing the data in JSON

OpenAI requires a formatted file with prompts and completions. JSON is one of the recommended file formats. We will first import the necessary libraries. We will need the following:

The Natural Language Toolkit with `punk`, a pretrained tokenizer that we cover in more detail in *Chapter 10, Investigating the Role of Tokenizers in Shaping Transformer Models.* `sent_tokenize` will run in conjunction with `punk`.

`requests` for the HTTP request

`BeautifulSoup` to scrape the web page of the book

`json` to create a json file

`re` to parse the text with regular expressions

With these tools, we can go from Gutenberg to JSON with the book of our choice:

```
#From Gutenberg to JSON
import nltk
nltk.download('punkt')
from nltk.tokenize import sent_tokenize
import requests
from bs4 import BeautifulSoup
import json
import re
```

We first get the book directly from Project Gutenberg by uncommenting the following code:

```
# First, fetch the text of the book from Project Gutenberg
#url = 'http://www.gutenberg.org/cache/epub/4280/pg4280.html'
#response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')
```

Or we can download the book from the GitHub repository and read it:

```
# Option 2: from the GitHub repository:
#Development access to delete when going into production
!curl -H 'Authorization: token {github_token}' -L
https://raw.githubusercontent.com/Denis2054/Transformers_3rd_Edition/master/Chapter08/gute
--output "gutenberg.org_cache_epub_4280_pg4280.html"
# Open and read the downloaded HTML file
with open("gutenberg.org_cache_epub_4280_pg4280.html", 'r', encoding='utf-8') as file:
    file_contents = file.read()
soup = BeautifulSoup(response.content, 'html.parser')
```

Then we clean it up and split the sentences:

```
# Get the text of the book and clean it up a bit
text = soup.get_text()
text = re.sub('\s+', ' ', text).strip()
# Split the text into sentences
sentences = sent_tokenize(text)
```

Now comes a tricky part. OpenAI expects a prompt and a completion for each line we want to train it for. However, the separators need to be carefully chosen:

```
# Define the separator and ending
prompt_separator = " ->"
completion_ending = "\n"
```

With these well-define separators, we can create the prompts and completions:

```
# Now create the prompts and completions
data = []
for i in range(len(sentences) - 1):
    data.append({
        "prompt": sentences[i] + prompt_separator,
        "completion": " " + sentences[i + 1] + completion_ending
    })
```

Finally, we save the prompt and completions in a json file:

```
# Write the prompts and completions to a file
with open('kant_prompts_and_completions.json', 'w') as f:
    for line in data:
        f.write(json.dumps(line) + '\n')
```

This process must be carefully prepared to avoid the dataset being partially or totally rejected by OpenAI's preparation module.We now check our dataset before submitting it to the data preparation module:

Copy Explain

```
import pandas as pd
# Load the data
df = pd.read_json('kant_prompts_and_completions.json', lines=True)
df
```

The output shows that we did a good job. We have a prompt and completion per line:

	prompt	completion
0	The Project Gutenberg Etext of The Critique of...	Be sure to check the copyright laws for your ...
1	Be sure to check the copyright laws for your c...	We encourage you to keep this file, exactly a...
2	We encourage you to keep this file, exactly as...	Please do not remove this.\n
3	Please do not remove this. ->	This header should be the first thing seen wh...
4	This header should be the first thing seen whe...	Do not change or edit it without written perm...
...	...	...
6122	78-79. is their motto, under which they may le...	As regards those who wish to pursue a scienti...
6123	As regards those who wish to pursue a scientif...	When I mention, in relation to the former, th...
6124	When I mention, in relation to the former, the...	The critical path alone is still open.\n
6125	The critical path alone is still open. ->	If my reader has been kind and patient enough...
6126	If my reader has been kind and patient enough ...	End of Project Gutenberg's The Critique of Pu...

6127 rows × 2 columns

Figure 8.3: The prompt and completion data

We're ready to run OpenAI's data preparation module.

1.2. Converting the data to JSONL

If the preparation of the JSON file went well, OpenAI's data preparation tool should run smoothly with the file we just created: kant_prompts_and_completions.jsonWe now run the `fine_tunes` tool:

Copy Explain

```
!openai tools fine_tunes.prepare_data -f "kant_prompts_and_completions.json"
```

The output shows that there are no errors in the dataset:

```
Analyzing...
- Your JSON file appears to be in a JSONL format. Your file will be converted to JSONL format
- Your file contains 6127 prompt-completion pairs
- All prompts end with suffix ` ->`
- All completions end with suffix `\n`
Based on the analysis we will perform the following actions:
```

- [Necessary] Your format `JSON` will be converted to `JSONL` If your dataset contains errors, the preparation tool will try to fix the data. You might lose data in the process.In this case, everything is fine. You will be prompted to convert the JSON file into a JSONL file:

```
Your data will be written to a new JSONL file. Proceed [Y/n]: Y
Wrote modified file to `kant_prompts_and_completions_prepared.jsonl`
Feel free to take a look!
```

A few lines later, OpenAI provides information on how to run a prompt:

```
After you've fine-tuned a model, remember that your prompt has to end with the indicator string ` ->` for the model to start generating completions, rather than continuing with the prompt. Make sure to include `stop=["\n"]` so that the generated texts ends at the expected place.
```

Once your model starts training, it'll approximately take 1.44 hours to train a `curie` model, and less for `ada` and `babbage`. Queue will approximately take half an hour per job ahead of you.OpenAI saves the converted file to `kantgpt_prepared.jsonl`.We will now take a look at the JSONL file as recommended:

```
import json
# Open the file and read the lines
with open('kant_prompts_and_completions_prepared.jsonl', 'r') as f:
    lines = f.readlines()
# Parse and print the first 5 lines
for line in lines[:5]:
    data = json.loads(line)
    print(json.dumps(data, indent=4))
```

The output shows that everything went well:

```
{
    "prompt": "The Project Gutenberg Etext of The Critique of Pure Reason, by Immanuel Kant Copyright laws are changing all over the world. ->",
    "completion": " Be sure to check the copyright laws for your country before distributing this or any other Project Gutenberg file.\n"
}
```

Our last preparation step consists of creating the file on OpenAI:

```
openai.File.create(
  file=open("/content/kant_prompts_and_completions_prepared.jsonl", "rb"),
  purpose='fine-tune'
)
```

We are ready to fine-tune an OpenAI model.

2. Fine-tuning an original model

You can fine-tune several models on OpenAI, including GPT-3.5-turbo and GPT-4. You can also fine-tune GPT-3 models such as babbage-002:https://platform.openai.com/docs/guides/fine-tuning/what-models-can-be-fine-tunedFine-tuning babbage-002 only costs a small fraction of the cost of fine-tuning GPT-4, for example. Start with a cost-effective model and use it as your base model. Then, if you need better results, try the more expensive ones if necessary.Check the data preparation constraints and dataset formats of a model before fine-tuning it.We will fine-tune a babbage-002 model in this section.The fine-tuning process is ready to run with the JSONL file:

```
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.FineTuningJob.create(training_file="file-ER2cK59joySxwqaOx7MrbGOD",
model="babbage-002")
```

OpenAI has many requests. The fine-tuning request we make is in a queue. You will generally receive an email when the job is completed.However, you can run the following code to see the list of existing jobs:

Copy Explain

```
# List 10 fine-tuning jobs
openai.FineTuningJob.list(limit=10)
```

A list will be displayed:

Copy Explain

```
<OpenAIObject list at 0x7d339c5afc40> JSON: {
  "object": "list",
  "data": [
    {
      "object": "fine_tuning.job",
      "id": "ftjob-whkMkwaf2WDjJIsezLMC23T3",
      "model": "babbage-002",
      "created_at": 1693325760,
      "finished_at": null,
      "fine_tuned_model": null,
      "organization_id": "org-h2Kjmcir4wyGtqq1mJALLGIb",
      "result_files": [],
      "status": "running",
      "validation_file": null,
      "training_file": "file-ER2cK59joySxwqaOx7MrbGOD",
      "hyperparameters": {
        "n_epochs": 3
      },
      "trained_tokens": null
    },
```

Check the status of your current job in the list with the `"status":` property. In this case, the fine-tuning job is running:

Copy Explain

```
"status": "running"
```

You can also check the status of your specific current job to see if it is completed:

Copy Explain

```
# Retrieve the state of a fine-tune
openai.FineTuningJob.retrieve("ftjob-40XaISEdOEoitwxmsGQldiWJ")
```

In this case, the `status` property shows that the fine-tuning has succeeded:

Copy Explain

```
<FineTuningJob fine_tuning.job id=ftjob-40XaISEdOEoitwxmsGQldiWJ at 0x7d33947cb920>
JSON: {
  "object": "fine_tuning.job",
  "id": "ftjob-40XaISEdOEoitwxmsGQldiWJ",
  "model": "babbage-002",
  "created_at": 1693226909,
  "finished_at": 1693227408,
  "fine_tuned_model": "ft:babbage-002:personal::7sW5N8i3",
  "organization_id": "org-h2Kjmcir4wyGtqq1mJALLGIb",
  "result_files": [
    "file-1ihDmqKCzYF726Oc0TuQfWlB"
  ],
  "status": "succeeded",
  "validation_file": null,
  "training_file": "file-ER2cK59joySxwqaOx7MrbGOD",
  "hyperparameters": {
    "n_epochs": 3
  },
  "trained_tokens": 1564743
}
```

The trained model name is indicated:

Copy Explain

```
  "fine_tuned_model": "ft:babbage-002:personal::7sW5N8i3",
```

You can save the name of the model in a file, for example, for future use.The model is trained and ready for a completion task.

3. Running the fine-tuned GPT-3 model

You can run your fined-tuned model as any original model. You can run the model directly or do some organizing before. We will run the model for a completion task in this section.In the following example, the name of the fine-tuned model was saved first in a text file for future use. Then, the program reads the file, retrieves the fined-tuned model name, and displays it.

Copy Explain

```
f = open("drive/MyDrive/files/fine_tune.txt", "r")
FINE_TUNE=f.readline().strip() #load your saved model from a file or load it in this
variable
f.close()
FINE_TUNE
```

The output will be the name of the fine-tuned mode, for example:

```
ft:babbage-002:personal::7sW5N8i3
```

You can create a standard completion prompt to run your model with the parameters you wish to implement to steer the response:

```
prompt = "Freedom can be a concept or a virtue ->"
response=openai.Completion.create(
  model=FINE_TUNE, #Your model in FINE_TUNE,
  prompt=prompt,
  temperature=1,
  top_p=1,
  frequency_penalty=0,
  presence_penalty=0,
  stop="\n",
  max_tokens=200
)
```

You can go back to the Running NLP tasks with OpenAI GPT-3 section of this chapter for the description of each parameter. You can modify them to fit your needs.In this case, for example:

the prompt ends with "`->` "as in our dataset

`stop="\n"` tells the model to stop after generating a line

We have run a prompt with our fine-tuned model and obtained a raw output that we can display with `response`:

```
response
```

Or that we can process and wrap to get a clean output:

```
import textwrap
generated_text = response['choices'][0]['text']
# Remove leading and trailing whitespace
generated_text = generated_text.strip()
# Convert to a pretty paragraph by replacing newline characters with spaces
single_line_response = generated_text.replace('\n', ' ')
# Use textwrap.fill to nicely format the paragraph to wrap at 80 characters (or
whatever width you prefer)
wrapped_response = textwrap.fill(single_line_response, width=80)
print(wrapped_response)
```

The output will display the wrapped response:

```
Human nature is autarchy; therefore, our conduct can be very good, while human nature
is very bad, and yet, unless we abandon the idea of liberty, our conduct will be
absolutely good.
```

We will now add some management functions.

4. Managing fine-tuned models

At the time of the writing of this book, you cannot share fine-tuned models outside of your organization. You can deploy them for your projects.You can create several fine-tuned models, and you will need to manage them. You can ask OpenAI to display a list of your fined-tuned models and store them in a json file:

```
# List all created fine-tunes
!openai api fine_tunes.list > fine_tunes.json
```

We also run the following line to see the raw output as well:

```
!openai api fine_tunes.list
```

The raw output displays the details of each fine-tuned model in a json object. ChatGPT, GPT-4, explains each variable of the object in the next cell of the notebook, as shown in the following excerpt:

```
ChatGPT PLUS, GPT-4, provides a breakdown of the components of the JSON object
```

"object": This line specifies the type of object the JSON is representing. Here it's a fine-tuned model.

"id": This is the unique identifier for this fine-tuning job. This ID is typically used to reference this specific instance of fine-tuning.

You can also display the list of models with some of the main data in a Pandas DataFrame by loading the fine_tunes.json we just saved, converting the data to a Pandas DataFrame, selecting columns, renaming them for display, and converting the UNIX timestamps into a standard format, and display the DataFrame:

```
import pandas as pd
import json
from datetime import datetime
# Load data from json file
with open('fine_tunes.json') as f:
    data = json.load(f)
# Convert to a Pandas DataFrame:
df = pd.json_normalize(data['data'])
# Select specific columns
selected_columns = ['object', 'id', 'fine_tuned_model','status', 'created_at',
'updated_at']
df = df[selected_columns]
# Rename columns for display
column_mapping = {
    'object': 'Object',
    'id': 'ID',
    'fine_tuned_model': 'Fine_Tuned_Model',
    'filename':'Filename',
    'status': 'Status',
    'created_at': 'Created_At',
    'updated_at': 'Updated_At',
}
df.rename(columns=column_mapping, inplace=True)
# Convert UNIX timestamp to standard format
df['Created_At'] = pd.to_datetime(df['Created_At'], unit='s')
df['Updated_At'] = pd.to_datetime(df['Updated_At'], unit='s')
df
```

The output will be a Pandas DataFrame containing the information requested.Finally, you can select one of the models and delete it:

Copy Explain

```
#delete a model
# enter a model in the list of fine-tuned models
#FINE_TUNED_MODEL=[MODEL in list]
#try:
#  openai.Model.delete(FINE_TUNED_MODEL)
#except:
#  print("FINE_TUNED_MODEL not found")
```

You can also isolate each phase in separate notebooks:

The data in JSON can be prepared in a separate notebook or even a separate project to create a knowledge dataset. You can create prompts and completions for many other NLP tasks, such as classification, data exploration, and whatever task you need for your project. You can save and load the file when needed for OpenAI's fine-tuning tools.

1.2. Converting the data to JSONL can be an excellent way to visualize your dataset for OpenAI's fine-tunes or even as a knowledge dataset for any project. You can save the file and fine-tune an OpenAI model when you are ready or when it's required.

2. Fine-tuning an OpenAI model can be a separate program. You can load the prepared dataset when you wish.

3. Running the fine-tuned GPT-3 model can be done in a separate notebook or application.

You can also run the management functions in this section in another notebook or application. The flexibility of the fine-tuning process will alleviate the tension of each phase, which requires careful preparation and resources.

Fine-tuned for classification (discriminative)

The miracle of generative models, such as GPTs, is that they can perform a classification task with the right prompts!In this section, we will fine-tune babbage-002 to classify baseball and hockey text inputs. You will see that you can fine-tune an original OpenAI model to a wide range of tasks. Your imagination will be the limit!Open `Fine-tuned_classification.ipynb` in the chapter directory in the GitHub repository. The structure of the notebook is the same as `Fine_tuning_GPT_3.ipynb` notebook we just created. The main section titles are identical.`Fine_tuning_GPT_3.ipynb` was created for completion tasks with text as a prompt and completion, although you can modify it for any other NLP task. `Fine-tuned_classification.ipynb` is designed to classify baseball and hockey texts. You can adapt this notebook to other NLP tasks once you have explored it.The dataset is designed for classification tasks, but the process is the same as the one we went through in the previous section. The main difference between the two fine-tuning processes is that `Fine-tuned_classification.ipynb` contains a measurement functionality.We will go through the phases of the fine-tuning process and focus on the differences between the two notebooks in the sections that contain new information.The titles of the following subsections are the titles in the notebook.

Section 1.1. Preparing the data in JSON

In this section, the dataset is split into prompts and labels (baseball or hockey), which are completions:

Copy Explain

```
import pandas as pd
labels = [sports_dataset.target_names[x].split('.')[-1] for x in
sports_dataset['target']]
texts = [text.strip() for text in sports_dataset['data']]
df = pd.DataFrame(zip(texts, labels), columns = ['prompt','completion']) #[:300]
df.head()
```

The output shows that each text message now has a label:

prompt	completion
0	From: dougb@comm.mot.com (Doug Bank)\nSubject:...
1	From: gld@cunixb.cc.columbia.edu (Gary L Dare)...
2	From: rudy@netcom.com (Rudy Wade)\nSubject: Re...

Section 1.2. Converting the data to JSONL

OpenAI's data preparation tool provides machine learning functionality to split the data into training and validation sets:

```
!openai tools fine_tunes.prepare_data -f sport2.jsonl -q
```

`-f` specifies the filename, and `-q` triggers auto-accept for the questions asked. The data preparation tool detects the specific classification structure of the dataset and asks the right questions to add a suffix to the prompts and a whitespace to the beginning of the completion(a label, in this case). It also asks to split the dataset into a training and validation set :

```
- [Recommended] Add a suffix separator `\n\n###\n\n` to all prompts [Y/n]: Y
- [Recommended] Add a whitespace character to the beginning of the completion [Y/n]: Y
```

- [Recommended] Would you like to split into training and validation set? [Y/n]: YThe tool will generate two standard machine learning files, train and validation ("valid"):

"sport2_prepared_train.jsonl"

"sport2_prepared_valid.jsonl"

We create the training file on OpenAI:

```
openai.File.create(
  file=open("/content/sport2_prepared_train.jsonl", "rb"),
  purpose='fine-tune'
```

The `id` of our training file is displayed in the output:

Copy Explain

```
<File file id=file-QmcRGPcsCJf6R87OSs5alf3A at 0x7f143f344bd0> JSON: {
  "object": "file",
  "id": "file-QmcRGPcsCJf6R87OSs5alf3A",
  "purpose": "fine-tune",
  "filename": "file",
  "bytes": 1519036,
  "created_at": 1693235763,
  "status": "uploaded",
  "status_details": null
}
```

We do the same for the validation file:

Copy Explain

```
openai.File.create(
  file=open("/content/sport2_prepared_valid.jsonl", "rb"),
  purpose='fine-tune'
)
```

The `id` of our validation file is displayed in the output:

Copy Explain

```
<File file id=file-mCLhXE6D5RW7rRzQwEbzCFfo at 0x7f143f8b9f30> JSON: {
  "object": "file",
  "id": "file-mCLhXE6D5RW7rRzQwEbzCFfo",
  "purpose": "fine-tune",
  "filename": "file",
  "bytes": 387349,
  "created_at": 1693235843,
  "status": "uploaded",
  "status_details": null
}
```

We can now fine-tune an OpenAI model.

Section 2. Fine-tuning an original model

We now have two classical machine learning files ready for the training phase and we run it:

```
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.FineTuningJob.create(training_file="file-QmcRGPcsCJf6R87OSs5alf3A",
validation_file="file-mCLhXE6D5RW7rRzQwEbzCFfo", model="babbage-002")
```

The training job is created and the tracking information is displayed:

```
<FineTuningJob fine_tuning.job id=ftjob-XcWeeIdVarCz61BypAWCvBGG at 0x7f143f403100>
JSON: {
  "object": "fine_tuning.job",
  "id": "ftjob-XcWeeIdVarCz61BypAWCvBGG",
  "model": "babbage-002",
  "created_at": 1693236026,
  "finished_at": null,
  "fine_tuned_model": null,
  "organization_id": "org-h2Kjmcir4wyGtqq1mJALLGIb",
  "result_files": [],
  "status": "created",
  "validation_file": "file-mCLhXE6D5RW7rRzQwEbzCFfo",
  "training_file": "file-QmcRGPcsCJf6R87OSs5alf3A",
  "hyperparameters": {
    "n_epochs": 3
  },
  "trained_tokens": null
}
```

You can follow the fine-tuning status with the tracking instructions and display the information of the model when it's trained as in `Fine_tuning_GPT_3.ipynb`. The following section will measure the model's accuracy.

Section 3. Running the fine-tuned GPT-3 model

We can now take samples from the validation dataset:

```
test = pd.read_json('sport2_prepared_valid.jsonl', lines=True)
test.head(10)
```

The dataset contains the prompts and their completion labels:

	prompt	completion
0	From: gld@cunixb.cc.columbia.edu (Gary L Dare)...	hockey
1	From: smorris@venus.lerc.nasa.gov (Ron Morris ...	hockey
2	From: golchowy@alchemy.chem.utoronto.ca (Geral...	hockey
3	From: krattige@hpcc01.corp.hp.com (Kim Krattig...	baseball
4	From: warped@cs.montana.edu (Doug Dolven)\nSub...	baseball
5	From: jerry@sheldev.shel.isc-br.com (Gerald La...	baseball
6	From: kime@mongoose.torolab.ibm.com (Edward Ki...	baseball
7	From: cmk@athena.mit.edu (Charles M Kozierok)\...	baseball
8	From: pkortela@snakemail.hut.fi (Petteri Korte...	hockey
9	From: gak@wrs.com (Richard Stueven)\nSubject: ...	hockey

Figure 8.4: The prompt and labels data

The first record is a hockey message:

```
From: gld@cunixb.cc.columbia.edu (Gary L Dare) Subject: Re: Flames Truly Brutal in
Loss Nntp-Posting-Host: cunixb.cc.columbia.edu Reply-To: gld@cunixb.cc.columbia.edu
(Gary L Dare) Organization: PhDs In The Hall Distribution: na Lines: 13 This game
would have been great as part of a double-header on ABC or ESPN; the league would have
been able to push back-to-back wins by Le Magnifique and The Great One. Unfortunately,
the only network that would have done that was SCA, seen in few areas and hard to
justify as a pay channel. )-; gld -- ~~~~~~~~~~~~~~~~~~~~~~~~ Je me souviens
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Gary L. Dare > gld@columbia.EDU GO Winnipeg Jets GO!!! >
gld@cunixc.BITNET Selanne + Domi ==> Stanley ###
```

We can test one of the prompts:

```
# text to classify
text_content = test['prompt'][0]
#print(text_content)
```

Enter your trained model's name:

```
ft_model="ft:babbage-002:personal::7sYWloYn"
```

Let's construct a prompt:

Copy Explain

```
# Construct the prompt
prompt_text = f"For the following email, please classify by selecting the appropriate
option:\n\n{text_content}\n\nOptions:\n1. hockey\n2. baseball\n\nAnswer:"
```

We now run our model with the constructed prompt:

Copy Explain

```
# Use the completion API to get a prediction
response = openai.Completion.create(
  model=ft_model,
  prompt=prompt_text,
  max_tokens=3  # Limit the response to a few tokens to avoid lengthy outputs
)
print(response.choices[0].text.strip())
```

In this case, the output is correct:hockeyThe model was fine-tuned but the responses may not be reliable each time. In-depth testing should be conducted before deploying a trained model.But was the fine-tuning necessary? Let's go to the next section and see if a pretrained model can do the job without fine-tuning.

4. Further evaluation: Fine-tuning validation

As scientists, we need to verify an assertion. The following cells in the notebook implement a GPT-3 model to perform the classification task. Can a davinci model such as "davinci-002" perform a classification between baseball and hockey correctly with no fine-tuning. We can take a random message from X (former Twitter):

Copy Explain

```
text_content = """Thank you to the
@Canes
 and all you amazing Caniacs that have been so supportive! You guys are some of the
best fans in the NHL without a doubt! Really excited to start this new chapter in my
career with the
@DetroitRedWings
 !!"""
```

Then run it on a standard model

```
# Construct the prompt and run the task
prompt_text = f"For the following text, please classify by selecting the appropriate
option:\n\n{text_content}\n\nOptions:\n1. hockey\n2. baseball\n\nAnswer:"
res = openai.Completion.create(model="davinci-002", prompt=prompt_text, max_tokens=1,
temperature=0, logprobs=2)
res['choices'][0]['text']
```

The output is correct:

```
'hockey'
```

This output doesn't mean you should not fine-tune a model when necessary. It shows that you must examine several approaches before choosing the best path for your project.The fine-tuning method and results in this section are interesting. However, we can't assert that fine-tuning a model is the best project approach. The choice of using an existing model, training a new model, fine-tuning a model, or designing prompts will depend on the goals of a project.OpenAI can take you further by synchronizing the fine-tuning process with Wandb.

Wandb – Weights & Biases

OpenAI can sync our fine-tunes with Wandb's Weight & Biases platform and tools:https://docs.wandb.ai/guides/integrations/openai?utm_source=wandb_docs&utm_medium=code&utm_campaign=OpenAI+APIYou can monitor quite an amount of information on your fine-tunes. You will need a Wandb account and a paid OpenAI plan. This section will get you started with OpenAI and Wandb.

> Note: OpenAI regularly produces new models. Wandb is not always immediately connected to these new models.

Open `Wandb.ipynb` in the chapter directory of the GitHub repository.The first cell installs the latest version of `openai` with `wandb`:

```
try:
  import openai
except:
  !pip install --upgrade openai wandb
  import openai
```

Then we can run the wandb sync function that will retrieve the fine-tuning information gathered during the fine-tuning process:

```
!openai wandb sync
```

We can first sign up on Wandb:https://wandb.ai/homeIn any case, Wandb will ask us if we wish to create an account, use an existing one, or quit:

```
wandb: (1) Create a W&B account
wandb: (2) Use an existing W&B account
wandb: (3) Don't visualize my results
```

wandb: Enter your choice: 2We can also deploy a Weight and Biases (W&B) server locally:

```
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally:
https://wandb.me/wandb-server)
```

If we choose to use an existing account, we will be prompted to enter the API key, and the process will run automatically:wandb: You can find your API key in your browser here: https://wandb.ai/authorize

```
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: denis-rothman76 (rothman). Use `wandb login --relogin`
to force relogin
```

wandb: Tracking run with wandb version 0.15.4We can view the training data locally:Run data is saved locally in /content/wandb/run-20230702_090111-ft-BDMqCXWP0lU9wZWPlZ1Zh5iMWandb provides an intuitive interface to view the project. Click on the project link:View project at https://wandb.ai/rothman/GPT-3The project page will appear with the list of fine-tunes we have made:

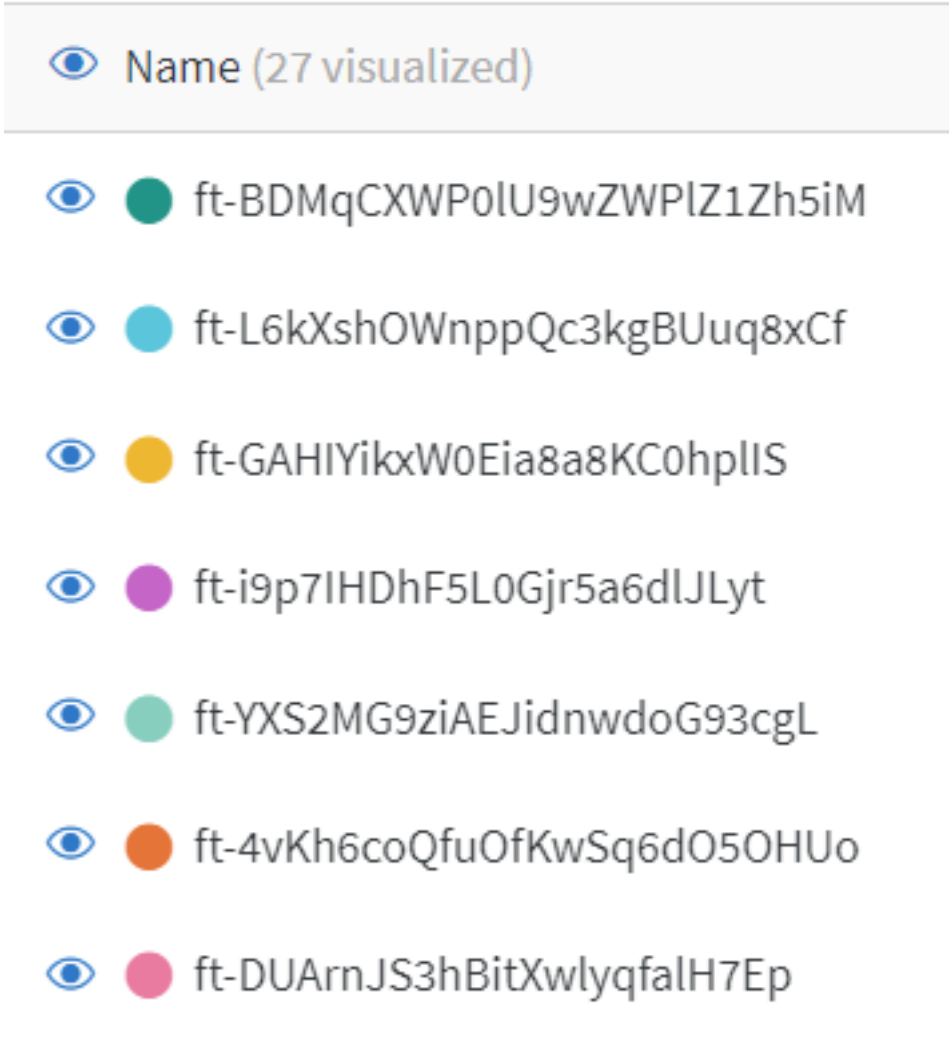

Figure 8.5: A list of fine-tuning tasks

We can analyze the charts for all the models or separately. Charts are available for the metrics described in section *Section 2.1. Measuring the model's Accuracy*, including training_loss:

Figure 8.6: The training_loss for al fine-tuning tasks

We can choose to view the current project (or the last project running on OpenAI):View run at https://wandb.ai/rothman/GPT-3/runs/ft-BDMqCXWP0lU9wZWPlZ1Zh5iMIf we click on the link, we will access detailed information on the training process, such as `training_token_accuracy`:

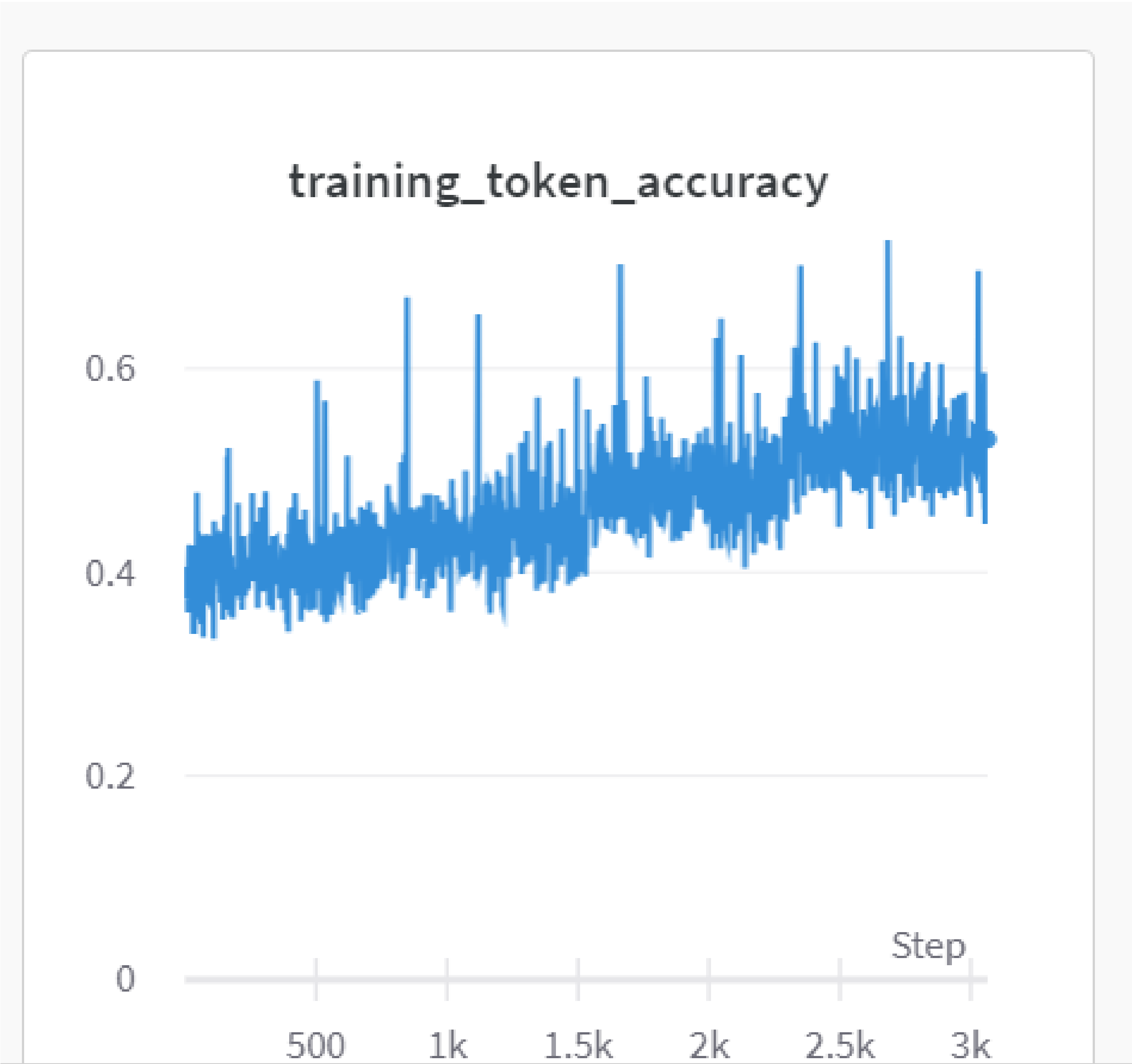

Figure 8.7: The training_token_accuracy graph

Wandb also displays the last (current) run's history:

```
Run history:
          elapsed_examples
            elapsed_tokens
             training_loss
training_sequence_accuracy
   training_token_accuracy

Run summary:
          elapsed_examples 24520.0
            elapsed_tokens 3641288.0
          fine_tuned_model ada:ft-personal-2023...
                    status succeeded
             training_loss 0.49254
training_sequence_accuracy 0.0
   training_token_accuracy 0.50802
```

Figure 8.8: The last run's history and summary

That is not all! There are more options, including an artifacts option. We can download the details of a fine-tune, for example:

```
import wandb
run = wandb.init()
artifact = run.use_artifact('rothman/GPT-3/fine_tune_details:v25',
type='fine_tune_details')
artifact_dir = artifact.download()
```

Then display the data in a Pandas Dataframe:

```
import pandas as pd
import json
# Load JSON file as a Python dictionary
with open('/content/artifacts/fine_tune_details:v25/fine_tune_details.json') as file:
    data = json.load(file)
# Normalize the data and convert it to a Pandas DataFrame
df = pd.json_normalize(data)
# Display the DataFrame
df
```

The output provides details on how the model was fine-tuned, as shown in the following excerpt:

hyperparams.n_epochs	hyperparams.batch_size	hyperparams.prompt_loss_weight
4	8	0.01

Figure 8.9: Hyperparameter information

If we go to the artifact menu online, we can monitor the lineage of the models we are fine-tuning. For example, the Kant dataset was trained several times in different ways (data). We can ask Weights & Biases' Wandb to display the lineage of the processing of a dataset we're working on:

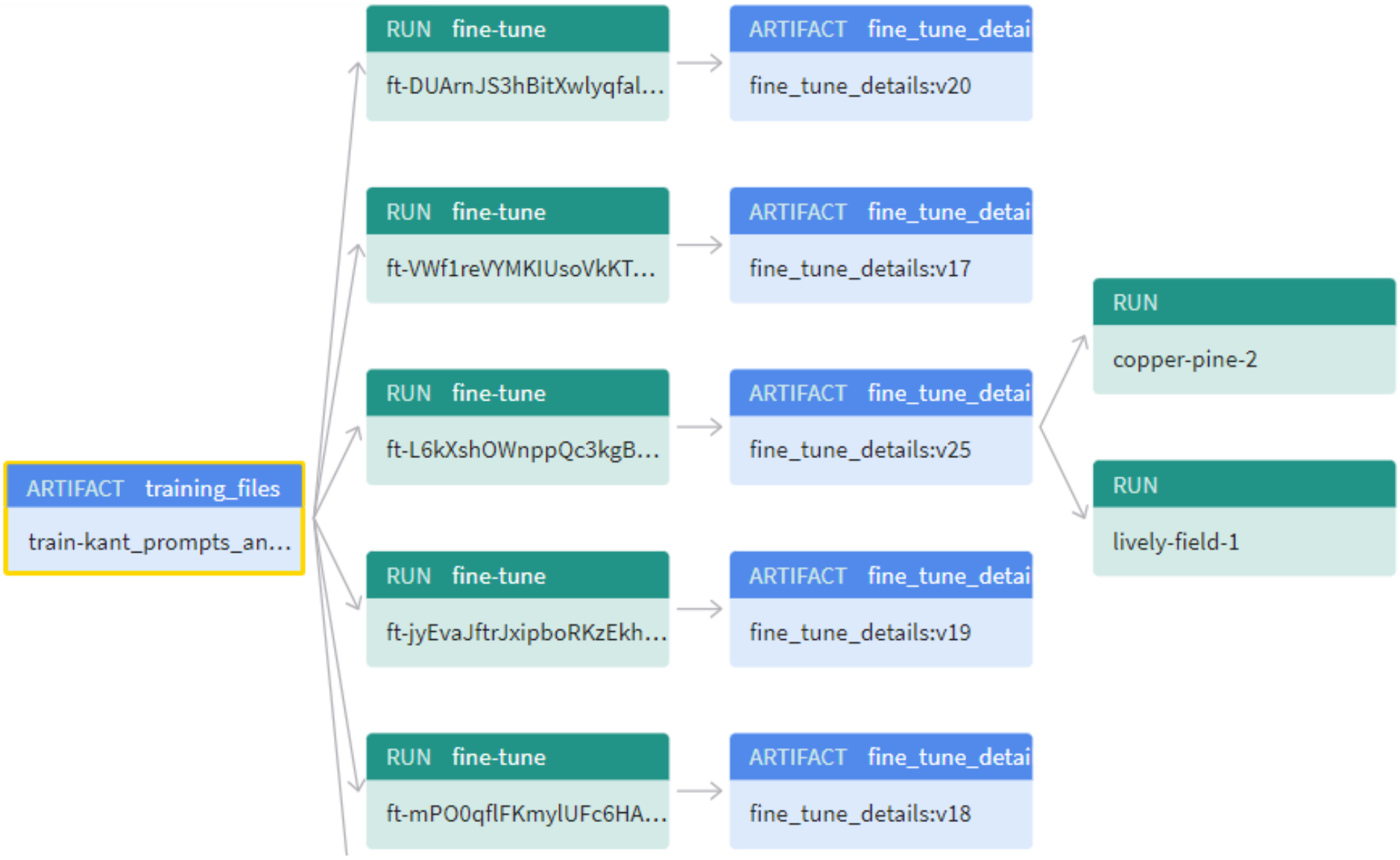

Figure 8.10: The fine-tuning process history

We can then drill down into the fine-tuning data of one of the fine-tunes to analyze the metrics. Datasets can prove very tricky to optimize and may require several fine-tunes to obtain the results we expect.There are many more tools and information available for your project in Wandb. We've completed our fine-tuning path. We can now summarize the chapter and move to a new adventure.

Summary

This chapter led us to the potential of adapting an OpenAI model to our needs through fine-tuning. The process requires careful data analysis and preparation. We must also determine if fine-tuning using OpenAI's platform does not violate our privacy, confidentiality, and security requirements.We first built a fine-tuning process for a completion(generative) task by loading a pre-processed dataset of *Immanuel Kant's Critique of Pure Reason*. We submitted it to OpenAI's data preparation tool. The tool converted our data into JSONL. An ada model was fine-tuned and stored. We then ran the model.Then the `babbage-002` model was fine-tuned for a classification (discriminative) task. This process brought us back to square one: can a standard OpenAI model achieve the same results as a fine-tuned model? If so, why bother fine-tuning a model?To satisfy our scientific curiosity, we ran `davinci` on the same task as the trained `ada` to classify a text to determine if it

was about hockey or baseball. Surprise! The standard model's response was acceptable.This leaves us where we began. Before investing resources to fine-tune, use a standard model or engineer prompts, a full investigation of the risks and opportunities must be made. We, humans, are still here for quite a while, I guess!Our next journey, *Chapter 9, Shattering the Black Box with Interpretable Tools*, will take us deep inside transformer models, to understand how they work and make predictions.

Questions

1. It is useless to fine-tune an OpenAI model. (True/False)
2. Any pretrained OpenAI model can do the task we need without fine-tuning. (True/False)
3. We don't need to prepare a dataset to fine-tune an OpenAI model. (True/False)
4. We don't need one if no datasets are available on the web. (follow-up question for question 3.) (True/False)
5. We don't need to keep track of the fine-tunes we created. (True/False)
6. As of July 2023, anybody can access our fine-tunes. (True/False)
7. Wandb is a state-of-art transformer model. (True/False)
8. Wandb can be synced with OpenAI models. (True/False)
9. Unfortunately, Wandb cannot display accuracy. (True/False)
10. The lineage of the fine-tunes is one of Wandb's artifacts. (True/False)

References

OpenAI fine-tuning documentation: https://platform.openai.com/docs/guides/fine-tuningOpenAI and Wandb guide: https://docs.wandb.ai/guides/integrations/openai?utm_source=wandb_docs&utm_medium=code&utm_campaign=OpenAI+APIWandb, Weights & Biases:https://wandb.ai/home

---

Further Reading

Weights and Biases articles: https://wandb.ai/site/articlesFine-tuning research: *Fine-Tuning Language Models with Just Forward Passes*, Maddadi, et al.(2023), https://arxiv.org/abs/2305.17333

Join our book's Discord space

Join the book's Discord workspace:https://www.packt.link/Transformers