Finetuning_OpenAI

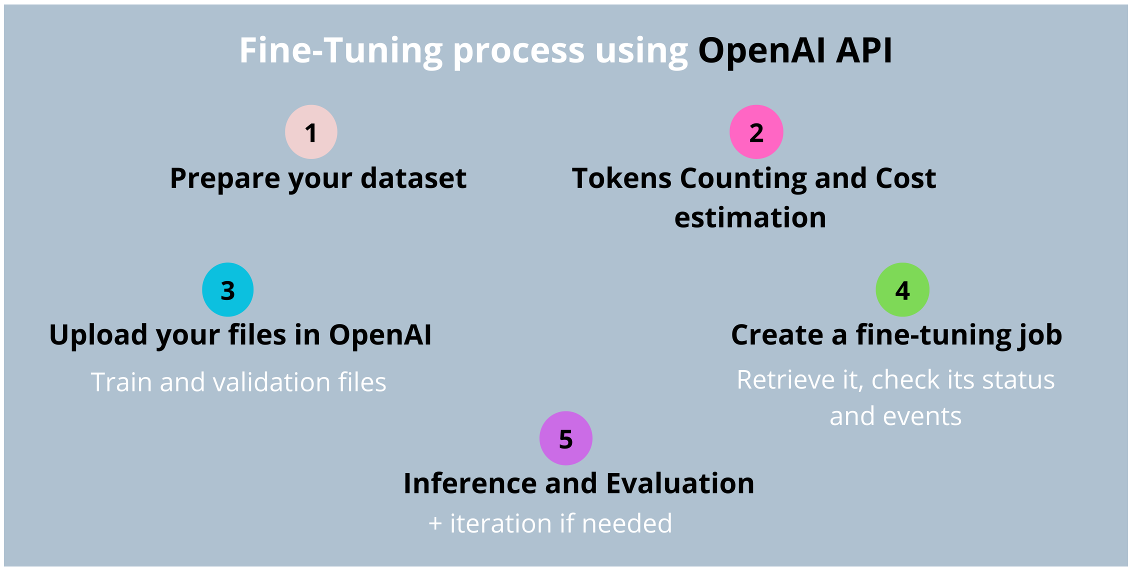
November 25, 2023

1 Install Import libraries

```
[ ]: !pip install openai
     !pip install tiktoken
```

```
[161]: from IPython import display
       display.Image(path_image)
```

[161]:


Fine-Tuning process using OpenAI API
1 Prepare your dataset
2 Tokens Counting and Cost estimation
3 Upload your files in OpenAI
Train and validation files
4 Create a fine-tuning job
Retrieve it, check its status and events
5 Inference and Evaluation
+ iteration if needed

2 Data Preparation for Fine Tuning

We're going to fine-tune GPT-3.5 using an airline tweets dataset to detect sentiments in tweets. This is just an example, to practice and see how the fine-tuning process works

2.1 Loading dataset

2.1.1 AirlineTweets

```
[ ]: !wget -nc https://www.dropbox.com/s/lkd0eklmi64m9xm/AirlineTweets.csv?dl=0
```

```
File ‘AirlineTweets.csv?dl=0’ already there; not retrieving.
```

```
[ ]: import pandas as pd
     df = pd.read_csv('AirlineTweets.csv?dl=0')
     df.head(2)
```

```
[ ]:              tweet_id airline_sentiment  airline_sentiment_confidence  \
     0  570306133677760513           neutral                        1.0000
     1  570301130888122368          positive                        0.3486

       negativereason  negativereason_confidence         airline  \
     0            NaN                        NaN  Virgin America
     1            NaN                        0.0  Virgin America

       airline_sentiment_gold      name negativereason_gold  retweet_count  \
     0                    NaN   cairdin                 NaN              0
     1                    NaN  jnardino                 NaN              0

                                                    text tweet_coord  \
     0               @VirginAmerica What @dhepburn said.         NaN
     1  @VirginAmerica plus you've added commercials t…         NaN

                    tweet_created tweet_location               user_timezone
     0  2015-02-24 11:35:52 -0800            NaN  Eastern Time (US & Canada)
     1  2015-02-24 11:15:59 -0800            NaN  Pacific Time (US & Canada)
```

2.2 Prepare dataset for Fine Tuning:

2.2.1 Create messages[{role :content}]: For role: system, user, assistant

```
[ ]: training_data = []

     system_message = "You are a helpful assistant. You are to extract the sentiment␣
      ↪analysis from the provided airline tweets."

     def create_user_message(row):
         return f"""Airline: {row['airline']}\n\nTweet: {row['text']}\n\nAirline␣
      ↪Sentiment: """

     def create_final_message(row):
         messages = []
```

```
    messages.append({"role": "system", "content": system_message})

    user_message = create_user_message(row)
    messages.append({"role": "user", "content": user_message})

    messages.append({"role": "assistant", "content": row["airline_sentiment"]})

    return {"messages": messages}

create_final_message(df.iloc[0])
```

[ ]:
```
{'messages': [{'role': 'system',
   'content': 'You are a helpful assistant. You are to extract the sentiment
analysis from the provided airline tweets.'},
  {'role': 'user',
   'content': 'Airline: Virgin America\n\nTweet: @VirginAmerica What @dhepburn
said.\n\nAirline Sentiment: '},
  {'role': 'assistant', 'content': 'neutral'}]}
```

2.2.2 Split for Training and validation sets

[ ]:
```
print(f"df shape {df.shape}")

training_df = df.loc[0:600]
training_data = training_df.apply(create_final_message, axis=1).tolist()

for example in training_data[:3]:
    print(example)
```

```
df shape (14640, 15)
{'messages': [{'role': 'system', 'content': 'You are a helpful assistant. You
are to extract the sentiment analysis from the provided airline tweets.'},
{'role': 'user', 'content': 'Airline: Virgin America\n\nTweet: @VirginAmerica
What @dhepburn said.\n\nAirline Sentiment: '}, {'role': 'assistant', 'content':
'neutral'}]}
{'messages': [{'role': 'system', 'content': 'You are a helpful assistant. You
are to extract the sentiment analysis from the provided airline tweets.'},
{'role': 'user', 'content': "Airline: Virgin America\n\nTweet: @VirginAmerica
plus you've added commercials to the experience… tacky.\n\nAirline Sentiment:
"}, {'role': 'assistant', 'content': 'positive'}]}
{'messages': [{'role': 'system', 'content': 'You are a helpful assistant. You
are to extract the sentiment analysis from the provided airline tweets.'},
{'role': 'user', 'content': "Airline: Virgin America\n\nTweet: @VirginAmerica I
didn't today… Must mean I need to take another trip!\n\nAirline Sentiment: "},
{'role': 'assistant', 'content': 'neutral'}]}
```

[ ]:
```
print(example['messages'][1]['content'])
```

```
Airline: Virgin America

Tweet: @VirginAmerica and it's a really big bad thing about it

Airline Sentiment:
```

[ ]:
```
validation_df = df.loc[600:800]
validation_data = validation_df.apply(create_final_message, axis=1).tolist()
```

2.2.3 Convert to jsonL

[ ]:
```
import json
import numpy as np
```

[ ]:
```
def write_jsonl(data_list: list, filename: str) -> None:
    with open(filename, "w") as out:
        for ddict in data_list:
            jout = json.dumps(ddict) + "\n"
            out.write(jout)
```

[ ]:
```
training_file_name = "airline_tweets_training.jsonl"
write_jsonl(training_data, training_file_name)

validation_file_name = "airline_tweets_validation.jsonl"
write_jsonl(validation_data, validation_file_name)
```

[ ]:
```
!head -n 2 airline_tweets_training.jsonl
```

```
{"messages": [{"role": "system", "content": "You are a helpful assistant. You
are to extract the sentiment analysis from the provided airline tweets."},
{"role": "user", "content": "Airline: Virgin America\n\nTweet: @VirginAmerica
What @dhepburn said.\n\nAirline Sentiment: "}, {"role": "assistant", "content":
"neutral"}]}
{"messages": [{"role": "system", "content": "You are a helpful assistant. You
are to extract the sentiment analysis from the provided airline tweets."},
{"role": "user", "content": "Airline: Virgin America\n\nTweet: @VirginAmerica
plus you've added commercials to the experience… tacky.\n\nAirline Sentiment:
"}, {"role": "assistant", "content": "positive"}]}
```

2.3 Tokens count and cost estimation

[ ]:
```
# Load the jsonl dataset
with open(training_file_name, 'r', encoding='utf-8') as f:
    dataset = [json.loads(line) for line in f]

# Initial dataset stats
print("Num examples:", len(dataset))
print("First example:")
for message in dataset[500]["messages"]:
    print(message)
```

```
Num examples: 601
First example:
{'role': 'system', 'content': 'You are a helpful assistant. You are to extract
the sentiment analysis from the provided airline tweets.'}
{'role': 'user', 'content': 'Airline: Virgin America\n\nTweet: @VirginAmerica
still waiting to see  @Starryeyes_Dev_  \n\nAirline Sentiment: '}
{'role': 'assistant', 'content': 'neutral'}
```

Tokens count

```
[ ]: import tiktoken
```

Different encodings are used in openai: **cl100k_base**, p50k_base, gpt2.

These encodings depend on the model you are using:

For gpt-4, **gpt-3.5-turbo**, text-embedding-ada-002, you need to use **cl100k_base**.

```
[ ]: encoding = tiktoken.get_encoding("cl100k_base")

     def num_tokens_from_messages(messages, tokens_per_message=3, tokens_per_name=1):
         num_tokens = 0
         for message in messages:
             num_tokens += tokens_per_message
             for key, value in message.items():
                 num_tokens += len(encoding.encode(value))
                 # print(f'key={key}, value={value}, nbr_tokens={len(encoding.
       ↪encode(value))}, cum_tokens={num_tokens}')
                 if key == "name":
                     num_tokens += tokens_per_name
         num_tokens += 3
         return num_tokens

     def num_assistant_tokens_from_messages(messages):
         num_tokens = 0
         for message in messages:
             if message["role"] == "assistant":
                 num_tokens += len(encoding.encode(message["content"]))
         return num_tokens
```

```
[ ]: num_missing_system = 0
     num_missing_user = 0
     num_messages = []
     message_all_lens = []
     assistant_message_lens = []
```

```
for ex in dataset:
    messages = ex["messages"]
    if not any(message["role"] == "system" for message in messages):
        num_missing_system += 1
    if not any(message["role"] == "user" for message in messages):
        num_missing_user += 1
    num_messages.append(len(messages))
    message_all_lens.append(num_tokens_from_messages(messages))
    assistant_message_lens.append(num_assistant_tokens_from_messages(messages))

print("Num messages missing system message:", num_missing_system)
print("Num messages missing user message:", num_missing_user)
print("\nSome statistics:")
print(f"min={min(message_all_lens)}, max={max(message_all_lens)}")
print(f"p5={np.quantile(message_all_lens, 0.1)}, p95={np.
 ↪quantile(message_all_lens, 0.95)}")

print("\Examples of tokens nbr from the first messages")
print(message_all_lens[:5])

num_sample_token_long = sum(l > 4096 for l in message_all_lens)

if num_sample_token_long>0:
  print(f"\n{num_sample_token_long} messages may be over the 4096 token limit,␣
 ↪they will be truncated during fine-tuning")
else:
  print(f"\nNo message will be truncated during fine-tuning")
```

```
Num messages missing system message: 0
Num messages missing user message: 0

Some statistics:
min=53, max=121
p5=61.0, p95=89.0
\Examples of tokens nbr from the first messages
[60, 64, 66, 77, 63]

No message will be truncated during fine-tuning
```

Fine-tuning Cost estimation

[ ]:
```
# Epochs: Number of time the model is going through the training dataset
# For production, you may need to put a higher value for epochs
```

[ ]:
```
MAX_TOKENS_PER_EXAMPLE = 4096
```

```
TARGET_EPOCHS = 3
MIN_TARGET_EXAMPLES = 100
MAX_TARGET_EXAMPLES = 25000
MIN_DEFAULT_EPOCHS = 1
MAX_DEFAULT_EPOCHS = 25

n_epochs = TARGET_EPOCHS
n_train_examples = len(dataset)
if n_train_examples * TARGET_EPOCHS < MIN_TARGET_EXAMPLES:
    n_epochs = min(MAX_DEFAULT_EPOCHS, MIN_TARGET_EXAMPLES // n_train_examples)
elif n_train_examples * TARGET_EPOCHS > MAX_TARGET_EXAMPLES:
    n_epochs = max(MIN_DEFAULT_EPOCHS, MAX_TARGET_EXAMPLES // n_train_examples)

n_tokens_in_dataset = sum(min(MAX_TOKENS_PER_EXAMPLE, length) for length in␣
  ↪message_all_lens)
print(f"Dataset contains ~{n_tokens_in_dataset} tokens, which will incur␣
  ↪training charges")
print(f"By default, you'll train for {n_epochs} epochs on this dataset")
print(f"By default, you'll be charged for ~{n_epochs * n_tokens_in_dataset}␣
  ↪tokens")
print(f"The final cost will be ~{n_epochs * n_tokens_in_dataset/1000  * 0.
  ↪008}$")
```

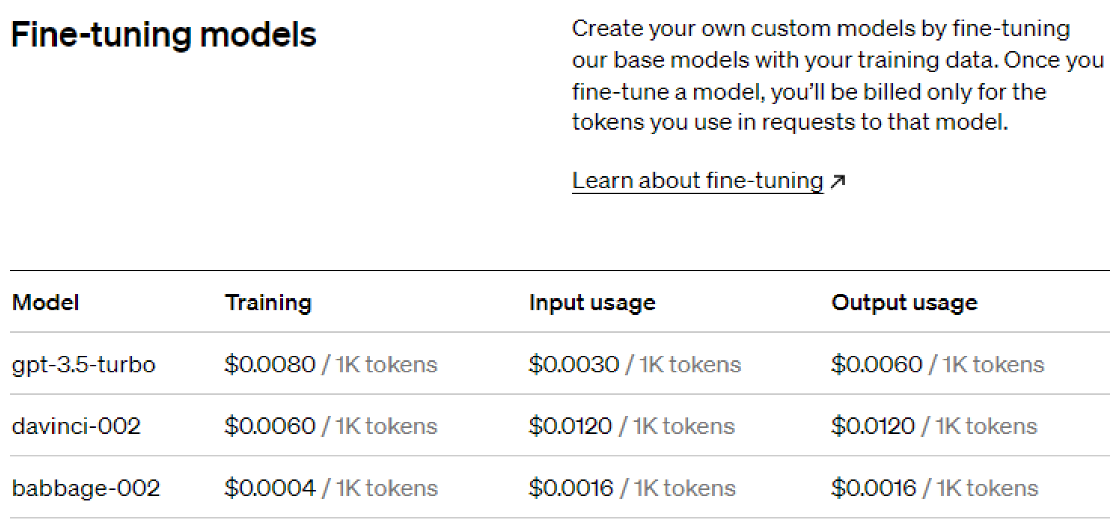
```
Dataset contains ~44955 tokens, which will incur training charges
By default, you'll train for 3 epochs on this dataset
By default, you'll be charged for ~134865 tokens
The final cost will be ~1.07892$
```

Pricing from OpenAI

```
[ ]: from IPython import display
     display.Image(path_image)
```

```
Mounted at /content/drive
```

```
[ ]:
```



Fine-tuning models

Create your own custom models by fine-tuning our base models with your training data. Once you fine-tune a model, you'll be billed only for the tokens you use in requests to that model.

Learn about fine-tuning ↗

Model	Training	Input usage	Output usage
gpt-3.5-turbo	$0.0080 / 1K tokens	$0.0030 / 1K tokens	$0.0060 / 1K tokens
davinci-002	$0.0060 / 1K tokens	$0.0120 / 1K tokens	$0.0120 / 1K tokens
babbage-002	$0.0004 / 1K tokens	$0.0016 / 1K tokens	$0.0016 / 1K tokens

3 Fine Tuning process

```
[ ]: from google.colab import userdata
     openai_api_key = userdata.get('OPENAI_API_KEY')
```

```
[ ]: from openai import OpenAI
     client = OpenAI(api_key=openai_api_key)
```

3.1 Upload files in OpenAI

```
[ ]: training_file_creation = client.files.create(
       file=open(training_file_name, "rb"),
       purpose='fine-tune'
     )
```

```
[ ]: training_file_creation
```

```
[ ]: FileObject(id='file-513TXhzAGIypvaUt2RqKLW39', bytes=234597,
     created_at=1700860036, filename='airline_tweets_training.jsonl', object='file',
     purpose='fine-tune', status='processed', status_details=None)
```

```
[ ]: id_file_training = training_file_creation.id
```

```
[ ]: validation_file_creation = client.files.create(
       file=open(validation_file_name, "rb"),
       purpose='fine-tune'
     )
```

```
[ ]: validation_file_creation
```

```
[ ]: FileObject(id='file-aOQYL6VpOFbu8QdvOP5Q06ZI', bytes=77406,
     created_at=1700860056, filename='airline_tweets_validation.jsonl',
     object='file', purpose='fine-tune', status='processed', status_details=None)
```

```
[ ]: id_file_validation = validation_file_creation.id
```

```
[ ]: ## To delete files
     # client.files.delete(validation_file_creation.id)
```

3.2 Create Fine Tuning Job

```
[ ]: fine_tuning_job = client.fine_tuning.jobs.create(training_file=id_file_training,
                                 model="gpt-3.5-turbo",
                                 suffix='airline_sentiment',
                                 validation_file=id_file_validation)
```

3.2.1 Get job specification: id, model name, status

```
[ ]: fine_tuning_job
     # status at the beginning ==> Validating_files
     # The current status of the fine-tuning job, which can be either␣
      ↪validating_files, queued, running, succeeded, failed, or cancelled.
```

```
[ ]: FineTuningJob(id='ftjob-0Wg7WZaUPIEnFFOJgBLTS5PO', created_at=1700860459,
     error=None, fine_tuned_model=None, finished_at=None,
     hyperparameters=Hyperparameters(n_epochs='auto', batch_size='auto',
     learning_rate_multiplier='auto'), model='gpt-3.5-turbo-0613',
     object='fine_tuning.job', organization_id='org-PKDk6D4mPARkEfXOj2JB21sK',
     result_files=[], status='validating_files', trained_tokens=None,
     training_file='file-513TXhzAGIypvaUt2RqKLW39', validation_file='file-
     aOQYL6VpOFbu8QdvOP5Q06ZI')
```

```
[ ]: job_id = fine_tuning_job.id
     job_id
```

```
[ ]: 'ftjob-0Wg7WZaUPIEnFFOJgBLTS5PO'
```

```
[ ]: # Fine tuning process could take time ==> you will receive an email once␣
      ↪fine-tuning is finished
```

3.3 List fine-tuned jobs

To get the list of all your jobs (running or finished):

```
[ ]: client.fine_tuning.jobs.list()#.data[0]
```

```
[ ]: SyncCursorPage[FineTuningJob](data=[FineTuningJob(id='ftjob-
     OWg7WZaUPIEnFFOJgBLTS5PO', created_at=1700860459, error=None,
     fine_tuned_model='ft:gpt-3.5-turbo-0613:personal:airline-sentiment:8OYMFXDs',
     finished_at=1700862634, hyperparameters=Hyperparameters(n_epochs=3,
     batch_size=1, learning_rate_multiplier=2), model='gpt-3.5-turbo-0613',
     object='fine_tuning.job', organization_id='org-PKDk6D4mPARkEfXOj2JB21sK',
     result_files=['file-anvVhIrPGl9TEOLKWGOMFsIK'], status='succeeded',
     trained_tokens=131259, training_file='file-513TXhzAGIypvaUt2RqKLW39',
     validation_file='file-aOQYL6VpOFbu8QdvOP5Q06ZI'), FineTuningJob(id='ftjob-
     sGIus9vadEOTVEgSRPuu0sWZ', created_at=1697907644, error=None,
     fine_tuned_model='ft:gpt-3.5-turbo-0613:personal:ner-recipe:8C9ptbha',
     finished_at=1697908436, hyperparameters=Hyperparameters(n_epochs=3,
     batch_size=1, learning_rate_multiplier=2), model='gpt-3.5-turbo-0613',
     object='fine_tuning.job', organization_id='org-PKDk6D4mPARkEfXOj2JB21sK',
     result_files=['file-pBZUwlvYMQ5mbpFftjKSy76k'], status='succeeded',
     trained_tokens=40047, training_file='file-tBa8aMquvTNAqTHYJfzJBYMl',
     validation_file='file-PeCozuyv5cIshHhBoqACv3s3')], object='list',
     has_more=False)
```

```
[ ]: # client.fine_tuning.jobs.list() ==> list of all your fine-tuning jobs
     # ==> id
     # ==> fine_tuned_model ==> the name of your finetuned model with the suffix you␣
      ↪specified in your call
     # ==> model ==> the last final model available in OpenAI ==> 'gpt-3.
      ↪5-turbo-0613'
     # ==> status ==> "suceeded" when it has finished correctly
```

```
[ ]: client.fine_tuning.jobs.list().data[0]
     # job_id = 'ftjob-OWg7WZaUPIEnFFOJgBLTS5PO'
     # job_id = client.fine_tuning.jobs.list().data[0].id
```

```
[ ]: FineTuningJob(id='ftjob-OWg7WZaUPIEnFFOJgBLTS5PO', created_at=1700860459,
     error=None, fine_tuned_model='ft:gpt-3.5-turbo-0613:personal:airline-
     sentiment:8OYMFXDs', finished_at=1700862634,
     hyperparameters=Hyperparameters(n_epochs=3, batch_size=1,
     learning_rate_multiplier=2), model='gpt-3.5-turbo-0613',
     object='fine_tuning.job', organization_id='org-PKDk6D4mPARkEfXOj2JB21sK',
     result_files=['file-anvVhIrPGl9TEOLKWGOMFsIK'], status='succeeded',
     trained_tokens=131259, training_file='file-513TXhzAGIypvaUt2RqKLW39',
     validation_file='file-aOQYL6VpOFbu8QdvOP5Q06ZI')
```

3.4 Cancel a fine-tuned job

```
[ ]: # client.fine_tuning.jobs.cancel(fine_tuning_job_id = job_id)
```

3.5 Retrieve Fine-tuned job

To retrieve a given job by its id: Check the “status” : running, succeeded

```
[ ]: retrieve_fine_tuned_job = client.fine_tuning.jobs.retrieve(job_id)
     retrieve_fine_tuned_job
```

```
[ ]: FineTuningJob(id='ftjob-0Wg7WZaUPIEnFFOJgBLTS5PO', created_at=1700860459,
     error=None, fine_tuned_model='ft:gpt-3.5-turbo-0613:personal:airline-
     sentiment:8OYMFXDs', finished_at=1700862634,
     hyperparameters=Hyperparameters(n_epochs=3, batch_size=1,
     learning_rate_multiplier=2), model='gpt-3.5-turbo-0613',
     object='fine_tuning.job', organization_id='org-PKDk6D4mPARkEfXOj2JB21sK',
     result_files=['file-anvVhIrPGl9TEOLKWGOMFsIK'], status='succeeded',
     trained_tokens=131259, training_file='file-513TXhzAGIypvaUt2RqKLW39',
     validation_file='file-aOQYL6VpOFbu8QdvOP5Q06ZI')
```

```
[ ]: retrieve_fine_tuned_job
```

```
[ ]: FineTuningJob(id='ftjob-0Wg7WZaUPIEnFFOJgBLTS5PO', created_at=1700860459,
     error=None, fine_tuned_model=None, finished_at=None,
     hyperparameters=Hyperparameters(n_epochs=3, batch_size=1,
     learning_rate_multiplier=2), model='gpt-3.5-turbo-0613',
     object='fine_tuning.job', organization_id='org-PKDk6D4mPARkEfXOj2JB21sK',
     result_files=[], status='running', trained_tokens=None,
     training_file='file-513TXhzAGIypvaUt2RqKLW39', validation_file='file-
     aOQYL6VpOFbu8QdvOP5Q06ZI')
```

3.6 List events for fine-tuned job

```
[ ]: events_list_job = client.fine_tuning.jobs.
       ↪list_events(fine_tuning_job_id=job_id, limit=50)
     events = events_list_job.data
     events.reverse()

     for event in events:
       print(event.message)
```

```
Created fine-tuning job: ftjob-0Wg7WZaUPIEnFFOJgBLTS5PO
Validating training file: file-513TXhzAGIypvaUt2RqKLW39 and validation file:
file-aOQYL6VpOFbu8QdvOP5Q06ZI
Files validated, moving job to queued state
Fine-tuning job started
Step 1/1803: training loss=3.85, validation loss=3.45
Step 101/1803: training loss=1.30, validation loss=5.98
Step 201/1803: training loss=0.00, validation loss=3.96
Step 301/1803: training loss=0.00, validation loss=3.28
Step 401/1803: training loss=0.00, validation loss=0.00
Step 501/1803: training loss=6.93, validation loss=0.00
Step 601/1803: training loss=0.00, validation loss=0.00
Step 701/1803: training loss=0.00, validation loss=0.00
Step 801/1803: training loss=0.00, validation loss=0.00
Step 901/1803: training loss=0.00, validation loss=0.00
Step 1001/1803: training loss=5.63, validation loss=0.00
Step 1101/1803: training loss=0.00, validation loss=0.00
Step 1201/1803: training loss=0.00, validation loss=0.56
Step 1301/1803: training loss=0.00, validation loss=0.00
Step 1401/1803: training loss=0.00, validation loss=0.00
Step 1501/1803: training loss=0.00, validation loss=6.16
Step 1601/1803: training loss=0.20, validation loss=0.00
Step 1701/1803: training loss=0.00, validation loss=0.00
Step 1801/1803: training loss=0.00, validation loss=0.00
New fine-tuned model created: ft:gpt-3.5-turbo-0613:personal:airline-
sentiment:8OYMFXDs
The job has successfully completed
```

4 Inference

4.1 Model name once fine-tuned is finished

[ ]:
```
retrieve_fine_tuned_job = client.fine_tuning.jobs.retrieve(job_id)
fine_tuned_model_id = retrieve_fine_tuned_job.fine_tuned_model

if fine_tuned_model_id is None:
    raise RuntimeError("Fine-tuned model ID not found. Your job has likely not␣
 ↪been completed yet.")

print("Fine-tuned model ID:", fine_tuned_model_id)
```

```
Fine-tuned model ID: ft:gpt-3.5-turbo-0613:personal:airline-sentiment:8OYMFXDs
```

4.2 Inference: Using ChatCompletion with the fine-tuned model

[ ]:
```
df.shape
```

[ ]:
```
(14640, 15)
```

4.2.1 Using chat completion for a given tweet example

```
[ ]: test_df = df.loc[800:1000]
     test_row = test_df.iloc[10]
     test_messages = []
     test_messages.append({"role": "system", "content": system_message})
     user_message = create_user_message(test_row)
     test_messages.append({"role": "user", "content": create_user_message(test_row)})

     print(test_messages)
```

```
[{'role': 'system', 'content': 'You are a helpful assistant. You are to extract
the sentiment analysis from the provided airline tweets.'}, {'role': 'user',
'content': 'Airline: United\n\nTweet: @united Your website deserves a new
design. #html5 FTW!\n\nAirline Sentiment: '}]
```

```
[ ]: response = client.chat.completions.create(
         model=fine_tuned_model_id,
         messages=test_messages
     )

     print(f"Tweet = {test_row['text']}")
     print(f"\nrole: {response.choices[0].message.role}, content : {response.
      ↪choices[0].message.content}")
```

```
Tweet = @united Your website deserves a new design. #html5 FTW!

role: assistant, content : negative
```

4.2.2 Using chat completion for the test dataset tweets

```
[ ]: def create_final_message_test(row):
       test_messages = []
       test_messages.append({"role": "system", "content": system_message})

       user_message = create_user_message(row)
       test_messages.append({"role": "user", "content": create_user_message(row)})

       return test_messages
```

```
[ ]: %%time
     finetuned_model_resp =[]
     for i in range(0, len(test_df)):
       row = test_df.iloc[i]
       messages_test = create_final_message_test(row)

       response = client.chat.completions.create(
         model=fine_tuned_model_id,
         messages=messages_test
       )

       # print(f"\nTweet = {row['text']}")
       # print(f"role: {response.choices[0].message.role}, content : {response.
     ↪choices[0].message.content}")
       # print(f"Tweet = {row['airline_sentiment']}")
       finetuned_model_resp.append(response.choices[0].message.content)
```

[ ]:
```
test_df['result'] = finetuned_model_resp
```

```
<ipython-input-113-059c20cd175a>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['result'] = finetuned_model_resp
```

[ ]:
```
print(f"different classes in the y_true {test_df['airline_sentiment'].
 ↪unique()}")
print(f"different classes in the y_pred {test_df['result'].unique()}")
```

```
different classes in the y_true ['negative' 'neutral' 'positive']
different classes in the y_pred ['negative' 'neutral' 'positive']
```

[ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

y_true = test_df['airline_sentiment'].values
accuracy = accuracy_score(y_true, finetuned_model_resp)
precision = precision_score(y_true, finetuned_model_resp, average=None)
recall = recall_score(y_true, finetuned_model_resp,average=None)
f1 = f1_score(y_true, finetuned_model_resp,average=None)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
Accuracy: 0.8557213930348259
Precision: [0.93382353 0.64864865 0.75      ]
Recall: [0.91366906 0.66666667 0.80769231]
F1 Score: [0.92363636 0.65753425 0.77777778]
```