Customize a model with fine-tuning

Prerequisites

- When to use Azure OpenAI fine-tuning guide.
- An Azure subscription.
- Access granted to Azure OpenAI in the desired Azure subscription.
- An Azure OpenAI resource that's located in a region that supports fine-tuning of the Azure OpenAI model
- Fine-tuning access requires **Cognitive Services OpenAI Contributor**.

Models

The following models support fine-tuning:

- babbage-002
- davinci-002
- gpt-35-turbo (0613)
- gpt-35-turbo (1106)
- gpt-35-turbo (0125)

Regions currently support fine-tuning.

Fine-tuning models

babbage-002 and davinci-002 are not trained to follow instructions. Querying these base models should only be done as a point of reference to a fine-tuned version to evaluate the progress of your training.

gpt-35-turbo - fine-tuning of this model is limited to a subset of regions, and is not available in every region the base model is available.

Model ID	Fine-Tuning Regions	Max Request (tokens)	Training Data (up to)
babbage-002	North Central US Sweden Central Switzerland West	16,384	Sep 2021

davinci-002	North Central US Sweden Central Switzerland West	16,384	Sep 2021
gpt-35-turbo (0613)	East US2 North Central US Sweden Central Switzerland West	4,096	Sep 2021
gpt-35-turbo (1106)	East US2 North Central US Sweden Central Switzerland West	Input: 16,385 Output: 4,096	Sep 2021
gpt-35-turbo (0125)	East US2 North Central US Sweden Central Switzerland West	16,385	Sep 2021

Review the workflow for Azure OpenAI Studio

Take a moment to review the fine-tuning workflow for using Azure OpenAI Studio:

1. Prepare your training and validation data.
2. Use the **Create custom model** wizard in Azure OpenAI Studio to train your custom model.
    a. Select a base model.
    b. Choose your training data.
    c. Optionally, choose your validation data.
    d. Optionally, configure advanced options for your fine-tuning job.

e. Review your choices and train your new custom model.

3. Check the status of your custom fine-tuned model.
4. Deploy your custom model for use.
5. Use your custom model.
6. Optionally, analyze your custom model for performance and fit.

Prepare your training and validation data

Your training data and validation data sets consist of input and output examples for how you would like the model to perform.

Different model types require a different format of training data.

chat_completion models

The training and validation data you use **must** be formatted as a JSON Lines (JSONL) document. For gpt-35-turbo-0613 the fine-tuning dataset must be formatted in the conversational format that is used by the Chat completions API.

Example file format


```
{"messages": [{"role": "system", "content": "You are an Xbox customer support agent whose primary goal is to help users with issues they are experiencing with their Xbox devices. You are friendly and concise. You only provide factual answers to queries, and do not provide answers that are not related to Xbox."}, {"role": "user", "content": "Is Xbox better than PlayStation?"}, {"role": "assistant", "content": "I apologize, but I cannot provide personal opinions. My primary job is to assist you with any issues related to your Xbox device. Do you have any Xbox-related issues that need addressing?"}]}

{"messages": [{"role": "system", "content": "You are an Xbox customer support agent whose primary goal is to help users with issues they are experiencing with their Xbox devices. You are friendly and concise. You only provide factual answers to queries, and do not provide answers that are not related to Xbox."}, {"role": "user", "content": "Can I play PC games on my Xbox?"}, {"role": "assistant", "content": "While Xbox consoles don't natively support PC games, they can play an array of Xbox games, many of which are also available on PC. Can I assist you with finding games compatible with your Xbox device?"}]}

{"messages": [{"role": "system", "content": "You are an Xbox customer support agent whose primary goal is to help users with issues they are experiencing with their Xbox devices. You are friendly and concise. You only provide factual answers to queries, and do not provide answers that are not related to Xbox."}, {"role": "user", "content": "I'm having trouble connecting my Xbox to the Wi-Fi."}, {"role": "assistant", "content": "No worries, let's go through the network settings on your Xbox. Can you please tell me what happens when you try to connect it to the Wi-Fi?"}]}
```


Multi-turn chat file format

Multiple turns of a conversation in a single line of your jsonl training file is also supported. To skip fine-tuning on specific assistant messages add the optional weight key value pair. Currently weight can be set to 0 or 1.

```
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "What's the capital of France?"}, {"role": "assistant", "content": "Paris", "weight": 0}, {"role": "user", "content": "Can you be more sarcastic?"}, {"role": "assistant", "content": "Paris, as if everyone doesn't know that already.", "weight": 1}]}

{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"}, {"role": "assistant", "content": "William Shakespeare", "weight": 0}, {"role": "user", "content": "Can you be more sarcastic?"}, {"role": "assistant", "content": "Oh, just some guy named William Shakespeare. Ever heard of him?", "weight": 1}]}

{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "How far is the Moon from Earth?"}, {"role": "assistant", "content": "384,400 kilometers", "weight": 0}, {"role": "user", "content": "Can you be more sarcastic?"}, {"role": "assistant", "content": "Around 384,400 kilometers. Give or take a few, like that really matters.", "weight": 1}]}
```

In addition to the JSONL format, training and validation data files must be encoded in UTF-8 and include a byte-order mark (BOM). The file must be less than 512 MB in size.

Create your training and validation datasets

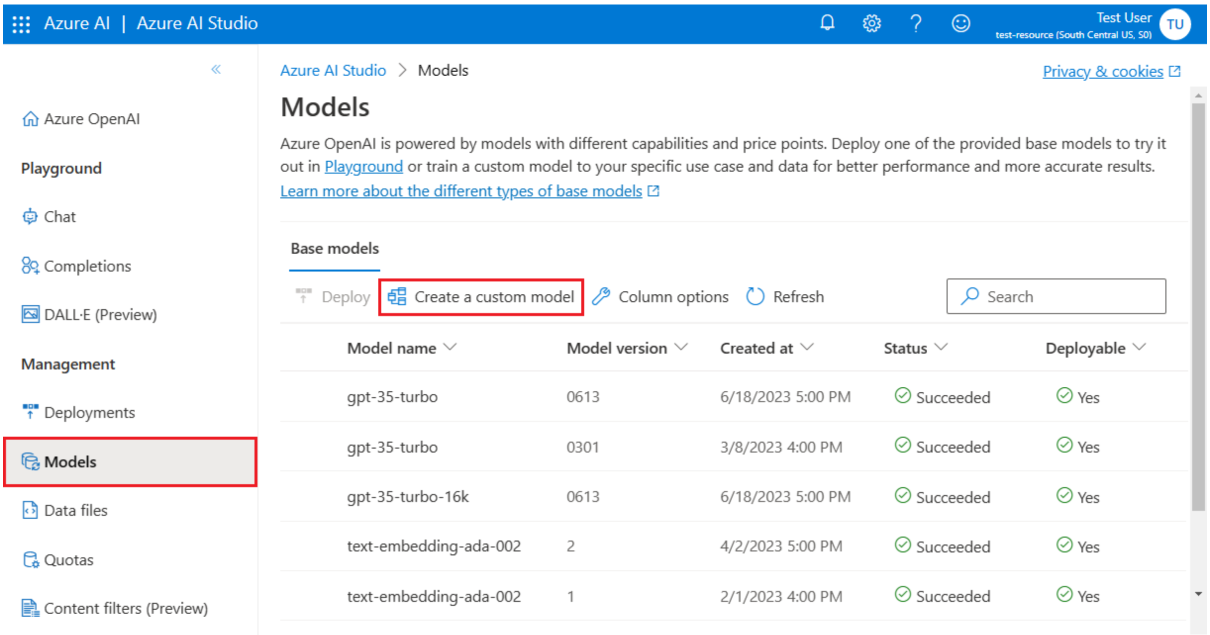
The more training examples you have, the better. Fine tuning jobs will not proceed without at least 10 training examples, but such a small number are not enough to noticeably influence model responses. It is best practice to provide hundreds, if not thousands, of training examples to be successful.

In general, doubling the dataset size can lead to a linear increase in model quality. But keep in mind, low quality examples can negatively impact performance. If you train the model on a large amount of internal data, without first pruning the dataset for only the highest quality examples you could end up with a model that performs much worse than expected.

Use the Create custom model wizard

Azure OpenAI Studio provides the **Create custom model** wizard, so you can interactively create and train a fine-tuned model for your Azure resource.

1. Open Azure OpenAI Studio at https://oai.azure.com/ and sign in with credentials that have access to your Azure OpenAI resource. During the sign-in workflow, select the appropriate directory, Azure subscription, and Azure OpenAI resource.
2. In Azure OpenAI Studio, browse to the **Management > Models** pane, and select **Create a custom model**.



The **Create custom model** wizard opens.

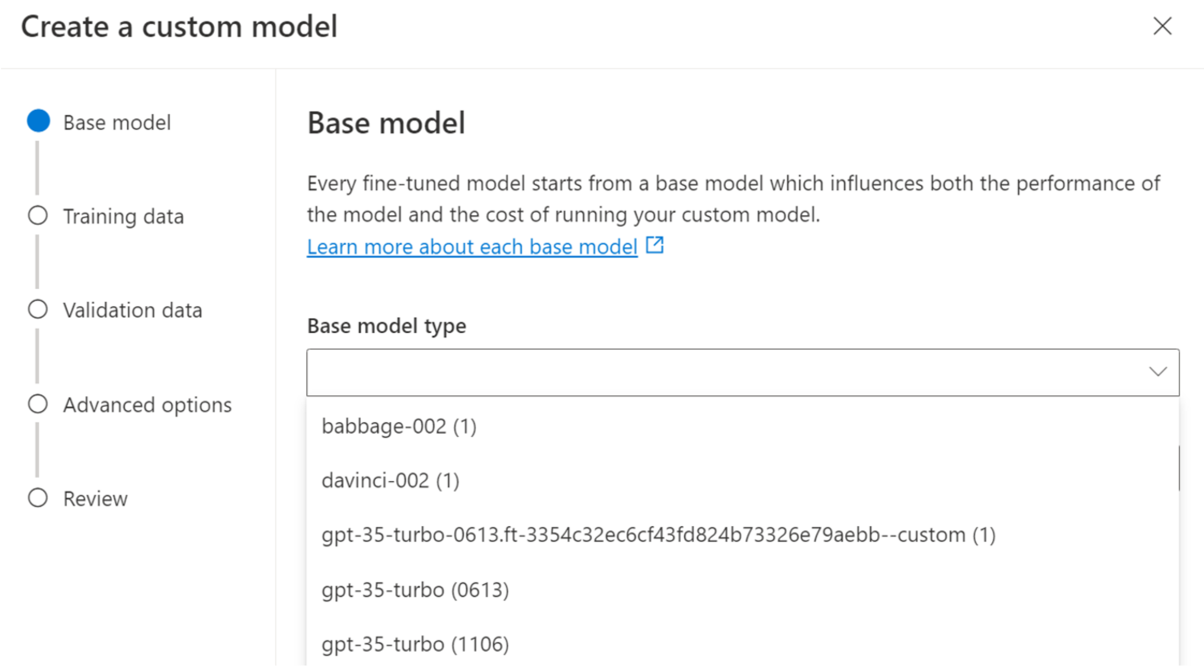
Select the base model

The first step in creating a custom model is to choose a base model. The **Base model** pane lets you choose a base model to use for your custom model. Your choice influences both the performance and the cost of your model.

Select the base model from the **Base model type** dropdown, and then select **Next** to continue.

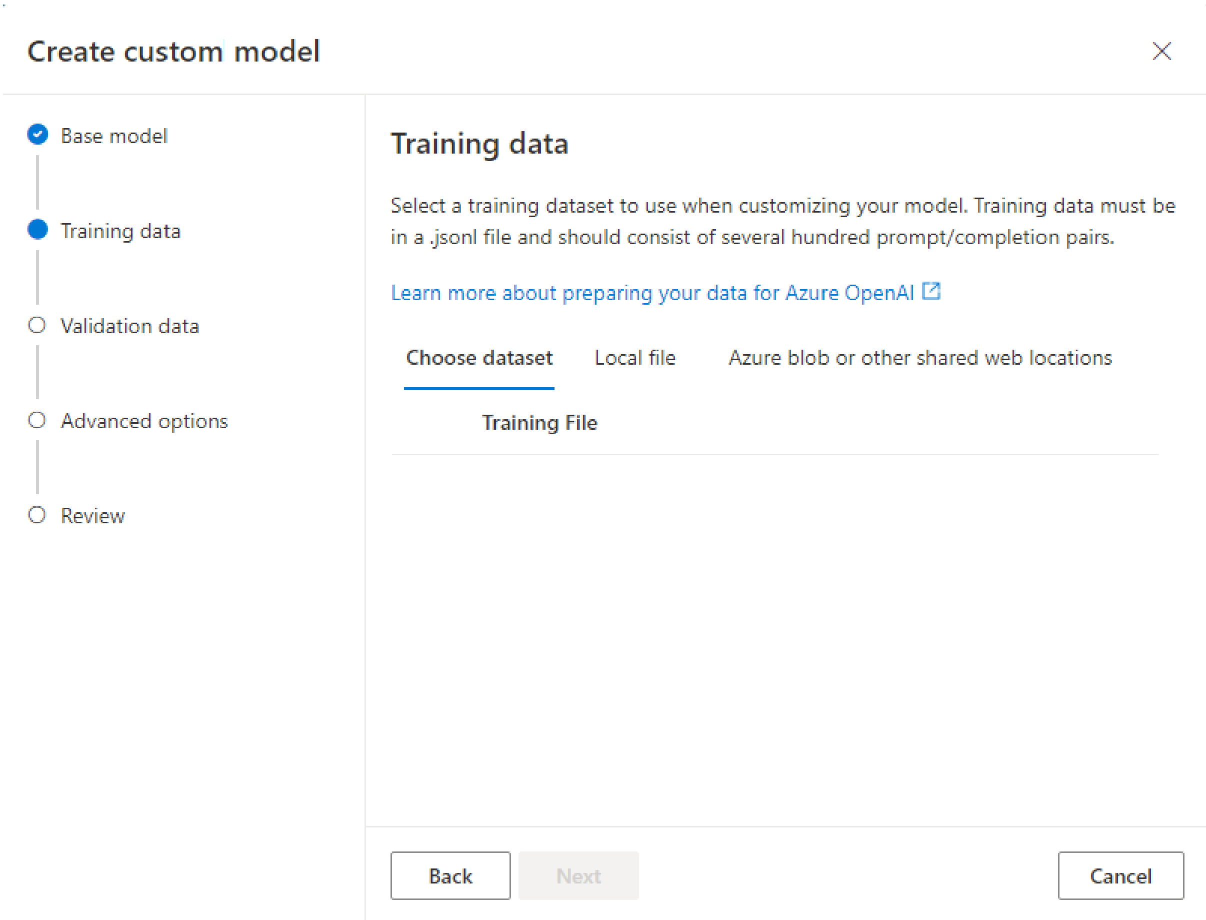
You can create a custom model from one of the following available base models:

- babbage-002
- davinci-002
- gpt-35-turbo (0613)
- gpt-35-turbo (1106)
- Or you can fine tune a previously fine-tuned model, formatted as base-model.ft-{jobid}.



Choose your training data

The next step is to either choose existing prepared training data or upload new prepared training data to use when customizing your model. The **Training data** pane displays any existing, previously uploaded datasets and also provides options to upload new training data.



- If your training data is already uploaded to the service, select **Choose dataset**.
    - Select the file from the list shown in the **Training data** pane.
- To upload new training data, use one of the following options:
    - Select **Local file** to upload training data from a local file.
    - Select **Azure blob or other shared web locations** to import training data from Azure Blob or another shared web location.

For large data files, we recommend that you import from an Azure Blob store. Large files can become unstable when uploaded through multipart forms because the requests are atomic and can't be retried or resumed

Note

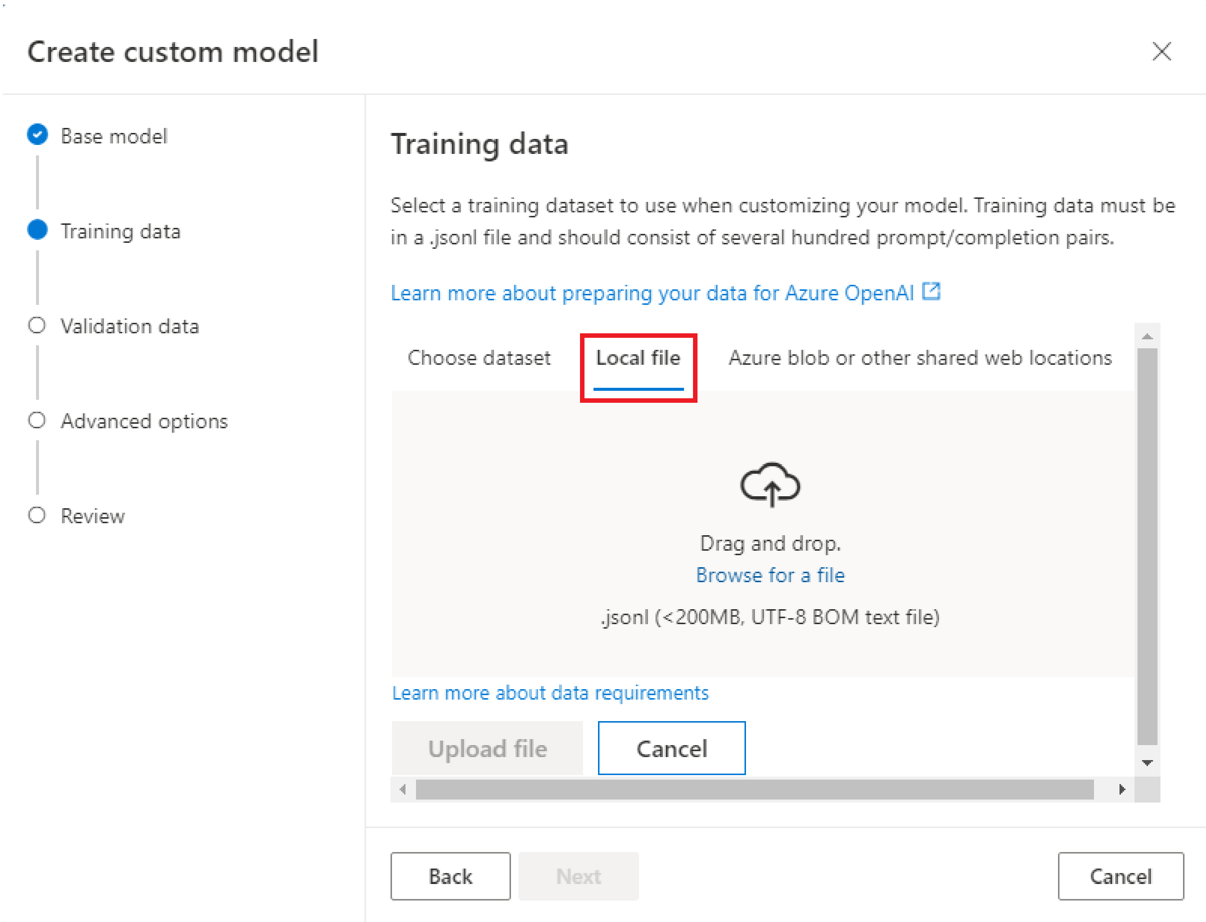
Training data files must be formatted as JSONL files, encoded in UTF-8 with a byte-order mark (BOM). The file must be less than 512 MB in size.

Upload training data from local file

You can upload a new training dataset to the service from a local file by using one of the following methods:

- Drag and drop the file into the client area of the **Training data** pane, and then select **Upload file**.
- Select **Browse for a file** from the client area of the **Training data** pane, choose the file to upload from the **Open** dialog, and then select **Upload file**.

After you select and upload the training dataset, select **Next** to continue.

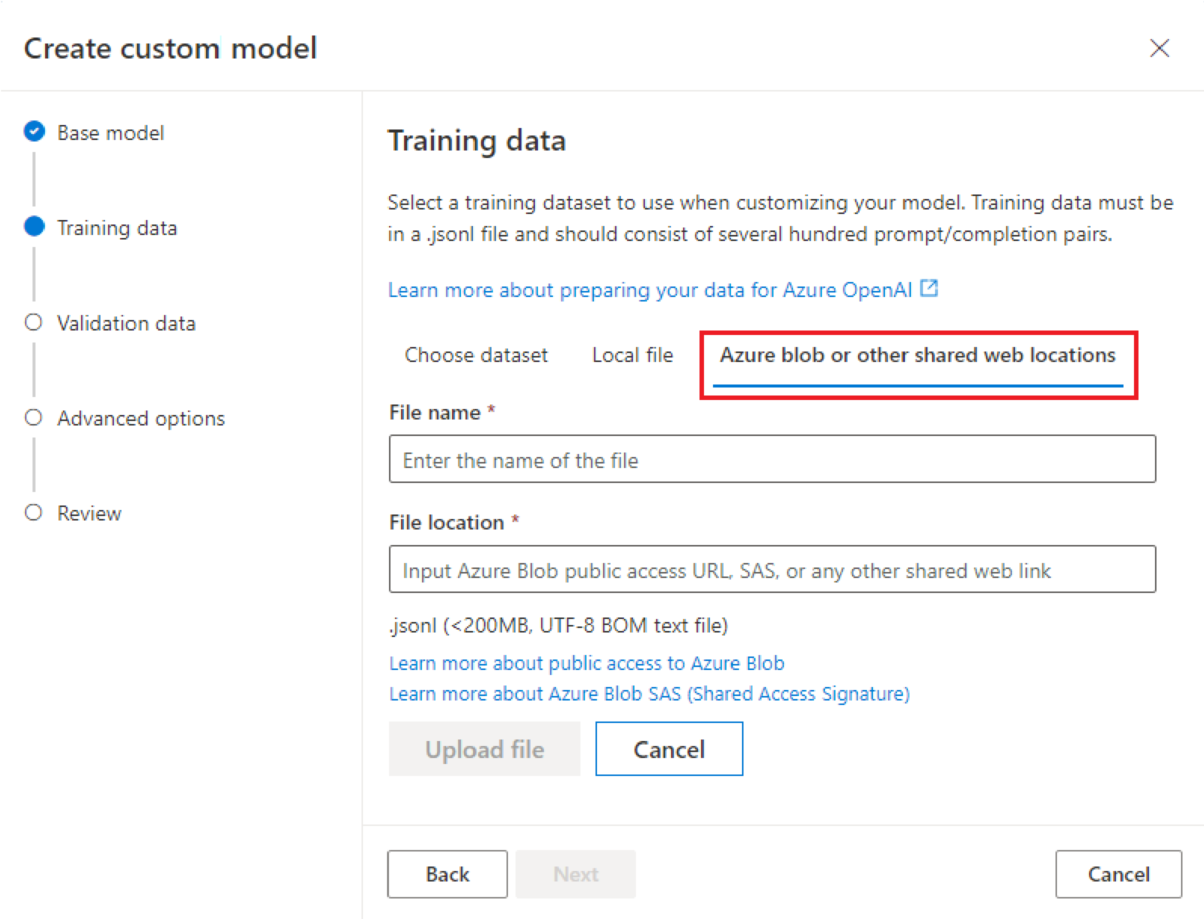


Import training data from Azure Blob store

You can import a training dataset from Azure Blob or another shared web location by providing the name and location of the file.

1. Enter the **File name** for the file.
2. For the **File location**, provide the Azure Blob URL, the Azure Storage shared access signature (SAS), or other link to an accessible shared web location.
3. Select **Upload file** to import the training dataset to the service.

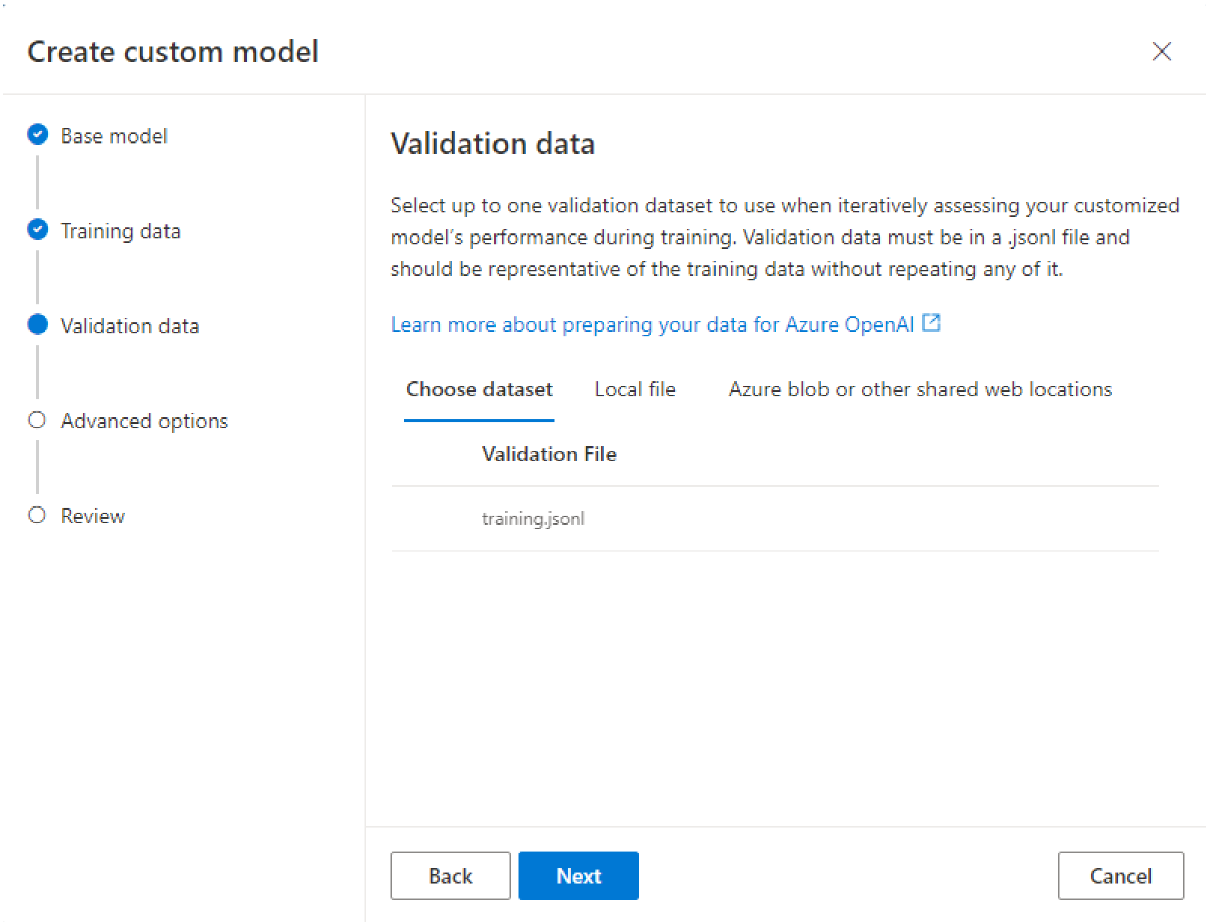
After you select and upload the training dataset, select **Next** to continue.



Choose your validation data

The next step provides options to configure the model to use validation data in the training process. If you don't want to use validation data, you can choose **Next** to continue to the advanced options for the model. Otherwise, if you have a validation dataset, you can either choose existing prepared validation data or upload new prepared validation data to use when customizing your model.

The **Validation data** pane displays any existing, previously uploaded training and validation datasets and provides options by which you can upload new validation data.



- If your validation data is already uploaded to the service, select **Choose dataset**.
  - Select the file from the list shown in the **Validation data** pane.
- To upload new validation data, use one of the following options:
  - Select **Local file** to upload validation data from a local file.
  - Select **Azure blob or other shared web locations** to import validation data from Azure Blob or another shared web location.

For large data files, we recommend that you import from an Azure Blob store. Large files can become unstable when uploaded through multipart forms because the requests are atomic and can't be retried or resumed.

Note

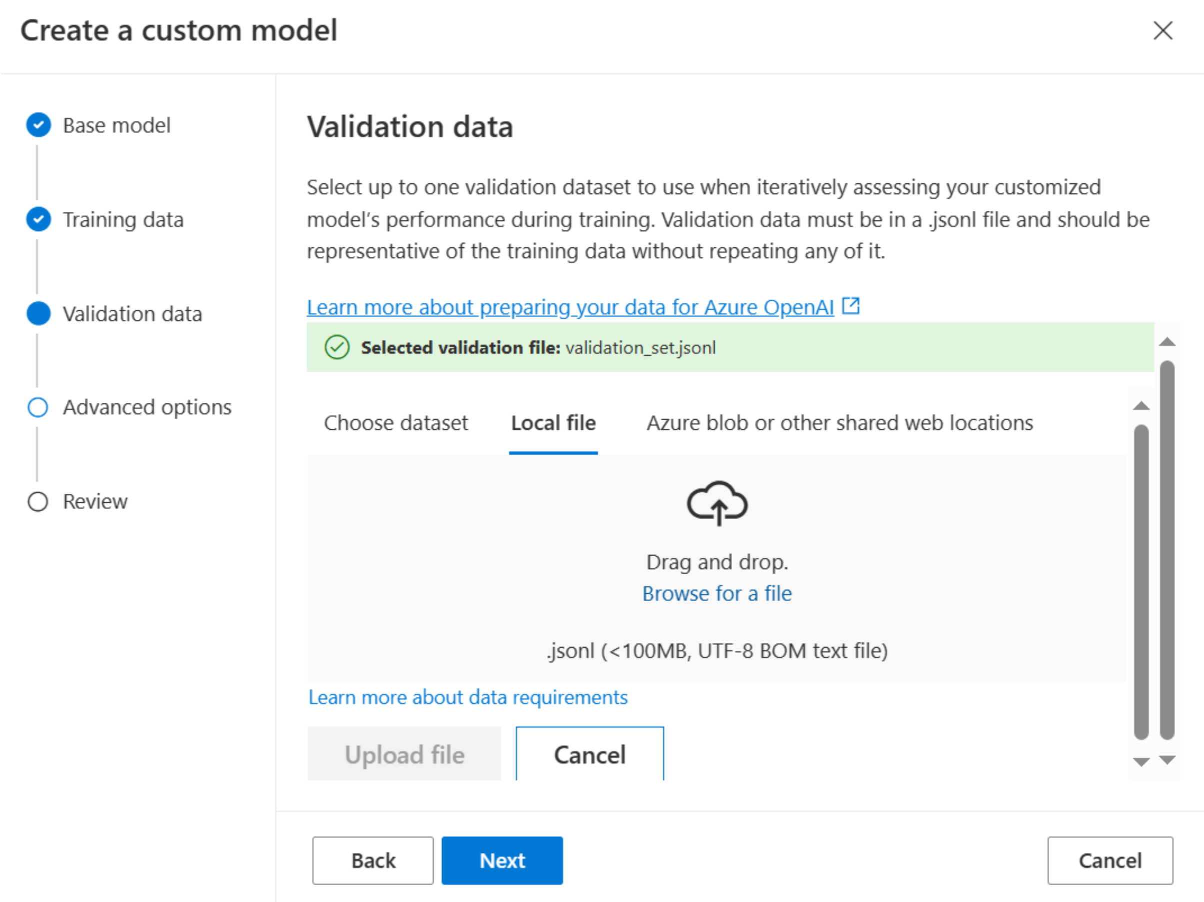
Similar to training data files, validation data files must be formatted as JSONL files, encoded in UTF-8 with a byte-order mark (BOM). The file must be less than 512 MB in size.

Upload validation data from local file

You can upload a new validation dataset to the service from a local file by using one of the following methods:

- Drag and drop the file into the client area of the **Validation data** pane, and then select **Upload file**.
- Select **Browse for a file** from the client area of the **Validation data** pane, choose the file to upload from the **Open** dialog, and then select **Upload file**.

After you select and upload the validation dataset, select **Next** to continue.

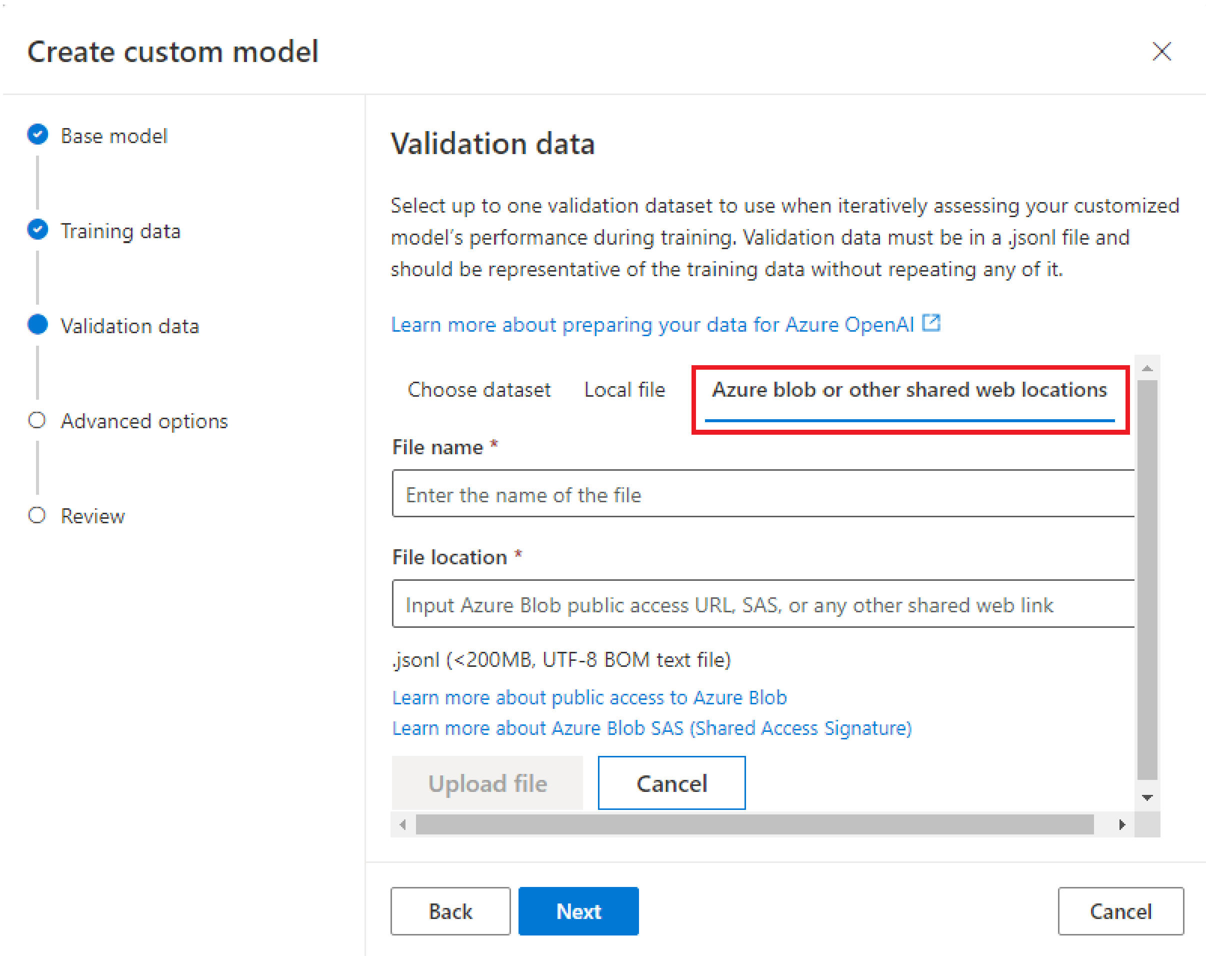


Import validation data from Azure Blob store

You can import a validation dataset from Azure Blob or another shared web location by providing the name and location of the file.

1. Enter the **File name** for the file.
2. For the **File location**, provide the Azure Blob URL, the Azure Storage shared access signature (SAS), or other link to an accessible shared web location.
3. Select **Upload file** to import the training dataset to the service.

After you select and upload the validation dataset, select **Next** to continue.

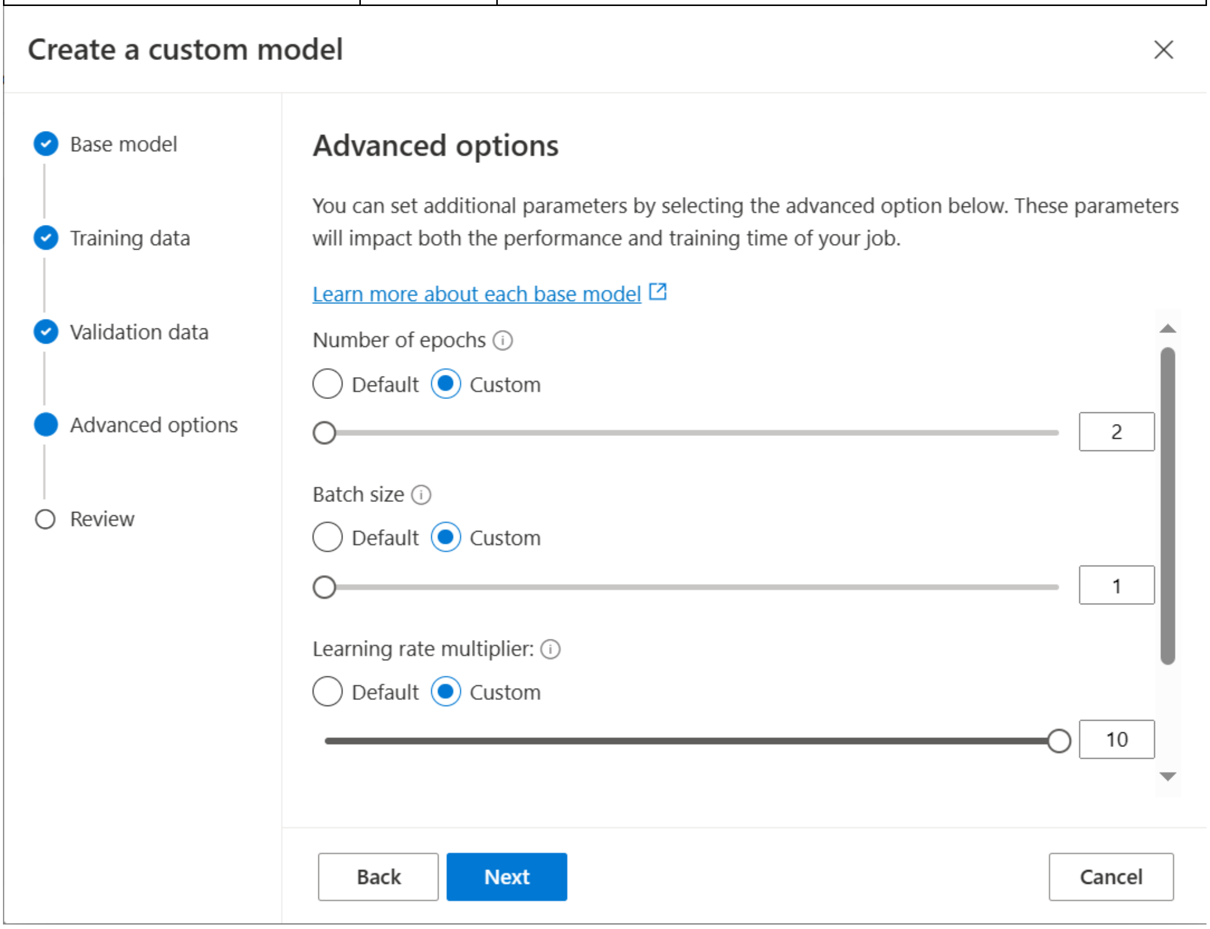


Configure advanced options

The **Create custom model** wizard shows the hyperparameters for training your fine-tuned model on the **Advanced options** pane. The following hyperparameters are available:

Name	Type	Description
batch_size	integer	The batch size to use for training. The batch size is the number of training examples used to train a single forward and backward pass. In general, we've found that larger batch sizes tend to work better for larger datasets. The default value as well as the maximum value for this property are specific to a base model. A larger batch size means that model parameters are updated less frequently, but with lower variance.

learning_rate_multiplier	number	The learning rate multiplier to use for training. The fine-tuning learning rate is the original learning rate used for pre-training multiplied by this value. Larger learning rates tend to perform better with larger batch sizes. We recommend experimenting with values in the range 0.02 to 0.2 to see what produces the best results. A smaller learning rate may be useful to avoid overfitting.
n_epochs	integer	The number of epochs to train the model for. An epoch refers to one full cycle through the training dataset.

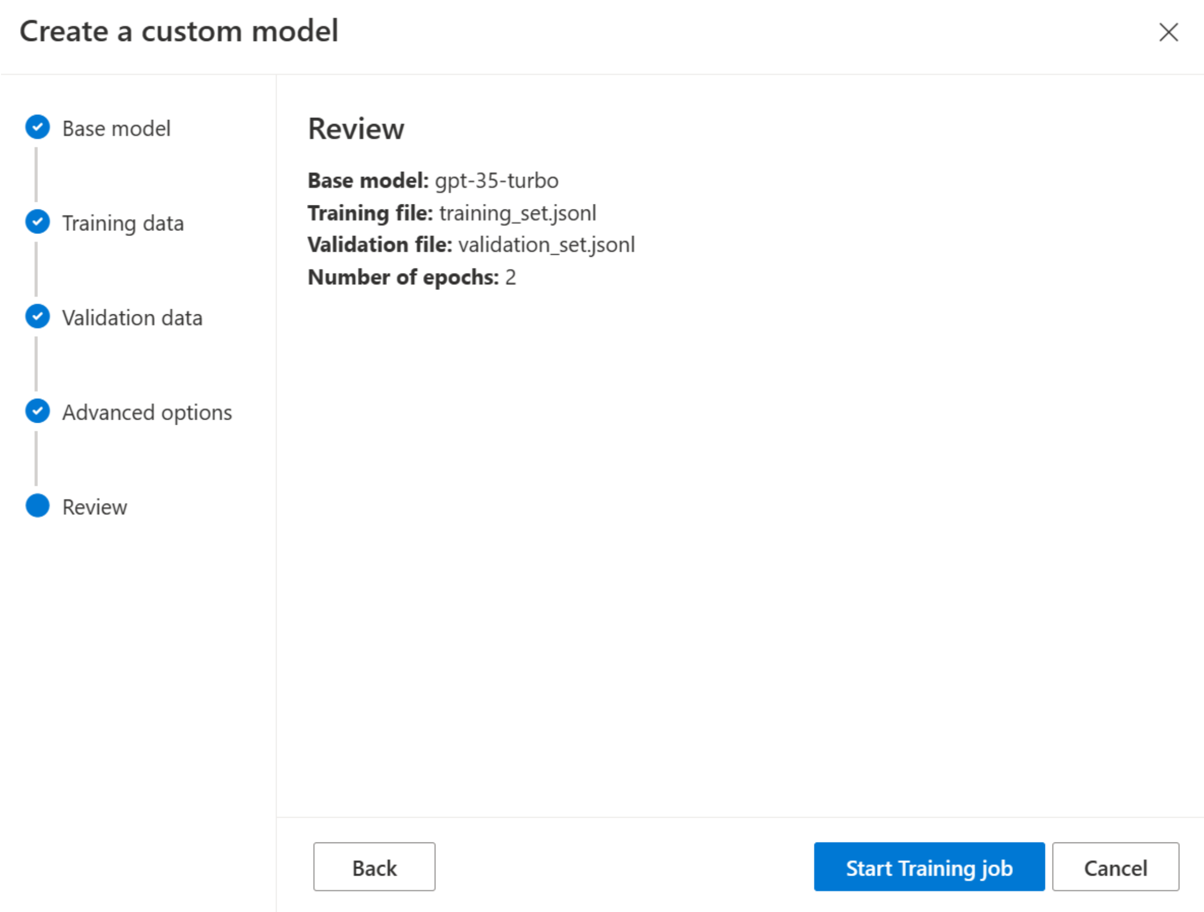


Select **Default** to use the default values for the fine-tuning job, or select **Advanced** to display and edit the hyperparameter values. When defaults are selected, we determine the correct value algorithmically based on your training data.

After you configure the advanced options, select **Next** to review your choices and train your fine-tuned model.

Review your choices and train your model

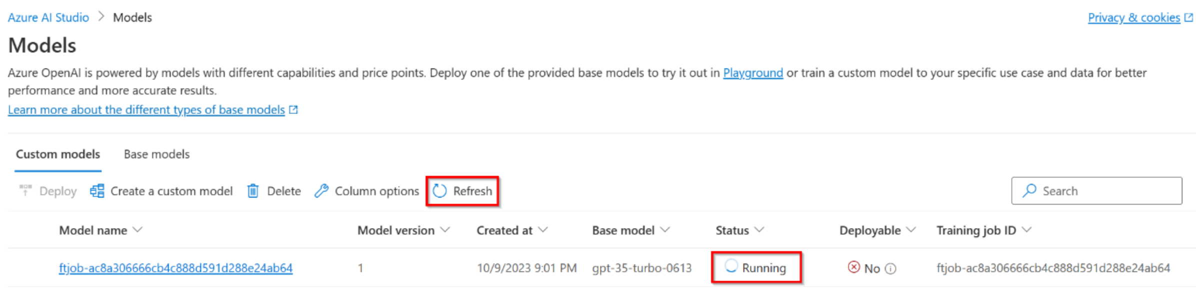
The **Review** pane of the wizard displays information about your configuration choices.



If you're ready to train your model, select **Start Training job** to start the fine-tuning job and return to the **Models** pane.

Check the status of your custom model

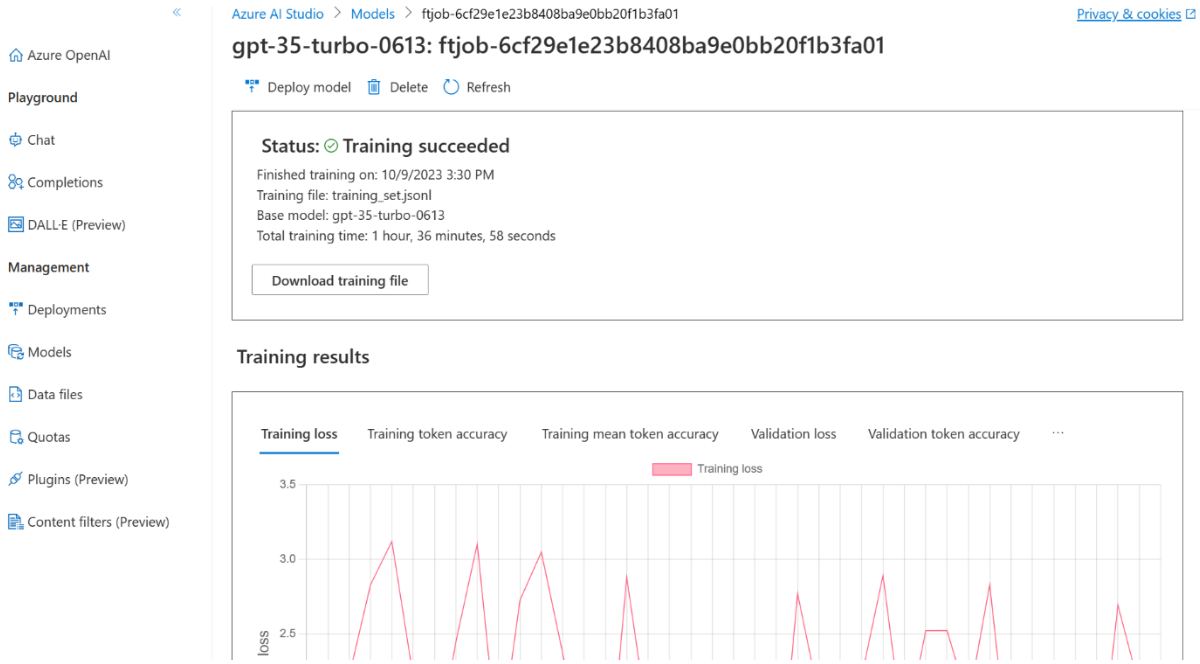
The **Models** pane displays information about your custom model in the **Customized models** tab. The tab includes information about the status and job ID of the fine-tune job for your custom model. When the job completes, the tab displays the file ID of the result file. You might need to select **Refresh** in order to see an updated status for the model training job.



After you start a fine-tuning job, it can take some time to complete. Your job might be queued behind other jobs on the system. Training your model can take minutes or hours depending on the model and dataset size.

Here are some of the tasks you can do on the **Models** pane:

- Check the status of the fine-tuning job for your custom model in the **Status** column of the **Customized models** tab.
- In the **Model name** column, select the model name to view more information about the custom model. You can see the status of the fine-tuning job, training results, training events, and hyperparameters used in the job.
- Select **Download training file** to download the training data you used for the model.
- Select **Download results** to download the result file attached to the fine-tuning job for your model and analyze your custom model for training and validation performance.
- Select **Refresh** to update the information on the page.



Deploy a custom model

When the fine-tuning job succeeds, you can deploy the custom model from the **Models** pane. You must deploy your custom model to make it available for use with completion calls.

Important

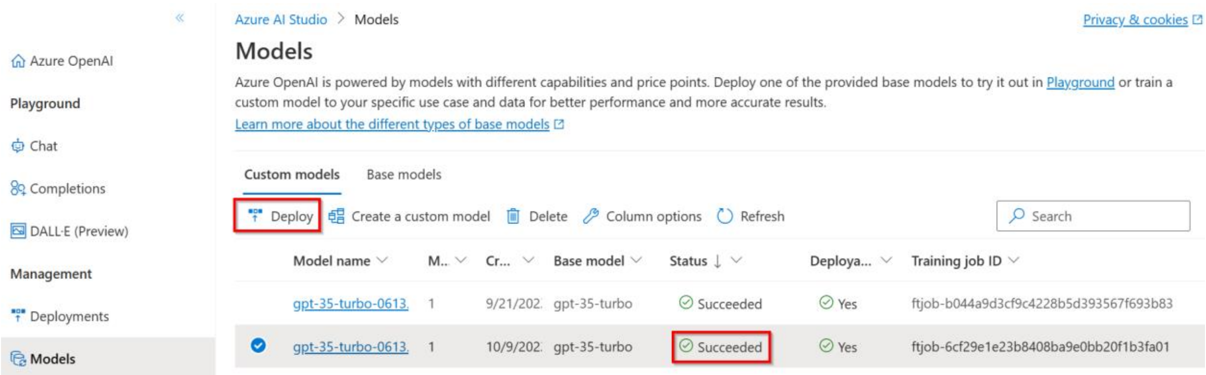
After you deploy a customized model, if at any time the deployment remains inactive for greater than fifteen (15) days, the deployment is deleted. The deployment of a customized model is *inactive* if the model was deployed more than fifteen (15) days ago and no completions or chat completions calls were made to it during a continuous 15-day period.

The deletion of an inactive deployment doesn't delete or affect the underlying customized model, and the customized model can be redeployed at any time. As described in [Azure OpenAI Service pricing](), each customized (fine-tuned) model that's deployed incurs an hourly hosting cost regardless of whether completions or chat completions calls are being made to the model. To learn more about planning and managing costs with Azure OpenAI, refer to the guidance in [Plan to manage costs for Azure OpenAI Service]().

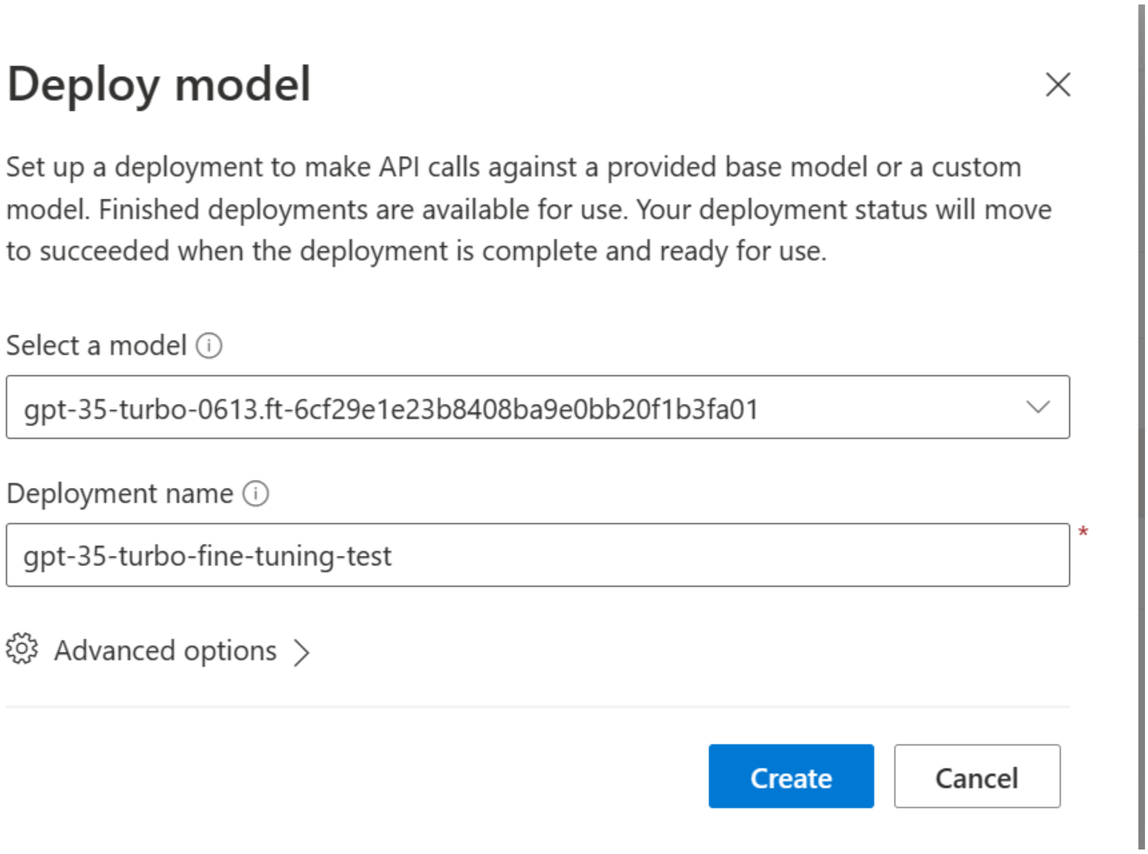
Note

Only one deployment is permitted for a custom model. An error message is displayed if you select an already-deployed custom model.

To deploy your custom model, select the custom model to deploy, and then select **Deploy model**.



The **Deploy model** dialog box opens. In the dialog box, enter your **Deployment name** and then select **Create** to start the deployment of your custom model.



You can monitor the progress of your deployment on the **Deployments** pane in Azure OpenAI Studio.

Cross region deployment

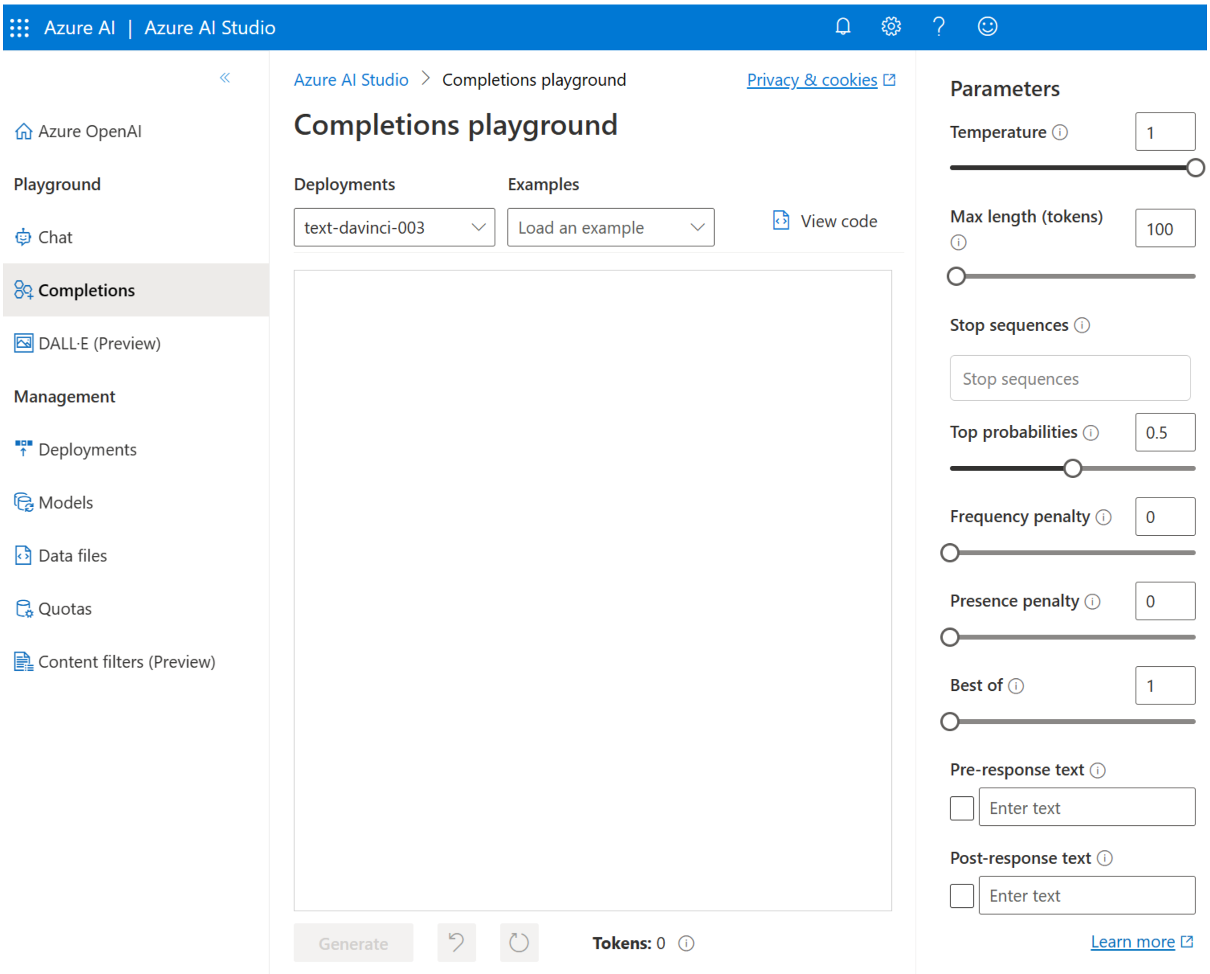
Fine-tuning supports deploying a fine-tuned model to a different region than where the model was originally fine-tuned. You can also deploy to a different subscription/region.

The only limitations are that the new region must also support fine-tuning and when deploying cross subscription the account generating the authorization token for the deployment must have access to both the source and destination subscriptions.

Cross subscription/region deployment can be accomplished via Python or REST.

Use a deployed custom model

After your custom model deploys, you can use it like any other deployed model. You can use the **Playgrounds** in [Azure OpenAI Studio](#) to experiment with your new deployment. You can continue to use the same parameters with your custom model, such as temperature and max_tokens, as you can with other deployed models. For fine-tuned babbage-002 and davinci-002 models you will use the Completions playground and the Completions API. For fine-tuned gpt-35-turbo-0613 models you will use the Chat playground and the Chat completion API.



Analyze your custom model

Azure OpenAI attaches a result file named *results.csv* to each fine-tuning job after it completes. You can use the result file to analyze the training and validation performance of your custom model. The file ID for the result file is listed for each custom model in the **Result file Id** column on the **Models** pane for Azure OpenAI Studio. You can use the file ID to identify and download the result file from the **Data files** pane of Azure OpenAI Studio.

The result file is a CSV file that contains a header row and a row for each training step performed by the fine-tuning job. The result file contains the following columns:

Column name	Description
step	The number of the training step. A training step represents a single pass, forward and backward, on a batch of training data.
train_loss	The loss for the training batch.

training_accuracy	The percentage of completions in the training batch for which the model's predicted tokens exactly matched the true completion tokens. For example, if the batch size is set to 3 and your data contains completions [[1, 2], [0, 5], [4, 2]], this value is set to 0.67 (2 of 3) if the model predicted [[1, 1], [0, 5], [4, 2]].
train_mean_token_accuracy	The percentage of tokens in the training batch correctly predicted by the model. For example, if the batch size is set to 3 and your data contains completions [[1, 2], [0, 5], [4, 2]], this value is set to 0.83 (5 of 6) if the model predicted [[1, 1], [0, 5], [4, 2]].
valid_loss	The loss for the validation batch.
valid_accuracy	The percentage of completions in the validation batch for which the model's predicted tokens exactly matched the true completion tokens. For example, if the batch size is set to 3 and your data contains completions [[1, 2], [0, 5], [4, 2]], this value is set to 0.67 (2 of 3) if the model predicted [[1, 1], [0, 5], [4, 2]].
validation_mean_token_accuracy	The percentage of tokens in the validation batch correctly predicted by the model. For example, if the batch size is set to 3 and your data contains completions [[1, 2], [0, 5], [4, 2]], this value is set to 0.83 (5 of 6) if the model predicted [[1, 1], [0, 5], [4, 2]].

You can also view the data in your results.csv file as plots in Azure OpenAI Studio. Select the link for your trained model, and you will see three charts: loss, mean token accuracy, and token accuracy. If you provided validation data, both datasets will appear on the same plot.

Look for your loss to decrease over time, and your accuracy to increase. If you see a divergence between your training and validation data, that may indicate that you are overfitting. Try training with fewer epochs, or a smaller learning rate multiplier.

When to use Azure OpenAI fine-tuning

When deciding whether or not fine-tuning is the right solution to explore for a given use case, there are some key terms that it's helpful to be familiar with:

- Prompt Engineering is a technique that involves designing prompts for natural language processing models. This process improves accuracy and relevancy in responses, optimizing the performance of the model.
- Retrieval Augmented Generation (RAG) improves Large Language Model (LLM) performance by retrieving data from external sources and incorporating it into a prompt. RAG allows businesses to achieve customized solutions while maintaining data relevance and optimizing costs.
- Fine-tuning retrains an existing Large Language Model using example data, resulting in a new "custom" Large Language Model that has been optimized using the provided examples.

What is Fine Tuning with Azure OpenAI?

When we talk about fine tuning, we really mean *supervised fine-tuning* not continuous pre-training or Reinforcement Learning through Human Feedback (RLHF). Supervised fine-tuning refers to the process of retraining pre-trained models on specific datasets, typically to improve model performance on specific tasks or introduce information that wasn't well represented when the base model was originally trained.

Fine-tuning is an advanced technique that requires expertise to use appropriately. The questions below will help you evaluate whether you're ready for fine-tuning, and how well you've thought through the process. You can use these to guide your next steps or identify other approaches that might be more appropriate.

Why do you want to fine-tune a model?

- You should be able to clearly articulate a specific use case for fine-tuning and identify the model you hope to fine-tune.
- Good use cases for fine-tuning include steering the model to output content in a specific and customized style, tone, or format, or scenarios where the information needed to steer the model is too long or complex to fit into the prompt window.

Common signs you might not be ready for fine-tuning yet:

- No clear use case for fine tuning, or an inability to articulate much more than “I want to make a model better”.
- If you identify cost as your primary motivator, proceed with caution. Fine-tuning might reduce costs for certain use cases by shortening prompts or allowing you

to use a smaller model but there's a higher upfront cost to training and you'll have to pay for hosting your own custom model.

- If you want to add out of domain knowledge to the model, you should start with retrieval augmented generation (RAG) with features like Azure OpenAI's on your data or embeddings. Often, this is a cheaper, more adaptable, and potentially more effective option depending on the use case and data.

What have you tried so far?

Fine-tuning is an advanced capability, not the starting point for your generative AI journey. You should already be familiar with the basics of using Large Language Models (LLMs). You should start by evaluating the performance of a base model with prompt engineering and/or Retrieval Augmented Generation (RAG) to get a baseline for performance.

Having a baseline for performance without fine-tuning is essential for knowing whether or not fine-tuning has improved model performance. Fine-tuning with bad data makes the base model worse, but without a baseline, it's hard to detect regressions.

If you are ready for fine-tuning you:

- Should be able to demonstrate evidence and knowledge of Prompt Engineering and RAG based approaches.
- Be able to share specific experiences and challenges with techniques other than fine-tuning that were already tried for your use case.
- Need to have quantitative assessments of baseline performance, whenever possible.

Common signs you might not be ready for fine-tuning yet:

- Starting with fine-tuning without having tested any other techniques.
- Insufficient knowledge or understanding on how fine-tuning applies specifically to Large Language Models (LLMs).
- No benchmark measurements to assess fine-tuning against.

What isn't working with alternate approaches?

Understanding where prompt engineering falls short should provide guidance on going about your fine-tuning. Is the base model failing on edge cases or exceptions? Is the base model not consistently providing output in the right format, and you can't fit enough examples in the context window to fix it?

Examples of failure with the base model and prompt engineering will help you identify the data they need to collect for fine-tuning, and how you should be evaluating your fine-tuned model.

Here's an example: A customer wanted to use GPT-3.5-Turbo to turn natural language questions into queries in a specific, non-standard query language. They provided guidance in the prompt ("Always return GQL") and used RAG to retrieve the database schema. However, the syntax wasn't always correct and often failed for edge cases. They collected thousands of examples of natural language questions and the equivalent queries for their database, including cases where the model had failed before – and used that data to fine-tune the model. Combining their new fine-tuned model with their engineered prompt and retrieval brought the accuracy of the model outputs up to acceptable standards for use.

If you are ready for fine-tuning you:

- Have clear examples on how you have approached the challenges in alternate approaches and what's been tested as possible resolutions to improve performance.
- You've identified shortcomings using a base model, such as inconsistent performance on edge cases, inability to fit enough few shot prompts in the context window to steer the model, high latency, etc.

Common signs you might not be ready for fine-tuning yet:

- Insufficient knowledge from the model or data source.
- Inability to find the right data to serve the model.

What data are you going to use for fine-tuning?

Even with a great use case, fine-tuning is only as good as the quality of the data that you're able to provide. You need to be willing to invest the time and effort to make fine-tuning work. Different models will require different data volumes but you often need to be able to provide fairly large quantities of high-quality curated data.

Another important point is even with high quality data if your data isn't in the necessary format for fine-tuning you'll need to commit engineering resources in order to properly format the data.

Data	**Babbage-002 & Davinci-002**	**GPT-3.5-Turbo**
Volume	Thousands of Examples	Thousands of Examples

Format	Prompt/Completion	Conversational Chat

If you are ready for fine-tuning you:

- Have identified a dataset for fine-tuning.
- The dataset is in the appropriate format for training.
- Some level of curation has been employed to ensure dataset quality.

Common signs you might not be ready for fine-tuning yet:

- Dataset hasn't been identified yet.
- Dataset format doesn't match the model you wish to fine-tune.

How will you measure the quality of your fine-tuned model?

There isn't a single right answer to this question, but you should have clearly defined goals for what success with fine-tuning looks like. Ideally, this shouldn't just be qualitative but should include quantitative measures of success like utilizing a holdout set of data for validation, as well as user acceptance testing or A/B testing the fine-tuned model against a base model.