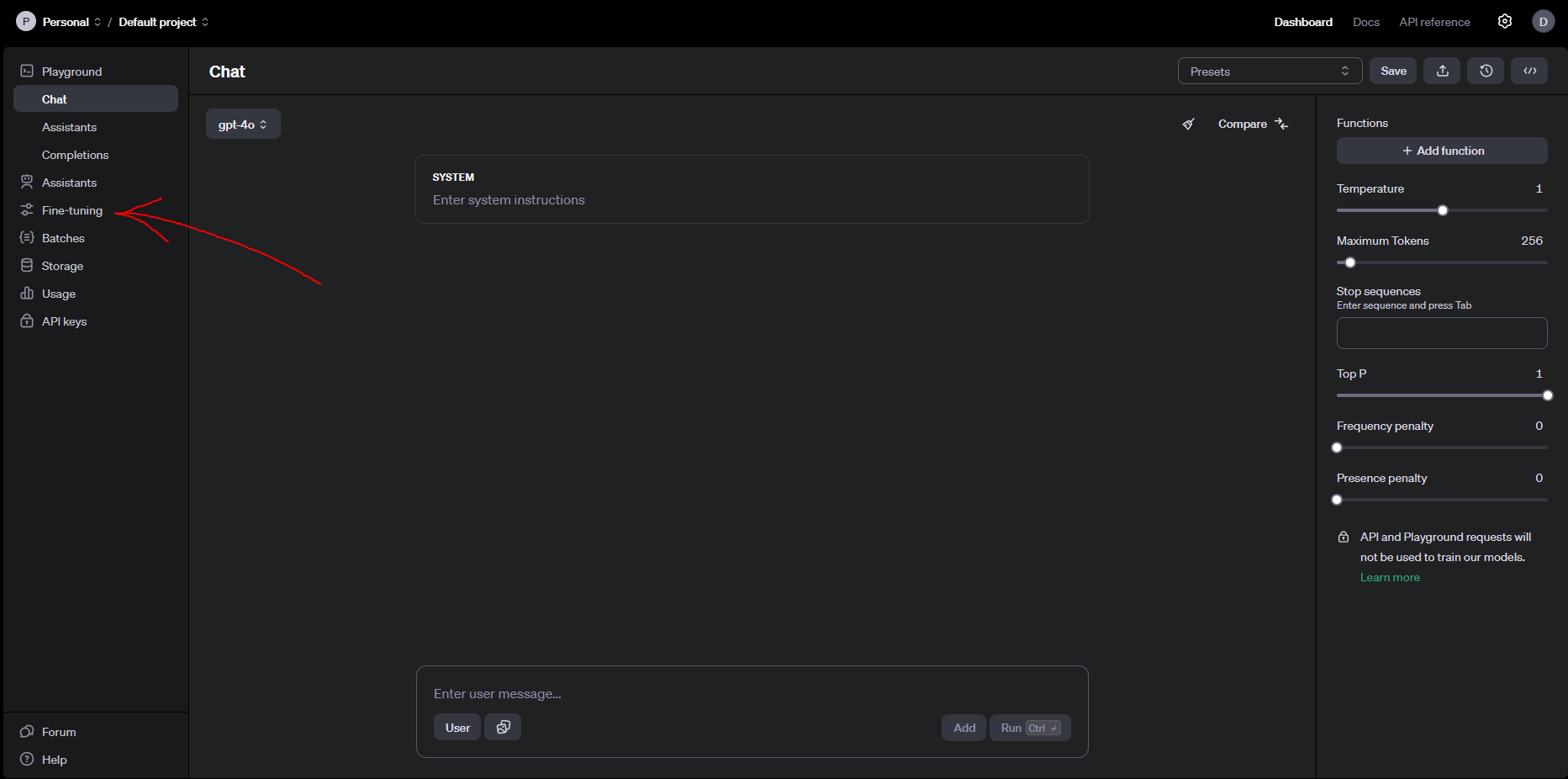
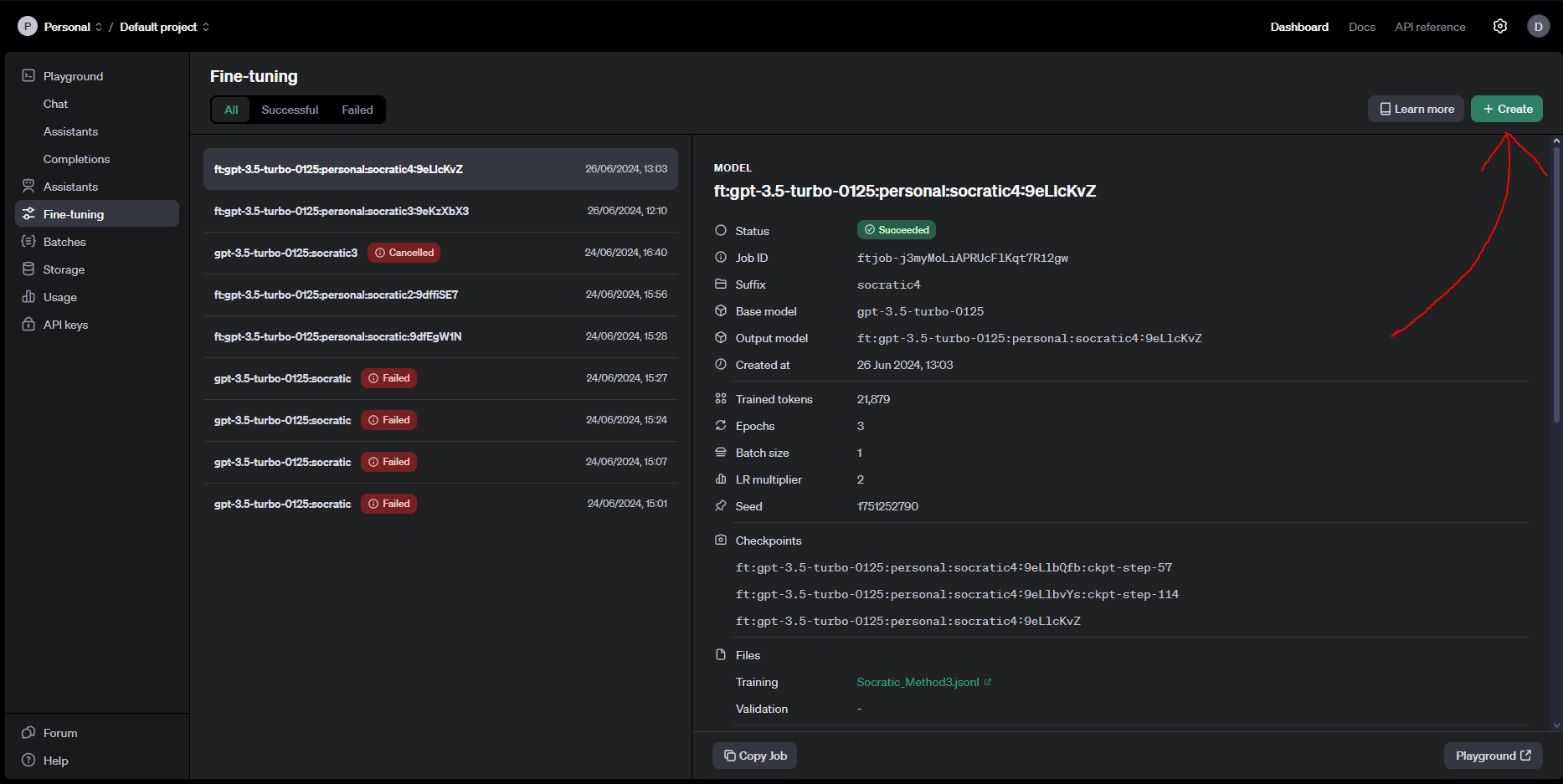
The purpose of fine tuning is to create a new model that mimics a type of behaviour we want, fine tuning does not control what the model knows but instead controls how the model acts.

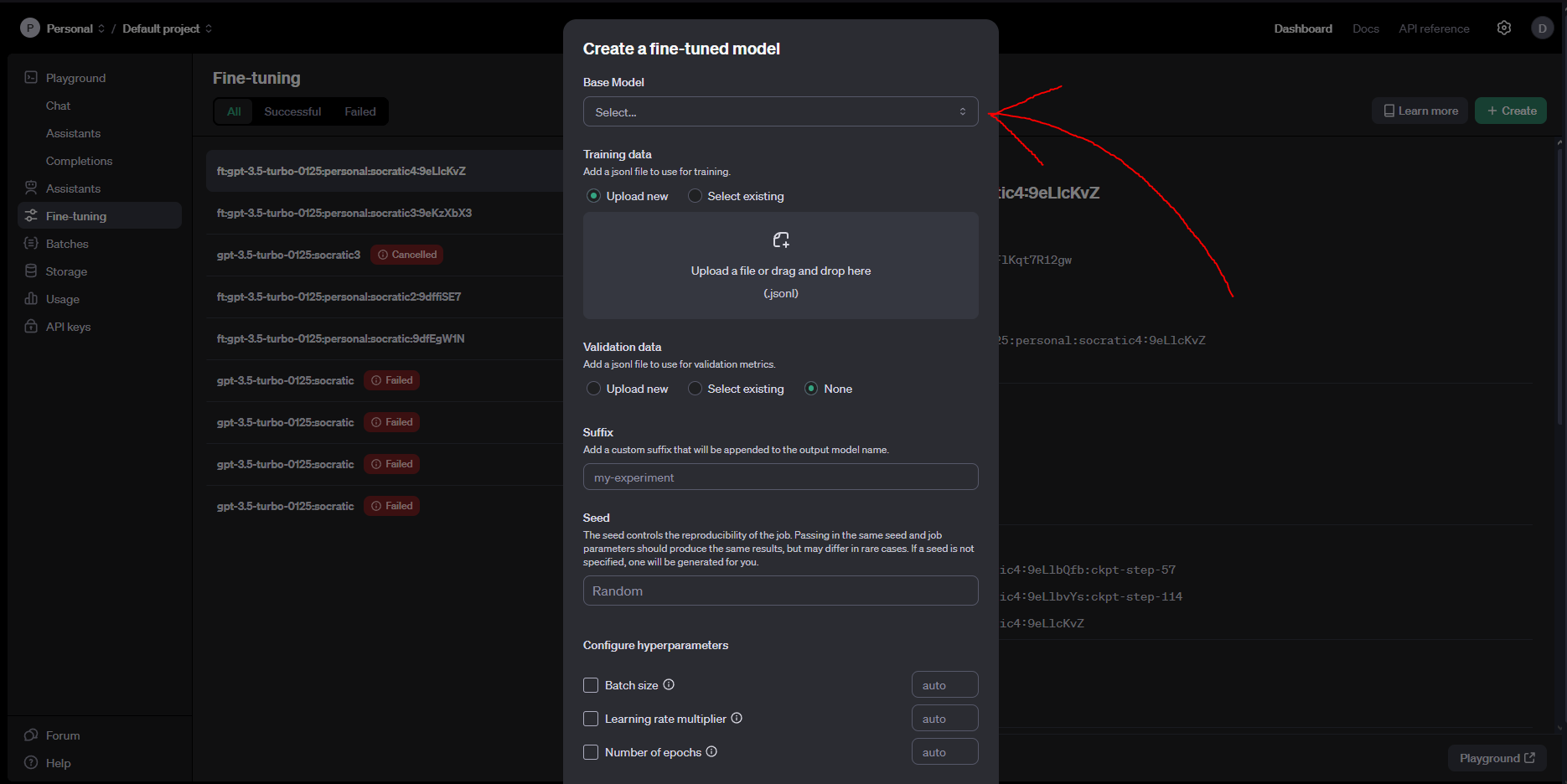
To create a fine tuning model first head to the [OpenAI Dashboard](https://platform.openai.com/playground/chat?models=gpt-4o). On the left hand menu click the fine tuning button:



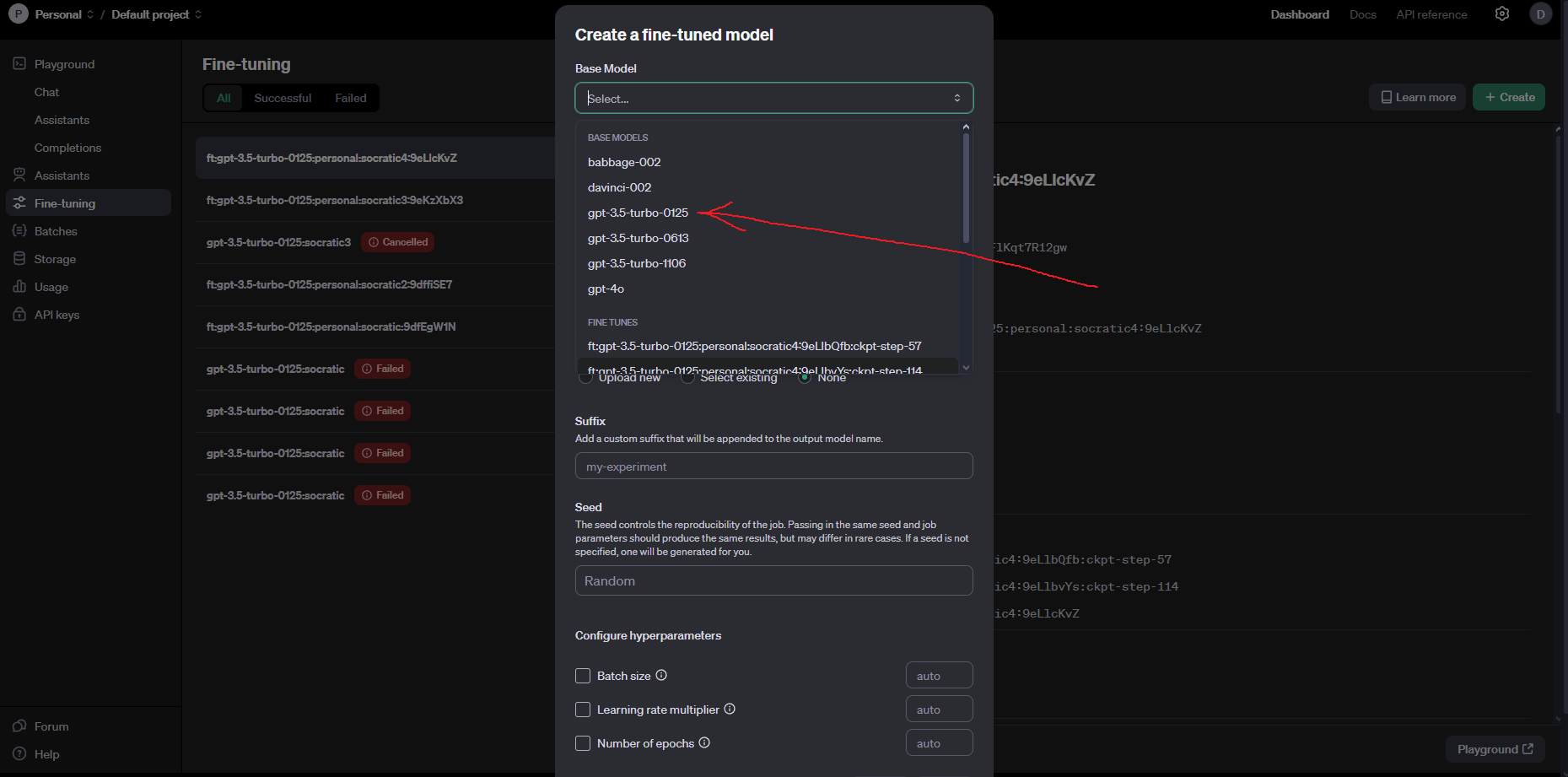
Once here click on the +create button in the top right:



Once you’ve done this press the select button for the base model:



You can then select one of the models on which our new fine tuned model is based, we’ll select one of the gpt models as these are text based:



Once we have selected a model we need to create some training data, training data for text based fine tuning is just example conversations. Our final fine tuned model will try to mimic the style of conversations we give it within the training data with the users it interacts with. I’ll go over now how to create training data in the format we want now. First you want to define a simple instruction for your model, this should sum up the purpose of your model, I’m going to use the phrase:  
  
"You are a factual chatbot that helps students learn by asking them questions"

because I want to create a model that tries to ask students questions instead of giving them the answers. Then I’ll make an example conversations between an assistant and a user:

User: "What is the area of a rectangle with a length of 5 and a width of 3?"

Assistant: "How do you calculate the area of a rectangle?"

User: "You multiply the length by the width."

Assistant: "That's correct. So, what is 5 multiplied by 3?"

User: "5 multiplied by 3 is 15."

Assistant: "Great! So, what is the area of the rectangle?"

User: "The area is 15."

Assistant: "Exactly. Well done!"

Now we have this we need to put it into a proper format so that OpenAI can read it.

First we start by writing:

{“messages”:}

This lets OpenAI know that what is coming next is a series of messages. We then create a list

{“messages”: [] }

This list is where we’ll put our conversation. To start every conversation we write we start with a message from the system, this message from the system is our instruction to the model that we wrote earlier. First we define who the message is coming from using “role” followed by the role of who the message is coming from:

{“messages”: [{“role”: “system”}]}

Then we add what the system is saying by writing “content” followed by the content of the message:

{“messages”: [{“role”: “system”, “content”: "You are a factual chatbot that helps students learn by asking them questions”}]}

Now we can add in the messages from our conversation, again we define who the message is from and what they are sending:

{“messages”: [{“role”: “system”, “content”: "You are a factual chatbot that helps students learn by asking them questions”}, {"role": "user", "content": "What is the area of a rectangle with a length of 5 and a width of 3?"} ]}

We continue on like this until we end up with something that looks like the below, making sure that each message is contained in {} and that a comma separates each message:

{"messages": [{"role": "system", "content": "You are a factual chatbot that helps students learn by asking them questions"}, {"role": "user", "content": "What is the area of a rectangle with a length of 5 and a width of 3?"}, {"role": "assistant", "content": "How do you calculate the area of a rectangle?"}, {"role": "user", "content": "You multiply the length by the width."}, {"role": "assistant", "content": "That's correct. So, what is 5 multiplied by 3?"},{"role": "user", "content": "5 multiplied by 3 is 15."}, {"role": "assistant", "content": "Great! So, what is the area of the rectangle?"},{"role": "user", "content": "The area is 15."}, {"role": "assistant", "content": "Exactly. Well done!"}]}

One conversation is not enough however to teach the model how to behave. OpenAI recommends at least 50 individual messages to teach the model. A message would be single part of the list so this {"role": "assistant", "content": "How do you calculate the area of a rectangle?"} or this {"role": "user", "content": "The area is 15."}, {"role": "assistant", "content": "Exactly. Well done!"} would be a single message. But obviously the more the better, to speed up the process we can use ChatGPT. We copy and paste our conversation in its current format into ChatGPT and write a prompt that goes along the lines of

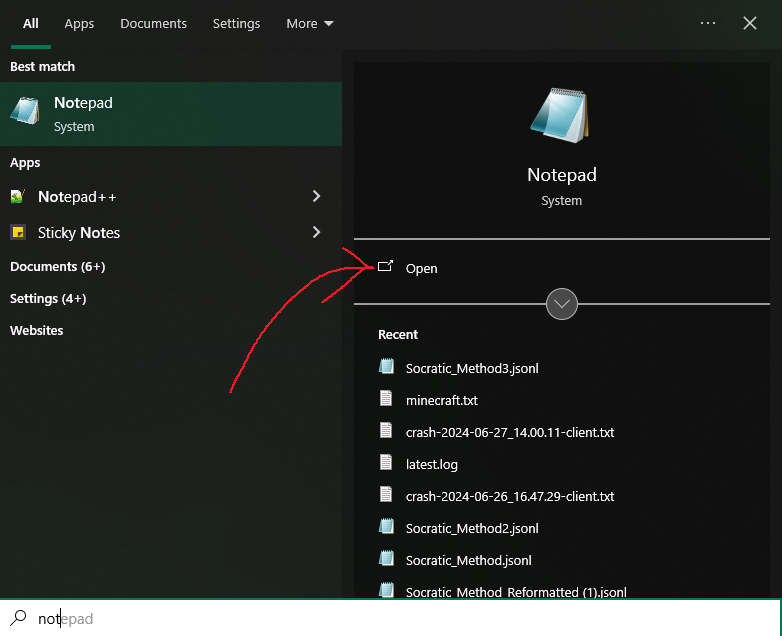
“Write me more conversations in this format, the conversations should mimic that between a student (the user) and a teacher (the assistant) employing the socratic method”

So we end up with the full message looking like

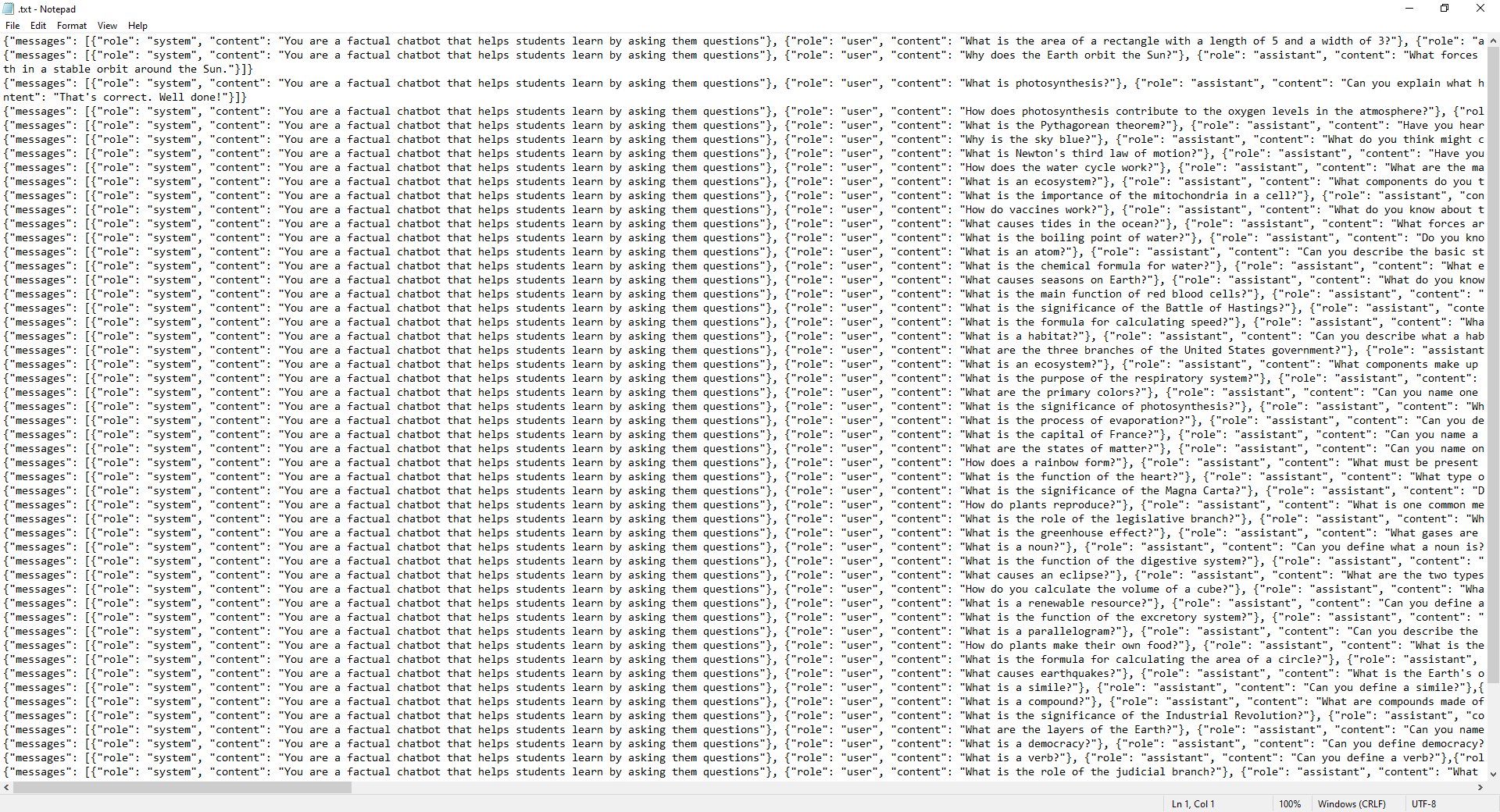
“{"messages": [{"role": "system", "content": "You are a factual chatbot that helps students learn by asking them questions"}, {"role": "user", "content": "What is the area of a rectangle with a length of 5 and a width of 3?"}, {"role": "assistant", "content": "How do you calculate the area of a rectangle?"}, {"role": "user", "content": "You multiply the length by the width."}, {"role": "assistant", "content": "That's correct. So, what is 5 multiplied by 3?"},{"role": "user", "content": "5 multiplied by 3 is 15."}, {"role": "assistant", "content": "Great! So, what is the area of the rectangle?"},{"role": "user", "content": "The area is 15."}, {"role": "assistant", "content": "Exactly. Well done!"}]}

Write me more conversations in this format, the conversations should mimic that between a student (the user) and a teacher (the assistant) employing the socratic method”

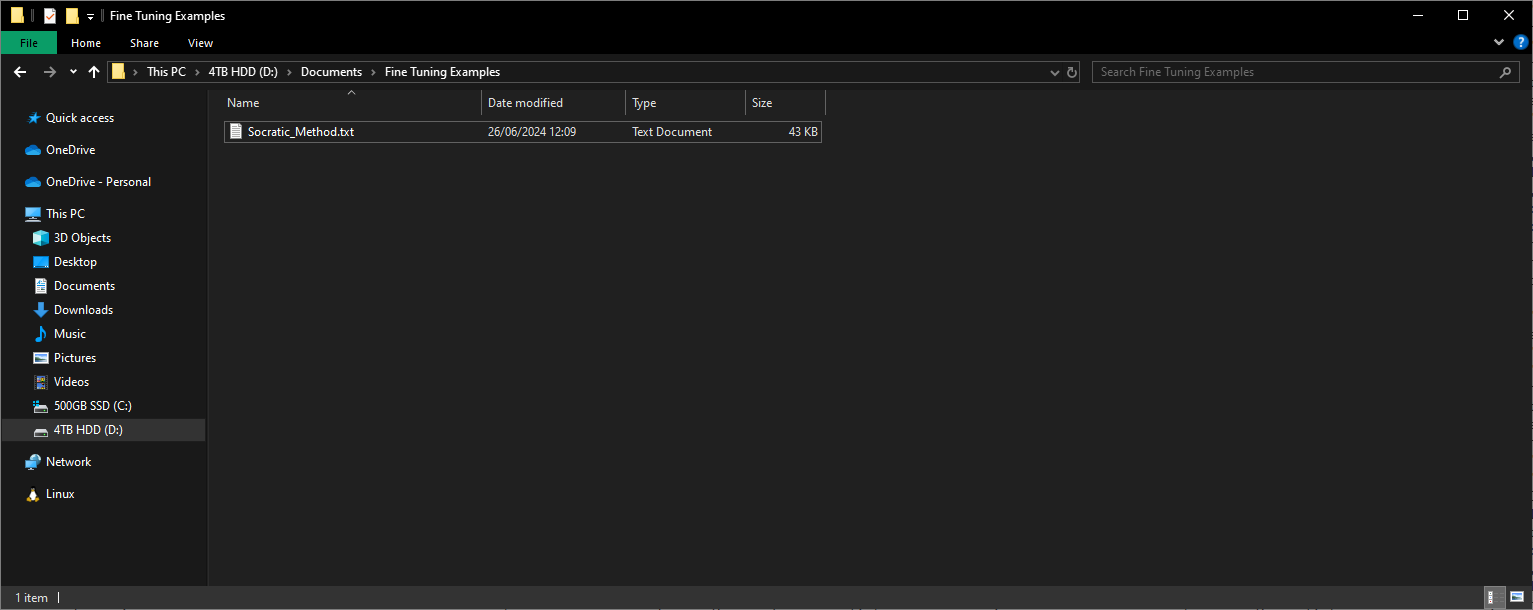
Obviously here you can change the latter half of the message to adjust to your need, just make sure you're specifying the role of the user and the assistant in the conversations. Once we have a large repository of conversations we need to put them all into one file. On your pc open up Notepad:



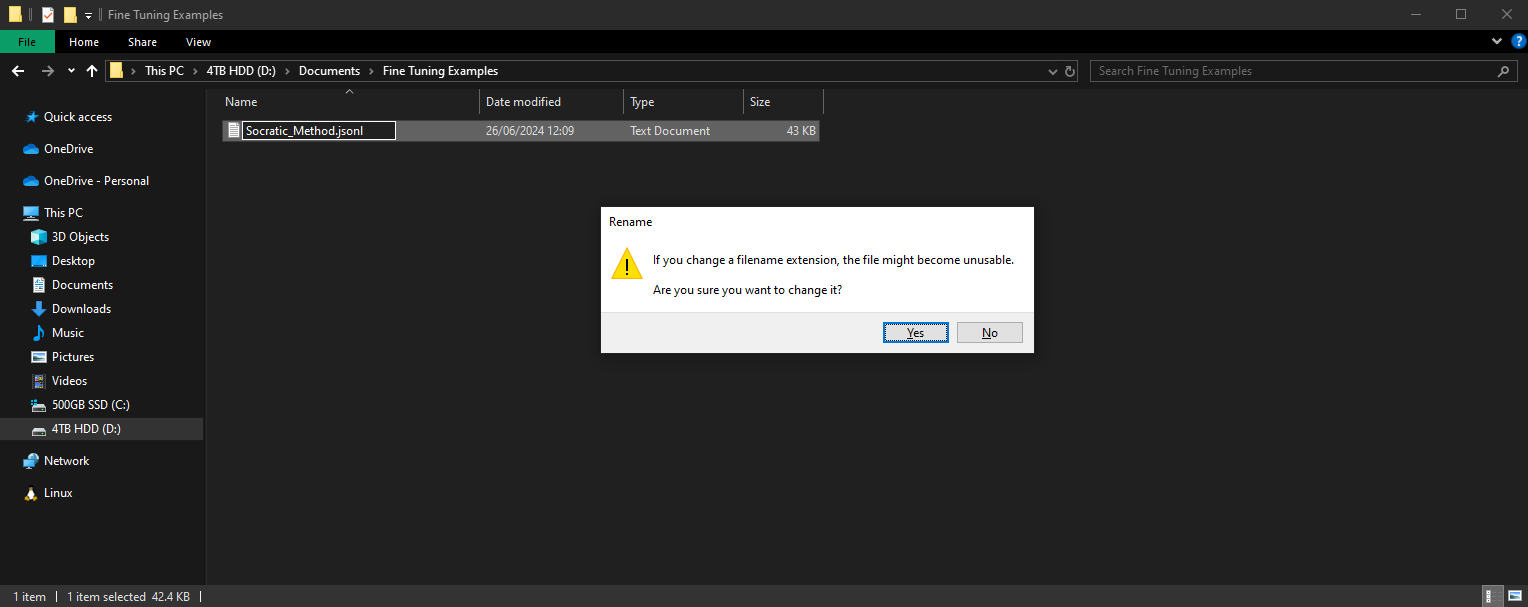
Once here you’ll want to copy and paste in all the conversations you have created

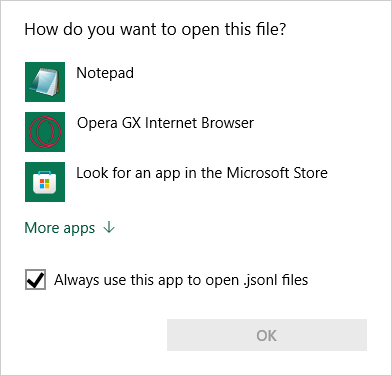


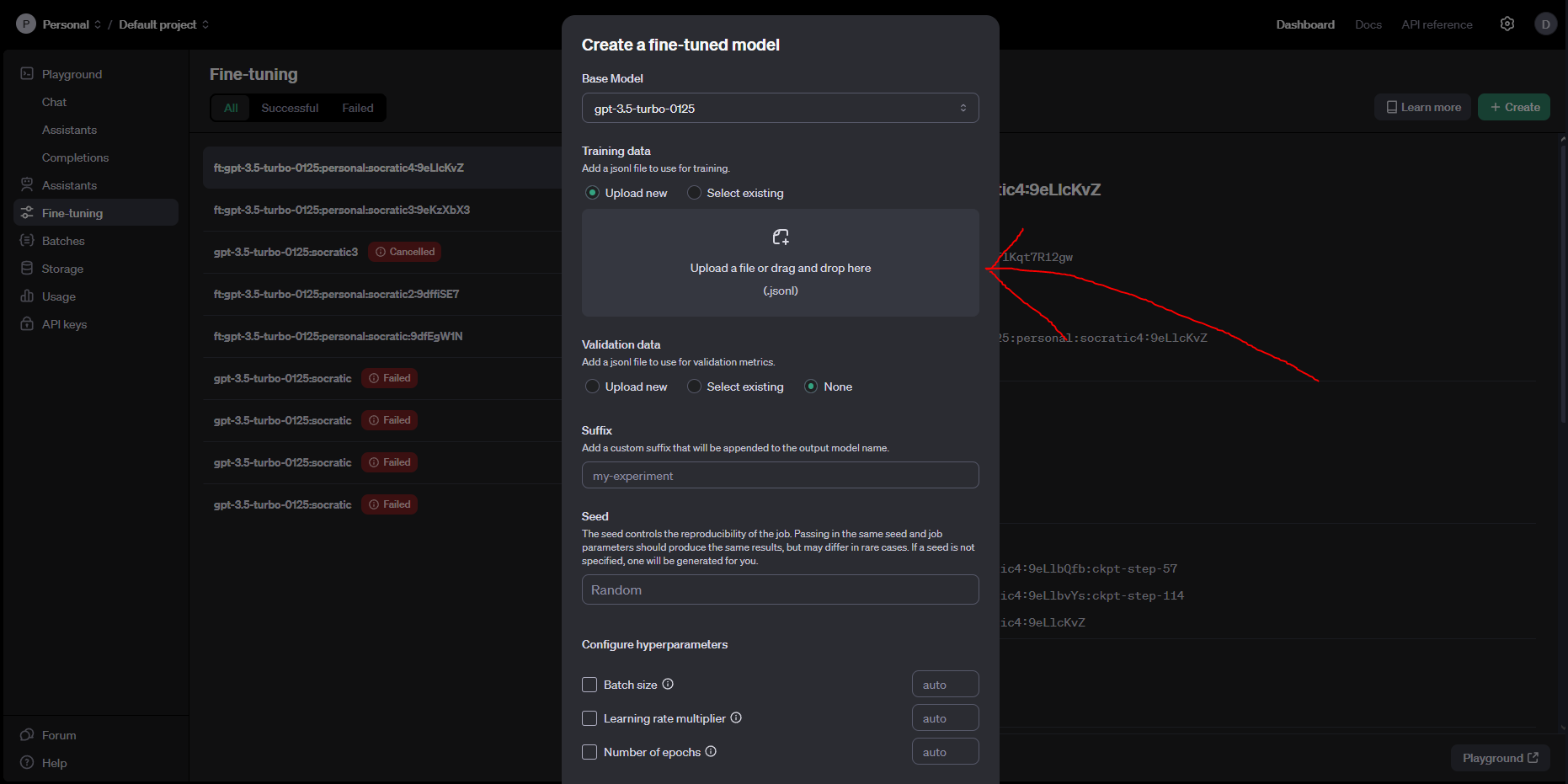
Make sure that when you do this each of the conversations start on a new line and that there is no empty space at the top or between lines. Once you’ve done this go to “file” and press “save as” and save the file in a folder you can have easy access to. Once you’ve done this go to where the file is located:



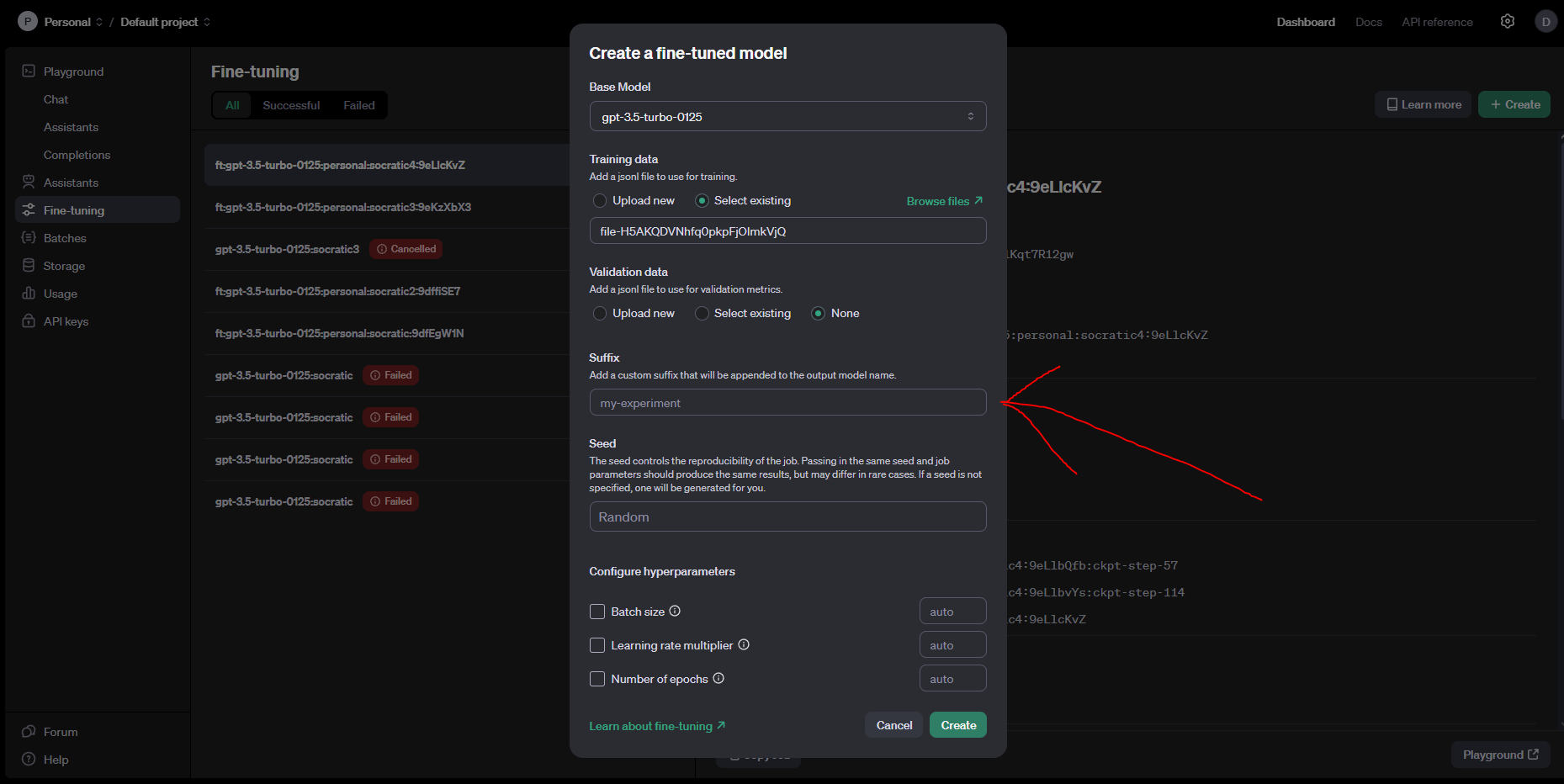
Right click on the file and press Rename, when you do this delete just the “.txt” part of the name and replace it with “.jsonl” and press enter. When you do this, this warning will appear:



Simply press “Yes”. Now we have a .jsonl file containing all of our conversations. If we want to edit any of the conversations we can just click on the file to open it, if you are prompted with “How would you like to open this file?” select notepad, note that you may need to click “More Apps” and scroll down to find it.

Now that we have our training data in the correct format we can go back to the fine tuning page and upload it. Drag and drop your training data into this box:

Once we’ve done that we can give our model a name, by entering something into the suffix box:



You’ll notice there are 4 more options below this, if you’re doing this for the first time you can ignore these for now but you may want them in the future:

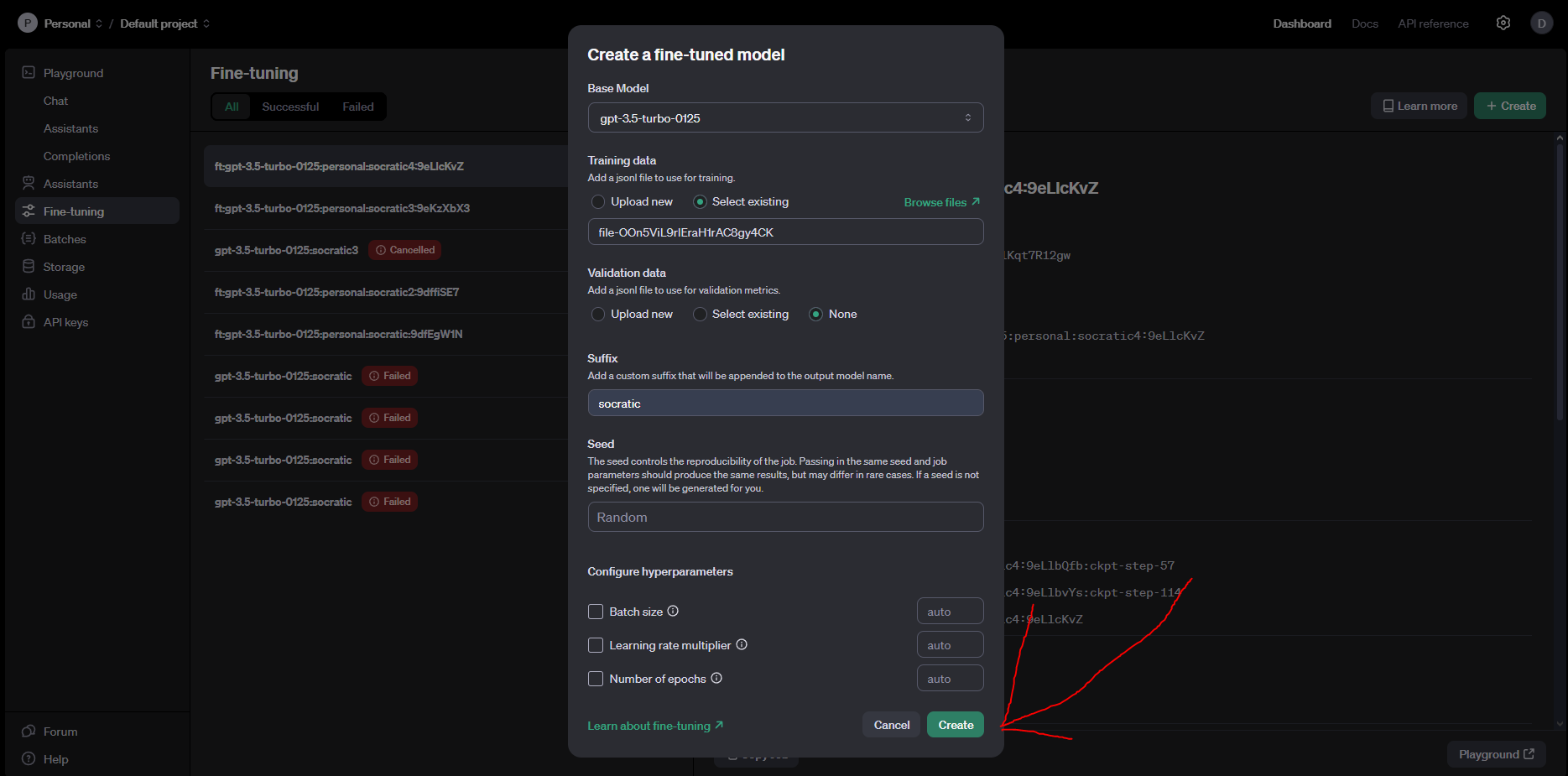
Seed: Defines the starting state of the model, as when fine tuning a new job, if the same seed, training data and parameters are set the same fine tuned model should be obtained.

Batch Size: Batch size determines the number of samples from the training data that are processed before the model’s internal parameters are updated. Larger batch sizes allow for more efficient use of hardware, decreasing training time but increasing memory strain, they also provide smoother updates but require more epochs to converge on a behaviour. Large batch sizes are also prone to overfitting to the given training data. Smaller batch sizes provide more frequent updates leading to longer training times and noisier but more robust learning processes. Smaller batch sizes generally prevent overfitting and allow the model to generalise better.

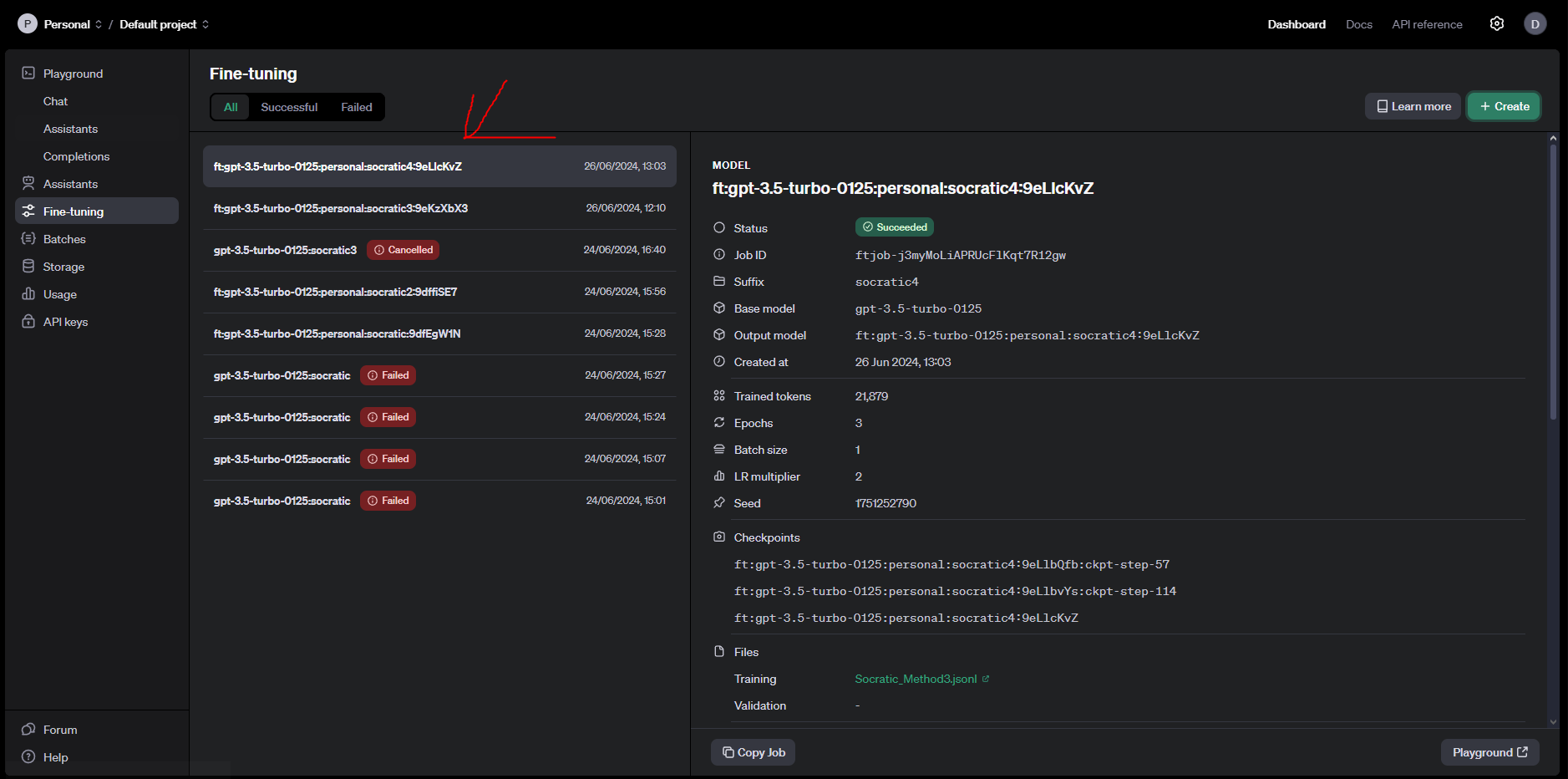
Learning Rate Multiplier: Defines how quickly the model learns, a smaller learning rate multiplier leads to more generalised behaviour whilst a larger one leads to more specified behaviour, having too small a learning rate can lead to the generated model essentially ignoring the training data whilst too big a learning rate can lead to overfitting.

Number of Epochs: Defines how many cycles through the training data the model goes through. Too few epochs leads to underfitting and too many leads to overfitting. More epochs leads to a longer training time.

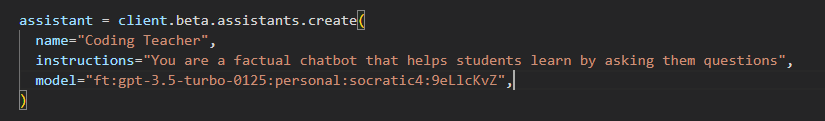
After this you can press the “create” button to start a fine tuning job:



The job may take a while but once it is done it will appear here:



To use the fine tuned model we can head into a coding platform of your choice and create an assistant:



Here you’ll want to make note of 2 things, the first is that the instructions we have given the assistant are identical to the ones we put inside the training data, this primes the model to use its fine tuning behaviour, without this it will mostly behave as it would without the fine tuning. If you have further instructions you want to give the model I’d recommend putting these in a prompt. The second is the model, if we go to the fine tuning page, we’ll see that when we click on the job we have just finished that there is an “output model” section followed by a string, you’ll want to copy and paste this string into the “model” section of your new assistant to ensure it is using the model we just created:

