

Learning Objectives

Learners will be able to...

- Define Fine-Tuning
- Generate a response using Fine-Tuned Model
- Run code using Fine-Tuned Model
- Interact with files using OpenAI

info

Make Sure You Know

You are familiar with Python.

Limitations

This is a gentle introduction. So there is a little bit of Python programming. The information might not be the most up to date as OpenAI releases new features.

Customizing GPT-3

One of
key
endpoi
that al
you to
the mc
your G
model
fine-tun
endpoi
definit
fine-tu
means
adjusti
precise
as to b
the hig
level o
perfor
or
effectiv
In othe
words,
adjust
(somet
so that
works
perfect
The fir
tuning
endpoi
allows
custon
the mc
your
particu
use cas
Fe
Learn



OpenAI pre-trained GPT-3 with a prepared data set in a semi-supervised fashion. If given a prompt and a few examples it can most likely guess what task you are trying to perform and generate a completion based on that. This process is called **Few-shot Learning**.

With fine-tuning one is able to customize a model for your application. We can use our own data and create a model custom made for our different projects. Customizing makes GPT-3 more efficient and faster. Matter of fact, it even allows us to use cheaper models more efficiently.

Fine-Tuning

Fine-tuning lets you get more out of the models available through the API by providing:

1. Higher quality results than prompt design
2. Ability to train on more examples than can fit in a prompt
3. Token savings due to shorter prompts
4. Lower latency requests

Fine tuning is all about changing the model so it can generate the responses you want every time. The capabilities and knowledge of the model will be fully focus on the dataset used for fine tuning.

Fine-tuning improves on few-shot learning by training on many more examples than can fit in the prompt, letting you achieve better results on a wide number of tasks. Once a model has been fine-tuned, you won't need to provide examples in the prompt anymore. For example, this how the dataset **json** file would like.

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

Before we get started, it is recommended to install the OpenAI command-line interface. To do this we are going to have to run the following command in the terminal(bottom left).

```
pip install --upgrade openai
```

▼ If you get numpy version error

You can run the following in terminal and try again:

```
pip install numpy --upgrade
```

Run the following in the terminal but this time switching API KEY with your API key.

```
export OPENAI_API_KEY="<OPENAI_API_KEY>"
```

Preparing training

Now that we have installed the necessary tools in the back end we can start training our data. Training data is how you teach GPT-3 what you'd like it to say.

GPT-3 expects your fine-tune dataset to be in a specific JSONL file format:

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

The tool accepts different formats, with the only requirement that they contain a prompt and a completion column/key. Then simply run it through the OpenAI command-line tool to validate it.

You can pass a CSV, TSV, XLSX, JSON or JSONL file, and it will save the output into a JSONL file ready for fine-tuning. JSON Lines (.jsonl) is like JSON, but each line is a valid JSON object. This means that each line is a separate, independent data entry, which can be read, parsed, and processed independently of the others. It will bring up suggesting changes as you proceed.

A sample data, `SampleData.csv`, has been provided for this exercise. It has about 50 examples. When going about it on your own please note it usually recommends a few hundred examples. The sample data is a csv which we can see on the top left. Run the following code, replace `<LOCAL_FILE>` with `SampleData.csv`.

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

When you are done, we can run the following line of code on your terminal so we can peek at our newly generated JSONL file.

```
head -5 SampleData_prepared_valid.jsonl
```

The following assumes you have already prepared training data. The training file goes in the `< >` and the second one is for the `BASE_MODEL` that is the name of the base model you're starting from (ada, babbage, curie, or davinci).

Train a new fine-tuned model, replace train file <TRAIN_FILE_ID_OR_PATH> with `SampleData_prepared.jsonl` and <BASE_MODEL> with `curie`:

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

This often takes a few minutes but can take longer depending on the data and the model.

Running the above command does several things:

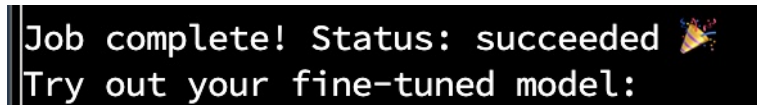
1. Uploads the file using the files API
2. Creates a fine-tune job
3. Streams events until the job is done

If by any chance, the work is interrupted, you can run the following command to get it back on track.

```
openai api fine_tunes.follow -i <YOUR_FINE_TUNE_JOB_ID>
```

the command you need from can be copied the terminal if the stream is interrupted.

When your command is done running you should see the following message.



```
Job complete! Status: succeeded 🚀  
Try out your fine-tuned model:
```

`.guides/img/success`

Please make sure you copy the model name in the response. starting with `Created fine-tune. <model> ft:` In our case should be `curie: ft`
hint: the command you need from can be copied the terminal if the stream is interrupted.

Using our fine-tuned model

Fine-tuning is a powerful technique to create a new model that's specific to your use case. Before fine-tuning your model, we strongly recommend reading these best practices and [specific guidelines](#) for your use case.

After creating our new model after our success message it should generate a message saying try out your model. Insert the fine-tuned model command we copied from the previous page. it should look like the command below. Use "Codio sign: its tuesday ->" for your prompt.

Run it in the terminal:

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

In order to generate output with Python you can try the following:

python3:

```
model= "curie:ft-personal-....",  
prompt="Codio sign: its tuesday ->"
```

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.

Play around with a couple of prompts. Please note that while playing around the AI can generate different answers every run. Additionally, you can play around with some of these as extra arguments like temperature top_p max_tokens.

When ready we can go over deleting our model. In case you no longer want to interact with a model you created you can run the following code to remove them.

Run it in the terminal:

```
openai api models.delete -i <FINE_TUNED_MODEL>
```

```
{  
  "deleted": true,  
  "id": "curie:ft-personal-2023-06-02-10-48-12",  
  "object": "model"  
}
```


A picture demonstrating what message would be printed after deleting the model. It shows deleted=true,id =.... and object=model

python3:

```
import openai
openai.Model.delete(FINE_TUNED_MODEL)
```

Files

Files are used to upload documents that can be used with features like Fine-tuning. Let's go over some basic commands to work with files with the OpenAI API.

Before that we need to identify what file can be uploaded. Each line in the file should be a JSON dictionary with two keys: "prompt" and "completion". The "prompt" key should contain the input text to be used for fine-tuning, and the "completion" key should contain the expected output text.

```
{"prompt": "What is the capital of France?", "completion": "The capital of France is Paris."}
```

Uploading a file

To upload a file to the OpenAI API, use the `create` command with the `file` and `purpose` arguments. To insert the following, click on the `file.py` tab on the left panel. Then copy and paste the following in.

The `file` argument specifies the name of the file as it will be stored in the API. The `purpose` argument specifies the intended use of the file, such as fine-tuning or search.

This code will read `"my_document.txt"` from your local machine and upload it to OpenAI for the purpose of fine-tuning. The response object will contain the server's response, which should include the ID of the newly created file which we need for the next part.

```
with open("my_document.txt", "rb") as f:
    response=openai.File.create(
        file=f,
        purpose="fine-tune"
    )
print(response)
```

List

To list the files that belong to the current user, use the `list` command:

```
files = openai.File.list()
print(files)
```

This will return a list of `File` objects that correspond to the files that belong to the user. Each `File` object has a `filename` and a `purpose` attribute, as well as other metadata like the file size and upload date.

One of the key pieces of data you will interact the most with is the file ids.

Copy and paste the following just to print out the ids of your files.

```
# Retrieve list of files
files = openai.File.list()

# Print all file IDs
for file in files.data:
    print(file.id)
```

Delete

To delete a file from the OpenAI API, use the `delete` method of the `File` object:

Now that you have your list id you can copy and paste the following to delete an file. Make sure to replace `"file-oUBzcM420uW04wDnnIR9uzSN"` with the actual ID of the file you want to delete.

```
# Delete the file by ID
file_id = "file-oUBzcM420uW04wDnnIR9uzSN"
openai.File.delete(file_id)
```

Downloading

To download the contents of a file from the OpenAI API, use the following method of the `File` object:

```

# Step 1: Upload a file
with open("my_document.txt", "rb") as f: # Replace with your
actual file
    response = openai.File.create(
        file=f,
        purpose="fine-tune"
    )

# The file ID is in the response from the API
file_id = response.id
print(f"Uploaded file ID: {file_id}")

# Step 2: Download the file using its ID
download_url =
f"https://api.openai.com/v1/files/{file_id}/content"

headers = {
    "Authorization": f"Bearer {openai.api_key}"
}

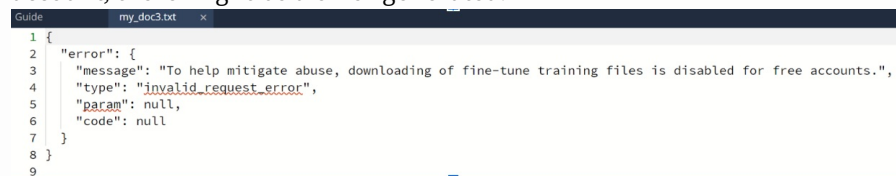
response = requests.get(download_url, headers=headers)

with open("my_doc2.txt", "wb") as f: # The file will be saved
with this name
    f.write(response.content)

print("Downloaded file saved as my_doc2.txt")

```

This will download the contents of the file "my_document.txt" from the API and save them to a local file with the same name. Please note if using a free account, there might be a error generated.



The screenshot shows a code editor with a dark theme. The top bar has tabs labeled 'Guide', 'my_doc3.txt', and 'x'. The code is as follows:

```

1 {
2   "error": {
3     "message": "To help mitigate abuse, downloading of fine-tune training files is disabled for free accounts.",
4     "type": "invalid_request_error",
5     "param": null,
6     "code": null
7   }
8 }
9

```

The error message is displayed in the code editor, indicating that downloading fine-tune training files is disabled for free accounts.