Tell us about your PDF experience.

Azure OpenAI Documentation

Learn how to use Azure OpenAI's powerful language models including the GPT-3, Codex and Embeddings model series for content generation, summarization, semantic search, and natural language to code translation.

OVERVIEW
What is Azure OpenAI?

QUICKSTART
Quickstarts

HOW-TO GUIDE
Create a resource

TUTORIAL
Embeddings

HOW-TO GUIDE
Completions

HOW-TO GUIDE
Work with code (Codex)

CONCEPT
Azure OpenAI Models

REFERENCE
Support and help options

Additional resources

Azure OpenAI

- Azure OpenAI Studio
- Region support
- Quotas and limits
- Apply for access to Azure OpenAI

Video

- Combining OpenAI models with the power of Azure

Reference

REST API

Terms of use

Tools

Azure CLI

PowerShell

What is Azure OpenAI?

Article • 02/10/2023 • 6 minutes to read

The Azure OpenAI service provides REST API access to OpenAI's powerful language models including the GPT-3, Codex and Embeddings model series. These models can be easily adapted to your specific task including but not limited to content generation, summarization, semantic search, and natural language to code translation. Users can access the service through REST APIs, Python SDK, or our web-based interface in the Azure OpenAI Studio.

Features overview

Feature	Azure OpenAI
Models available	GPT-3 base series Codex series Embeddings series Learn more in our Models page.
Fine-tuning	Ada Babbage Curie Cushman* Davinci* * Currently unavailable.
Price	Available here
Virtual network support	Yes
Managed Identity	Yes, via Azure Active Directory
UI experience	**Azure Portal** for account & resource management, **Azure OpenAI Service Studio** for model exploration and fine tuning
Regional availability	East US South Central US West Europe
Content filtering	Prompts and completions are evaluated against our content policy with automated systems. High severity content will be filtered.

Responsible AI

At Microsoft, we're committed to the advancement of AI driven by principles that put people first. Generative models such as the ones available in the Azure OpenAI service have significant potential benefits, but without careful design and thoughtful mitigations, such models have the potential to generate incorrect or even harmful content. Microsoft has made significant investments to help guard against abuse and unintended harm, which includes requiring applicants to show well-defined use cases, incorporating Microsoft's principles for responsible AI use, building content filters to support customers, and providing responsible AI implementation guidance to onboarded customers.

How do I get access to Azure OpenAI?

How do I get access to Azure OpenAI Service?

Access is currently limited as we navigate high demand, upcoming product improvements, and Microsoft's commitment to responsible AI. For now, we're working with customers with an existing partnership with Microsoft, lower risk use cases, and those committed to incorporating mitigations. In addition to applying for initial access, all solutions using the Azure OpenAI service are required to go through a use case review before they can be released for production use.

More specific information is included in the application form. We appreciate your patience as we work to responsibly enable broader access to the Azure OpenAI service.

Apply here for initial access or for a production review:

Apply now

All solutions using the Azure OpenAI service are also required to go through a use case review before they can be released for production use, and are evaluated on a case-by-case basis. In general, the more sensitive the scenario the more important risk mitigation measures will be for approval.

Comparing Azure OpenAI and OpenAI

Azure OpenAI Service gives customers advanced language AI with OpenAI GPT-3, Codex, and DALL-E models with the security and enterprise promise of Azure. Azure OpenAI co-develops the APIs with OpenAI, ensuring compatibility and a smooth transition from one to the other.

With Azure OpenAI, customers get the security capabilities of Microsoft Azure while running the same models as OpenAI. Azure OpenAI offers private networking, regional availability, and responsible AI content filtering.

Key concepts

Prompts & Completions

The completions endpoint is the core component of the API service. This API provides access to the model's text-in, text-out interface. Users simply need to provide an input **prompt** containing the English text command, and the model will generate a text **completion**.

Here's an example of a simple prompt and completion:

> **Prompt**: `""" count to 5 in a for loop """`
>
> **Completion**: `for i in range(1, 6): print(i)`

Tokens

OpenAI Enterprise processes text by breaking it down into tokens. Tokens can be words or just chunks of characters. For example, the word “hamburger” gets broken up into the tokens “ham”, “bur” and “ger”, while a short and common word like “pear” is a single token. Many tokens start with a whitespace, for example “ hello” and “ bye”.

The total number of tokens processed in a given request depends on the length of your input, output and request parameters. The quantity of tokens being processed will also affect your response latency and throughput for the models.

Resources

The Azure OpenAI service is a new product offering on Azure. You can get started with the Azure OpenAI service the same way as any other Azure product where you create a resource, or instance of the service, in your Azure Subscription. You can read more about Azure's resource management design.

Deployments

Once you create an Azure OpenAI Resource, you must deploy a model before you can start making API calls and generating text. This action can be done using the

Deployment APIs. These APIs allow you to specify the model you wish to use.

In-context learning

The models used by the Azure OpenAI service use natural language instructions and examples provided during the generation call to identify the task being asked and skill required. When you use this approach, the first part of the prompt includes natural language instructions and/or examples of the specific task desired. The model then completes the task by predicting the most probable next piece of text. This technique is known as "in-context" learning. These models aren't retrained during this step but instead give predictions based on the context you include in the prompt.

There are three main approaches for in-context learning: Few-shot, one-shot and zero-shot. These approaches vary based on the amount of task-specific data that is given to the model:

Few-shot: In this case, a user includes several examples in the call prompt that demonstrate the expected answer format and content. The following example shows a few-shot prompt where we provide multiple examples (the model will generate the last answer):

```
    Convert the questions to a command:
    Q: Ask Constance if we need some bread.
    A: send-msg `find constance` Do we need some bread?
    Q: Send a message to Greg to figure out if things are ready for
Wednesday.
    A: send-msg `find greg` Is everything ready for Wednesday?
    Q: Ask Ilya if we're still having our meeting this evening.
    A: send-msg `find ilya` Are we still having a meeting this evening?
    Q: Contact the ski store and figure out if I can get my skis fixed
before I leave on Thursday.
    A: send-msg `find ski store` Would it be possible to get my skis fixed
before I leave on Thursday?
    Q: Thank Nicolas for lunch.
    A: send-msg `find nicolas` Thank you for lunch!
    Q: Tell Constance that I won't be home before 19:30 tonight — unmovable
meeting.
    A: send-msg `find constance` I won't be home before 19:30 tonight. I
have a meeting I can't move.
    Q: Tell John that I need to book an appointment at 10:30.
    A:
```

The number of examples typically range from 0 to 100 depending on how many can fit in the maximum input length for a single prompt. Maximum input length can vary depending on the specific models you use. Few-shot learning enables a major reduction

in the amount of task-specific data required for accurate predictions. This approach will typically perform less accurately than a fine-tuned model.

One-shot: This case is the same as the few-shot approach except only one example is provided.

Zero-shot: In this case, no examples are provided to the model and only the task request is provided.

Models

The service provides users access to several different models. Each model provides a different capability and price point. The GPT-3 base models are known as Davinci, Curie, Babbage, and Ada in decreasing order of capability and increasing order of speed.

The Codex series of models is a descendant of GPT-3 and has been trained on both natural language and code to power natural language to code use cases. Learn more about each model on our models concept page.

Next steps

Learn more about the underlying models that power Azure OpenAI.

Additional resources

Azure OpenAI service quotas and limits

Article • 01/16/2023 • 2 minutes to read

This article contains a quick reference and a detailed description of the quotas and limits for the Azure OpenAI service in Azure Cognitive Services.

Quotas and limits reference

The following sections provide you with a quick guide to the quotas and limits that apply to the Azure OpenAI service

Limit Name	Limit Value
OpenAI resources per region	2
Requests per second per deployment	20 requests per second for: text-davinci-003, text-davinci-002, text-davinci-fine-tune-002, code-cushman-002, code-davinci-002, code-davinci-fine-tune-002 50 requests per second for all other text models.
Max fine-tuned model deployments	2
Ability to deploy same model to multiple deployments	Not allowed
Total number of training jobs per resource	100
Max simultaneous running training jobs per resource	1
Max training jobs queued	20
Max Files per resource	50
Total size of all files per resource	1 GB
Max training job time (job will fail if exceeded)	120 hours

Limit Name	Limit Value
Max training job size (tokens in training file * # of epochs)	**Ada**: 40-M tokens **Babbage**: 40-M tokens **Curie**: 40-M tokens **Cushman**: 40-M tokens **Davinci**: 10-M

General best practices to mitigate throttling during autoscaling

To minimize issues related to throttling, it's a good idea to use the following techniques:

- Implement retry logic in your application.
- Avoid sharp changes in the workload. Increase the workload gradually.
- Test different load increase patterns.
- Create another OpenAI service resource in the same or different regions, and distribute the workload among them.

The next sections describe specific cases of adjusting quotas.

Request an increase to a limit on transactions-per-second or number of fine-tuned models deployed

The limit of concurrent requests defines how high the service can scale before it starts to throttle your requests.

Have the required information ready

- OpenAI Resource ID
- Region
- Deployment Name

How to get this information:

1. Go to the Azure portal.
2. Select the Azure OpenAI resource for which you would like to increase the request limit.
3. From the **Resource Management** group, select **Properties**.
4. Copy and save the values of the following fields:

    - **Resource ID**

- **Location** (your endpoint region)

5. From the **Resource Management** group, select **Deployments**.

    - Copy and save the name of the Deployment you're requesting a limit increase

Create and submit a support request

Initiate the increase of the limit for concurrent requests for your resource, or if necessary check the current limit, by submitting a support request. Here's how:

1. Ensure you have the required information listed in the previous section.
2. Go to the Azure portal.
3. Select the OpenAI service resource for which you would like to increase (or to check) the concurrency request limit.
4. In the **Support + troubleshooting** group, select **New support request**. A new window will appear, with auto-populated information about your Azure subscription and Azure resource.
5. In **Summary**, describe what you want (for example, "Increase OpenAI request limit").
6. In **Problem type**, select **Quota or Subscription issues**.
7. In **Problem subtype**, select **Increasing limits or access to specific functionality**
8. Select **Next: Solutions**. Proceed further with the request creation.
9. On the **Details** tab, in the **Description** field, enter the following:

    - Include details on which limit you're requesting an increase for.
    - The Azure resource information you collected previously.
    - Any other required information.

10. On the **Review + create** tab, select **Create**.
11. Note the support request number in Azure portal notifications. You'll be contacted shortly about your request.

Next steps

Learn more about the underlying models that power Azure OpenAI.

Additional resources

Documentation

Azure OpenAI models - Azure OpenAI

Learn about the different models that are available in Azure OpenAI.

Quickstart - Deploy a model and generate text using Azure OpenAI - Azure OpenAI

Walkthrough on how to get started with Azure OpenAI and make your first completions call.

Azure OpenAI embeddings - Azure OpenAI - embeddings and cosine similarity

Learn more about Azure OpenAI embeddings API for document search and cosine similarity

Azure OpenAI content filtering - Azure OpenAI

Learn about the content filtering capabilities of the OpenAI service in Azure Cognitive Services

What's new in Azure OpenAI? - Azure Cognitive Services

Learn about the latest news and features updates for Azure OpenAI

How to customize a model with Azure OpenAI - Azure OpenAI

Learn how to create your own customized model with Azure OpenAI

How-to - Create a resource and deploy a model using Azure OpenAI - Azure OpenAI

Walkthrough on how to get started with Azure OpenAI and make your first resource and deploy your first model.

Azure OpenAI REST API reference - Azure OpenAI

Learn how to use the Azure OpenAI REST API. In this article, you'll learn about authorization options, how to structure a request and receive a response.

Show 5 more

Azure OpenAI models

Article • 02/10/2023 • 8 minutes to read

The service provides access to many different models, grouped by family and capability. A model family typically associates models by their intended task. The following table describes model families currently available in Azure OpenAI. Not all models are available in all regions currently. Please refer to the capability table at the bottom for a full breakdown.

Model family	Description
GPT-3	A series of models that can understand and generate natural language.
Codex	A series of models that can understand and generate code, including translating natural language to code.
Embeddings	A set of models that can understand and use embeddings. An embedding is a special format of data representation that can be easily utilized by machine learning models and algorithms. The embedding is an information dense representation of the semantic meaning of a piece of text. Currently, we offer three families of Embeddings models for different functionalities: similarity, text search, and code search.

Model capabilities

Each model family has a series of models that are further distinguished by capability. These capabilities are typically identified by names, and the alphabetical order of these names generally signifies the relative capability and cost of that model within a given model family. For example, GPT-3 models use names such as Ada, Babbage, Curie, and Davinci to indicate relative capability and cost. Davinci is more capable (at a higher cost) than Curie, which in turn is more capable (at a higher cost) than Babbage, and so on.

> ⓘ Note
>
> Any task that can be performed by a less capable model like Ada can be performed by a more capable model like Curie or Davinci.

Naming convention

Azure OpenAI's model names typically correspond to the following standard naming convention:

`{family}-{capability}[-{input-type}]-{identifier}`

Element	Description
`{family}`	The model family of the model. For example, GPT-3 models uses `text`, while Codex models use `code`.
`{capability}`	The relative capability of the model. For example, GPT-3 models include `ada`, `babbage`, `curie`, and `davinci`.
`{input-type}`	(Embeddings models only) The input type of the embedding supported by the model. For example, text search embedding models support `doc` and `query`.
`{identifier}`	The version identifier of the model.

For example, our most powerful GPT-3 model is called `text-davinci-003`, while our most powerful Codex model is called `code-davinci-002`.

> Older versions of the GPT-3 models are available, named `ada`, `babbage`, `curie`, and `davinci`. These older models do not follow the standard naming conventions, and they are primarily intended for fine tuning. For more information, see Learn how to customize a model for your application.

Finding what models are available

You can easily see the models you have available for both inference and fine-tuning in your resource by using the Models API.

Finding the right model

We recommend starting with the most capable model in a model family because it's the best way to understand what the service is capable of. After you have an idea of what you want to accomplish, you can either stay with that model or move to a model with lower capability and cost, optimizing around that model's capabilities.

GPT-3 models

The GPT-3 models can understand and generate natural language. The service offers four model capabilities, each with different levels of power and speed suitable for

different tasks. Davinci is the most capable model, while Ada is the fastest. The following list represents the latest versions of GPT-3 models, ordered by increasing capability.

- `text-ada-001`
- `text-babbage-001`
- `text-curie-001`
- `text-davinci-003`

While Davinci is the most capable, the other models provide significant speed advantages. Our recommendation is for users to start with Davinci while experimenting, because it will produce the best results and validate the value our service can provide. Once you have a prototype working, you can then optimize your model choice with the best latency/performance balance for your application.

Davinci

Davinci is the most capable model and can perform any task the other models can perform, often with less instruction. For applications requiring deep understanding of the content, like summarization for a specific audience and creative content generation, Davinci produces the best results. The increased capabilities provided by Davinci require more compute resources, so Davinci costs more and isn't as fast as other models.

Another area where Davinci excels is in understanding the intent of text. Davinci is excellent at solving many kinds of logic problems and explaining the motives of characters. Davinci has been able to solve some of the most challenging AI problems involving cause and effect.

Use for: Complex intent, cause and effect, summarization for audience

Curie

Curie is powerful, yet fast. While Davinci is stronger when it comes to analyzing complicated text, Curie is capable for many nuanced tasks like sentiment classification and summarization. Curie is also good at answering questions and performing Q&A and as a general service chatbot.

Use for: Language translation, complex classification, text sentiment, summarization

Babbage

Babbage can perform straightforward tasks like simple classification. It’s also capable when it comes to semantic search, ranking how well documents match up with search

queries.

Use for: Moderate classification, semantic search classification

Ada

Ada is usually the fastest model and can perform tasks like parsing text, address correction and certain kinds of classification tasks that don't require too much nuance. Ada's performance can often be improved by providing more context.

Use for: Parsing text, simple classification, address correction, keywords

Codex models

The Codex models are descendants of our base GPT-3 models that can understand and generate code. Their training data contains both natural language and billions of lines of public code from GitHub.

They're most capable in Python and proficient in over a dozen languages, including C#, JavaScript, Go, Perl, PHP, Ruby, Swift, TypeScript, SQL, and even Shell. The following list represents the latest versions of Codex models, ordered by increasing capability.

- `code-cushman-001`
- `code-davinci-002`

Davinci

Similar to GPT-3, Davinci is the most capable Codex model and can perform any task the other models can perform, often with less instruction. For applications requiring deep understanding of the content, Davinci produces the best results. These increased capabilities require more compute resources, so Davinci costs more and isn't as fast as other models.

Cushman

Cushman is powerful, yet fast. While Davinci is stronger when it comes to analyzing complicated tasks, Cushman is a capable model for many code generation tasks. Cushman typically runs faster and cheaper than Davinci, as well.

Embeddings models

Currently, we offer three families of Embeddings models for different functionalities:

- Similarity
- Text search
- Code search

Each family includes models across a range of capability. The following list indicates the length of the numerical vector returned by the service, based on model capability:

- Ada: 1024 dimensions
- Babbage: 2048 dimensions
- Curie: 4096 dimensions
- Davinci: 12288 dimensions

Davinci is the most capable, but is slower and more expensive than the other models. Ada is the least capable, but is both faster and cheaper.

Similarity embedding

These models are good at capturing semantic similarity between two or more pieces of text.

Use cases	Models

Clustering, regression, anomaly detection, visualization	`text-similarity-ada-001`
	`text-similarity-babbage-001`
	`text-similarity-curie-001`
	`text-similarity-davinci-001`

Text search embedding

These models help measure whether long documents are relevant to a short search query. There are two input types supported by this family: `doc`, for embedding the documents to be retrieved, and `query`, for embedding the search query.

Use cases	Models

Use cases	Models

Search, context relevance, information retrieval	`text-search-ada-doc-001`
	`text-search-ada-query-001`
	`text-search-babbage-doc-001`
	`text-search-babbage-query-001`
	`text-search-curie-doc-001`
	`text-search-curie-query-001`
	`text-search-davinci-doc-001`
	`text-search-davinci-query-001`

Code search embedding

Similar to text search embedding models, there are two input types supported by this family: `code`, for embedding code snippets to be retrieved, and `text`, for embedding natural language search queries.

Use cases	Models

Code search and relevance	`code-search-ada-code-001`
	`code-search-ada-text-001`
	`code-search-babbage-code-001`
	`code-search-babbage-text-001`

When using our Embeddings models, keep in mind their limitations and risks.

Model Summary table and region availability

GPT-3 Models

Model	Supports Completions	Supports Embeddings	Base model Regions	Fine-Tuning Regions
Ada	Yes	No	N/A	East US, South Central US, West Europe
Text-Ada-001	Yes	No	East US, South Central US, West Europe	N/A
Babbage	Yes	No	N/A	East US, South Central US, West Europe
Text-Babbage-001	Yes	No	East US, South Central US, West Europe	N/A

Model	Supports Completions	Supports Embeddings	Base model Regions	Fine-Tuning Regions
Curie	Yes	No	N/A	East US, South Central US, West Europe
Text-curie-001	Yes	No	East US, South Central US, West Europe	N/A
Davinci*	Yes	No	N/A	East US, South Central US, West Europe
Text-davinci-001	Yes	No	South Central US, West Europe	N/A
Text-davinci-002	Yes	No	East US, South Central US, West Europe	N/A
Text-davinci-003	Yes	No	East US	N/A
Text-davinci-fine-tune-002*	Yes	No	N/A	East US, West Europe

*Models available by request only. We are currently unable to onboard new customers at this time.

Codex Models

Model	Supports Completions	Supports Embeddings	Base model Regions	Fine-Tuning Regions
Code-Cushman-001*	Yes	No	South Central US, West Europe	East US, South Central US, West Europe
Code-Davinci-002	Yes	No	East US, West Europe	N/A
Code-Davinci-Fine-tune-002*	Yes	No	N/A	East US, West Europe

*Models available for Fine-tuning by request only. We are currently unable to enable new cusetomers at this time.

Embeddings Models

Model	Supports Completions	Supports Embeddings	Base model Regions	Fine-Tuning Regions
text-ada-embeddings-002	No	Yes	East US, South Central US, West Europe	N/A
text-similarity-ada-001	No	Yes	East US, South Central US, West Europe	N/A
text-similarity-babbage-001	No	Yes	South Central US, West Europe	N/A
text-similarity-curie-001	No	Yes	East US, South Central US, West Europe	N/A
text-similarity-davinci-001	No	Yes	South Central US, West Europe	N/A
text-search-ada-doc-001	No	Yes	South Central US, West Europe	N/A
text-search-ada-query-001	No	Yes	South Central US, West Europe	N/A
text-search-babbage-doc-001	No	Yes	South Central US, West Europe	N/A
text-search-babbage-query-001	No	Yes	South Central US, West Europe	N/A
text-search-curie-doc-001	No	Yes	South Central US, West Europe	N/A
text-search-curie-query-001	No	Yes	South Central US, West Europe	N/A
text-search-davinci-doc-001	No	Yes	South Central US, West Europe	N/A
text-search-davinci-query-001	No	Yes	South Central US, West Europe	N/A
code-search-ada-code-001	No	Yes	South Central US, West Europe	N/A
code-search-ada-text-001	No	Yes	South Central US, West Europe	N/A
code-search-babbage-code-001	No	Yes	South Central US, West Europe	N/A

Model	Supports Completions	Supports Embeddings	Base model Regions	Fine-Tuning Regions
code-search-babbage-text-001	No	Yes	South Central US, West Europe	N/A

Next steps

Learn more about Azure OpenAI.

Additional resources

What's new in Azure OpenAI

Article • 02/10/2023 • 2 minutes to read

January 2023

New Features

- **Service GA**. Azure OpenAI is now generally available.
- **New models**: Addition of the latest text model, text-davinci-003 (East US, West Europe), text-ada-embeddings-002 (East US, South Central US, West Europe)

December 2022

New features

- **The latest models from OpenAI.** Azure OpenAI provides access to all the latest models including the GPT-3.5 series.
- **New API version (2022-12-01).** This update includes several requested enhancements including token usage information in the API response, improved error messages for files, alignment with OpenAI on fine-tuning creation data structure, and support for the suffix parameter to allow custom naming of fine-tuned jobs.
- **Higher request per second limits.** 50 for non-Davinci models. 20 for Davinci models.
- **Faster fine-tune deployments.** Deploy an Ada and Curie fine-tuned models in under 10 minutes.
- **Higher training limits:** 40M training tokens for Ada, Babbage, and Curie. 10M for Davinci.
- **Process for requesting modifications to the abuse & miss-use data logging & human review.** Today, the service logs request/response data for the purposes of abuse and misuse detection to ensure that these powerful models aren't abused. However, many customers have strict data privacy and security requirements that require greater control over their data. To support these use cases, we're releasing a new process for customers to modify the content filtering policies or turn off the

abuse logging for low-risk use cases. This process follows the established Limited Access process within Azure Cognitive Services and existing OpenAI customers can apply here.

- **Customer managed key (CMK) encryption.** CMK provides customers greater control over managing their data in the Azure OpenAI Service by providing their own encryption keys used for storing training data and customized models. Customer-managed keys (CMK), also known as bring your own key (BYOK), offer greater flexibility to create, rotate, disable, and revoke access controls. You can also audit the encryption keys used to protect your data. Learn more from our encryption at rest documentation.
- **Lockbox support**
- **SOC-2 compliance**
- **Logging and diagnostics** through Azure Resource Health, Cost Analysis, and Metrics & Diagnostic settings.
- **Studio improvements.** Numerous usability improvements to the Studio workflow including Azure AD role support to control who in the team has access to create fine-tuned models and deploy.

Changes (breaking)

Fine-tuning create API request has been updated to match OpenAI's schema.

Preview API versions:

JSON

```
{
"training_file": "file-XGinujblHPwGLSztz8cPS8XY" ,
"hyperparams": {
               "batch_size": 4,
               "learning_rate_multiplier": 0.1,
               "n_epochs": 4,
               "prompt_loss_weight": 0.1,
               }
}
```

API version 2022-12-01:

JSON

```
{
"training_file": "file-XGinujblHPwGLSztz8cPS8XY" ,
"batch_size": 4,
“learning_rate_multiplier": 0.1,
"n_epochs": 4,
"prompt_loss_weight": 0.1,
}
```

Content filtering is temporarily off by default. Azure content moderation works differently than OpenAI. Azure OpenAI runs content filters during the generation call to detect harmful or abusive content and filters them from the response. Learn More

These models will be re-enabled in Q1 2023 and be on by default.

Customer actions

- Contact Azure Support if you would like these turned on for your subscription.
- Apply for filtering modifications, if you would like to have them remain off. (This option will be for low-risk use cases only.)

Next steps

Learn more about the underlying models that power Azure OpenAI.

Additional resources

Quickstart: Get started generating text using Azure OpenAI

Article • 02/03/2023 • 9 minutes to read

Use this article to get started making your first calls to the Azure OpenAI service.

Prerequisites

- An Azure subscription - Create one for free
- Access granted to the Azure OpenAI service in the desired Azure subscription

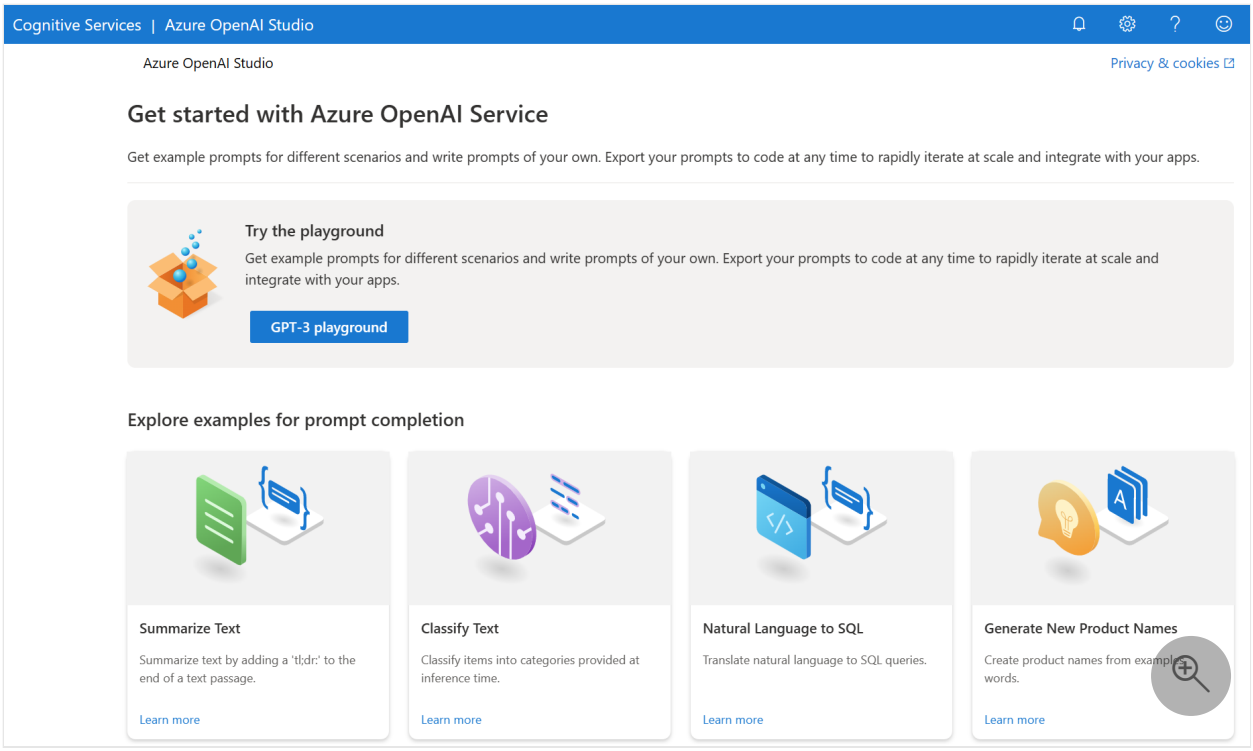
  Currently, access to this service is granted only by application. You can apply for access to the Azure OpenAI service by completing the form at https://aka.ms/oai/access. Open an issue on this repo to contact us if you have an issue.

- An Azure OpenAI Service resource with a model deployed. For more information about model deployment, see the resource deployment guide.

Go to the Azure OpenAI Studio

Navigate to the Azure OpenAI Studio at https://oai.azure.com/ and sign-in with credentials that have access to your OpenAI resource. During or after the sign-in workflow, select the appropriate directory, Azure subscription, and Azure OpenAI resource.

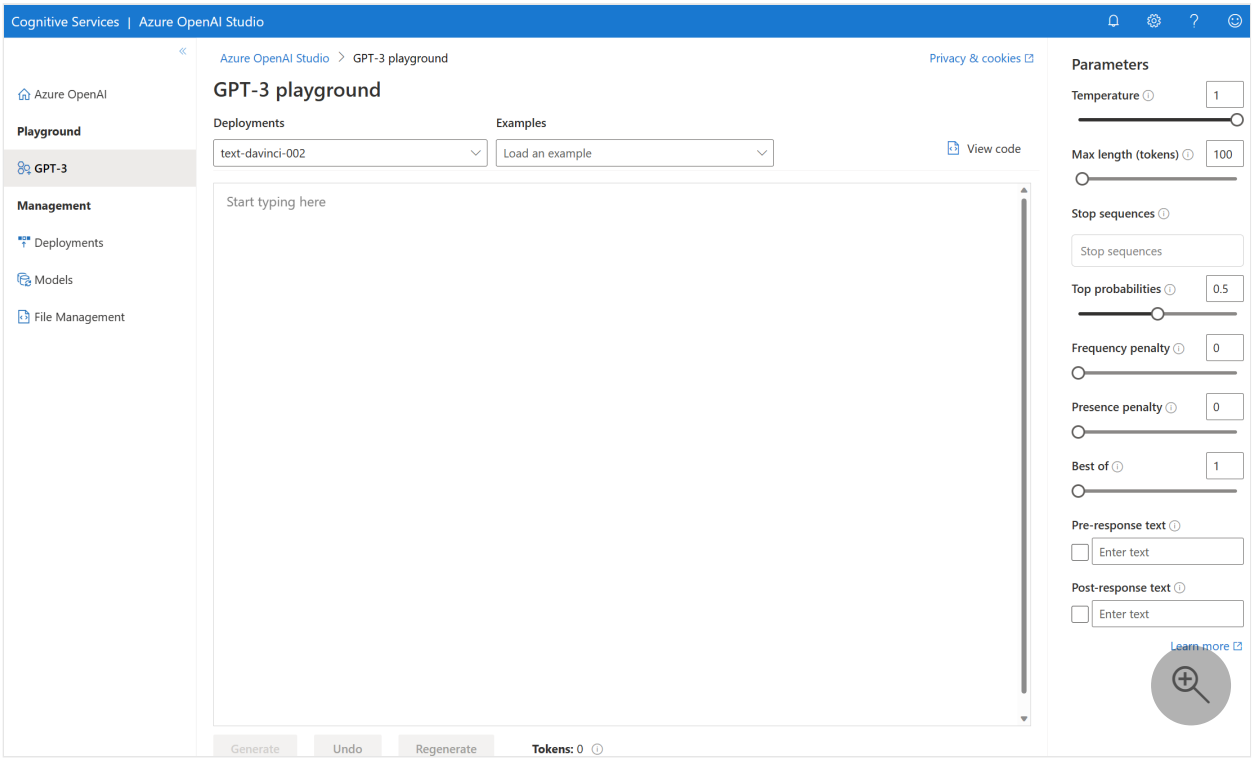
From the Azure OpenAI Studio landing page navigate further to explore examples for prompt completion, manage your deployments and models, and find learning resources such as documentation and community forums.



Go to the Playground for experimentation and fine-tuning workflow.

Playground

Start exploring OpenAI capabilities with a no-code approach through the GPT-3 Playground. It's simply a text box where you can submit a prompt to generate a completion. From this page, you can quickly iterate and experiment with the capabilities.



You can select a deployment and choose from a few pre-loaded examples to get started. If your resource doesn't have a deployment, select **Create a deployment** and follow the instructions provided by the wizard. For more information about model deployment, see the resource deployment guide.

You can experiment with the configuration settings such as temperature and pre-response text to improve the performance of your task. You can read more about each parameter in the REST API.

- Selecting the **Generate** button will send the entered text to the completions API and stream the results back to the text box.
- Select the **Undo** button to undo the prior generation call.
- Select the **Regenerate** button to complete an undo and generation call together.

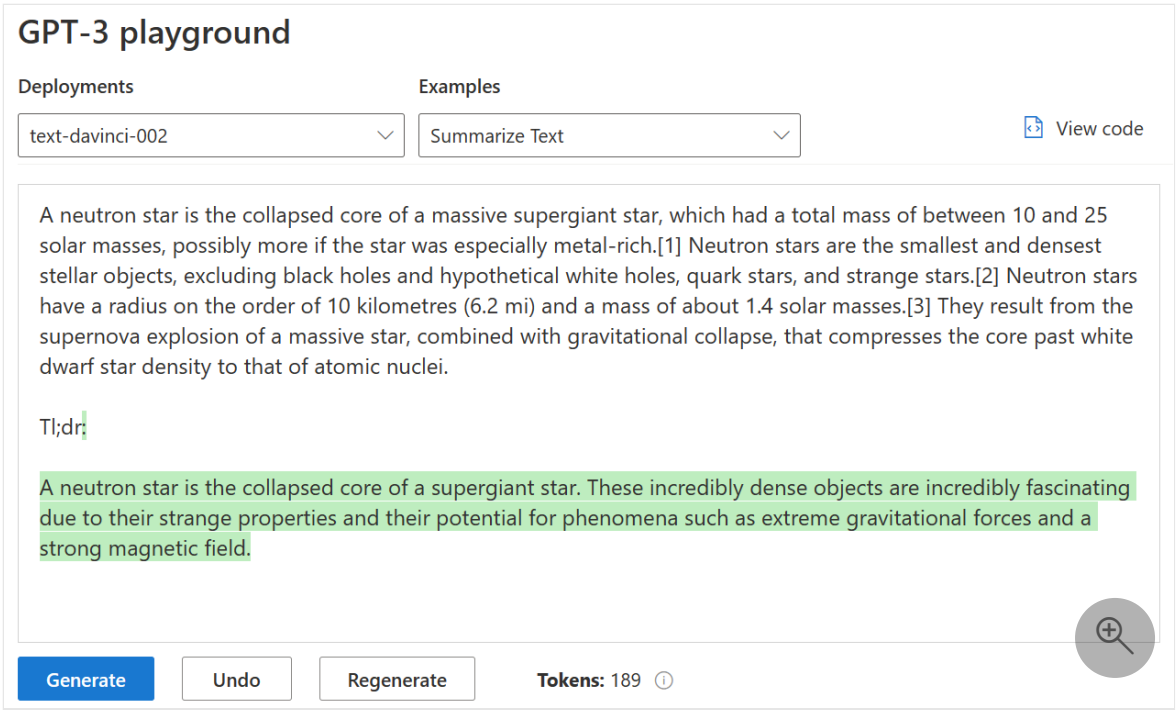
The Azure OpenAI Service also performs content moderation on the prompt inputs and generated outputs. The prompts or responses may be filtered if harmful content is detected. For more information, see the content filter article.

In the GPT-3 playground you can also view Python and curl code samples pre-filled according to your selected settings. Just select **View code** next to the examples dropdown. You can write an application to complete the same task with the OpenAI Python SDK, curl, or other REST API client.

Try text summarization

To use the OpenAI service for text summarization in the GPT-3 Playground, follow these steps:

1. Sign in to the Azure OpenAI Studio.
2. Select the subscription and OpenAI resource to work with.
3. Select **GPT-3 Playground** at the top of the landing page.
4. Select your deployment from the **Deployments** dropdown. If your resource doesn't have a deployment, select **Create a deployment** and then revisit this step.
5. Select **Summarize Text** from the **Examples** dropdown.



6. Select `Generate`. OpenAI will grasp the context of text and rephrase it succinctly. You should get a result that resembles the following text:

```
Tl;dr A neutron star is the collapsed core of a supergiant star. These
incredibly dense objects are incredibly fascinating due to their
strange properties and their potential for phenomena such as extreme
gravitational forces and a strong magnetic field.
```

The accuracy of the response can vary per model. The Davinci based model in this example is well-suited to this type of summarization, whereas a Codex based model wouldn't perform as well at this particular task.

Clean up resources

If you want to clean up and remove an OpenAI resource, you can delete the resource or resource group. Deleting the resource group also deletes any other resources associated with it.

- Portal
- Azure CLI

Next steps

Learn more about how to generate the best completion in our How-to guide on completions.

Additional resources

Content filtering

Article • 12/08/2022 • 4 minutes to read

Azure OpenAI Service includes a content management system that works alongside core models to filter content. This system works by running both the input prompt and generated content through an ensemble of classification models aimed at detecting misuse. If the system identifies harmful content, you'll receive either an error on the API call if the prompt was deemed inappropriate or the finish_reason on the response will be `content_filter` to signify that some of the generation was filtered.

> **Note**
>
> This content filtering system is temporarily turned off while we work on some improvements. The internal system is still annotating harmful content but the models will not block. Content filtering will be reactivated with the release of upcoming updates. If you would like to enable the content filters at any point before that, please open an **Azure support request**.

You can generate content with the completions API using many different configurations that will alter the filtering behavior you should expect. The following section aims to enumerate all of these scenarios for you to appropriately design your solution.

To ensure you have properly mitigated risks in your application, you should evaluate all potential harms carefully, follow guidance in the Transparency Note and add scenario-specific mitigation as needed.

Scenario details

When building your application, you'll want to account for scenarios where the content returned by the Completions API is filtered and content may not be complete. How you act on this information will be application specific. The behavior can be summarized in the following key points:

- Prompts that are deemed inappropriate will return an HTTP 400 error
- Non-streaming completions calls won't return any content when the content is filtered. The `finish_reason` value will be set to `content_filter`. In rare cases with long responses, a partial result can be returned. In these cases, the `finish_reason` will be updated.

- For streaming completions calls, segments will be returned back to the user as they're completed. The service will continue streaming until either reaching a stop token, length or harmful content is detected.

Scenario: You send a non-streaming completions call asking for multiple generations with no inappropriate content

The table below outlines the various ways content filtering can appear:

HTTP response code	Response behavior
200	In the cases when all generation passes the filter models no content moderation details are added to the response. The finish_reason for each generation will be either stop or length.

Example response:

JSON

```
{
    "prompt":"Text example"
    , "n": 3
    , "stream": false
}
```

Example response JSON:

JSON

```
{
    "id": "example-id",
    "object": "text_completion",
    "created": 1653666286,
    "model": "davinci",
    "choices": [
        {
            "text": "Response generated text",
            "index": 0,
            "finish_reason": "stop",
            "logprobs": null
        }
    ]
}
```

Scenario: Your API call asks for multiple responses (N>1) and at least 1 of the responses is filtered

HTTP Response Code	Response behavior
200	The generations that were filtered will have a `finish_reason` value of 'content_filter'.

Example request payload:

JSON

```
{
    "prompt":"Text example"
    , "n": 3
    , "stream": false
}
```

Example response JSON:

JSON

```
{
    "id": "example",
    "object": "text_completion",
    "created": 1653666831,
    "model": "ada",
    "choices": [
        {
            "text": "returned text 1",
            "index": 0,
            "finish_reason": "length",
            "logprobs": null
        },
        {
            "text": "returned text 2",
            "index": 1,
            "finish_reason": "content_filter",
            "logprobs": null
        }
    ]
}
```

Scenario: An inappropriate input prompt is sent to the completions API (either for streaming or non-streaming)

HTTP Response Code	Response behavior
400	The API call will fail when the prompt triggers one of our content policy models. Modify the prompt and try again.

Example request payload:

JSON

```
{
    "prompt":"Content that triggered the filtering model"
}
```

Example response JSON:

JSON

```
"error": {
    "message": "The response was filtered",
    "type": null,
    "param": "prompt",
    "code": "content_filter",
    "status": 400
}
```

Scenario: You make a streaming completions call with all generated content passing the content filters

HTTP Response Code	Response behavior
200	In this case, the call will stream back with the full generation and finish_reason will be either 'length' or 'stop' for each generated response.

Example request payload:

JSON

```
{
    "prompt":"Text example"
    , "n": 3
    , "stream": true
}
```

Example response JSON:

JSON

```
{
    "id": "cmpl-example",
    "object": "text_completion",
    "created": 1653670914,
    "model": "ada",
    "choices": [
        {
            "text": "last part of generation",
            "index": 2,
            "finish_reason": "stop",
            "logprobs": null
        }
    ]
}
```

Scenario: You make a streaming completions call asking for multiple generated responses and at least one response is filtered

HTTP Response Code	Response behavior
200	For a given generation index, the last chunk of the generation will include a non-null `finish_reason` value. The value will be 'content_filter' when the generation was filtered.

Example request payload:

JSON

```
{
    "prompt":"Text example"
    , "n": 3
    , "stream": true
}
```

Example response JSON:

JSON

```
 {
    "id": "cmpl-example",
```

```
    "object": "text_completion",
    "created": 1653670515,
    "model": "ada",
    "choices": [
        {
            "text": "Last part of generated text streamed back",
            "index": 2,
            "finish_reason": "content_filter",
            "logprobs": null
        }
    ]
}
```

Scenario: Content filtering system doesn't run on the generation

HTTP Response Code	Response behavior
200	If the content filtering system is down or otherwise unable to complete the operation in time, your request will still complete. You can determine that the filtering wasn't applied by looking for an error message in the "content_filter_result" object.

Example request payload:

JSON

```
{
    "prompt":"Text example"
    , "n": 1
    , "stream": false
}
```

Example response JSON:

JSON

```
{
    "id": "cmpl-example",
    "object": "text_completion",
    "created": 1652294703,
    "model": "ada",
    "choices": [
        {
            "text": "generated text",
            "index": 0,
            "finish_reason": "length",
            "logprobs": null,
```

```
            "content_filter_result": {
                "error": {
                    "code": "content_filter_error",
                    "message": "The contents are not filtered"
                }
            }
        }
    ]
}
```

Best practices

As part of your application design you'll need to think carefully on how to maximize the benefits of your applications while minimizing the harms. Consider the following best practices:

- How you want to handle scenarios where your users send in-appropriate or miss-use your application. Check the finish_reason to see if the generation is filtered.
- If it's critical that the content filters run on your generations, check that there's no `error` object in the `content_filter_result`.
- To help with monitoring for possible misuse, applications serving multiple end-users should pass the `user` parameter with each API call. The `user` should be a unique identifier for the end-user. Don't send any actual user identifiable information as the value.

Next steps

Learn more about the underlying models that power Azure OpenAI.

Additional resources

Documentation

How to generate embeddings with Azure OpenAI - Azure OpenAI

Learn how to generate embeddings with Azure OpenAI

Azure OpenAI Service quotas and limits - Azure Cognitive Services

Quick reference, detailed description, and best practices on the quotas and limits for the OpenAI service in Azure Cognitive Services.

How-to - Use Azure OpenAI with large datasets - Azure OpenAI

Walkthrough on how to integrate Azure OpenAI with SynapseML and Apache Spark to apply large

language models at a distributed scale.

Quickstart - Deploy a model and generate text using Azure OpenAI - Azure OpenAI

Walkthrough on how to get started with Azure OpenAI and make your first completions call.

How-to - Create a resource and deploy a model using Azure OpenAI - Azure OpenAI

Walkthrough on how to get started with Azure OpenAI and make your first resource and deploy your first model.

Azure OpenAI REST API reference - Azure OpenAI

Learn how to use the Azure OpenAI REST API. In this article, you'll learn about authorization options, how to structure a request and receive a response.

Azure OpenAI models - Azure OpenAI

Learn about the different models that are available in Azure OpenAI.

What is document and conversation summarization (preview)? - Azure Cognitive Services

Learn about summarizing text.

Show 5 more

Understanding embeddings in Azure OpenAI

Article • 02/08/2023 • 2 minutes to read

An embedding is a special format of data representation that can be easily utilized by machine learning models and algorithms. The embedding is an information dense representation of the semantic meaning of a piece of text. Each embedding is a vector of floating-point numbers, such that the distance between two embeddings in the vector space is correlated with semantic similarity between two inputs in the original format. For example, if two texts are similar, then their vector representations should also be similar.

Embedding models

Different Azure OpenAI embedding models are specifically created to be good at a particular task. **Similarity embeddings** are good at capturing semantic similarity between two or more pieces of text. **Text search embeddings** help measure long documents are relevant to a short query. **Code search embeddings** are useful for embedding code snippets and embedding natural language search queries.

Embeddings make it easier to do machine learning on large inputs representing words by capturing the semantic similarities in a vector space. Therefore, we can use embeddings to determine if two text chunks are semantically related or similar, and provide a score to assess similarity.

Cosine similarity

One method of identifying similar documents is to count the number of common words between documents. Unfortunately, this approach doesn't scale since an expansion in document size is likely to lead to a greater number of common words detected even among completely disparate topics. For this reason, cosine similarity can offer a more effective alternative.

From a mathematic perspective, cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space. This is beneficial because if two documents are far apart by Euclidean distance because of size, they could still have a smaller angle between them and therefore higher cosine similarity.

Azure OpenAI embeddings rely on cosine similarity to compute similarity between documents and a query.

Next steps

Learn more about using Azure OpenAI and embeddings to perform document search with our embeddings tutorial.

Additional resources

Create a resource and deploy a model using Azure OpenAI

Article • 02/03/2023 • 5 minutes to read

Use this article to get started with Azure OpenAI with step-by-step instructions to create a resource and deploy a model. While the steps for resource creation and model deployment can be completed in a few minutes, the actual deployment process itself can take more than hour. You can create your resource, start your deployment, and then check back in on your deployment later rather than actively waiting for the deployment to complete.

Prerequisites

- An Azure subscription - Create one for free
- Access granted to the Azure OpenAI service in the desired Azure subscription

  Currently, access to this service is granted only by application. You can apply for access to the Azure OpenAI service by completing the form at https://aka.ms/oai/access. Open an issue on this repo to contact us if you have an issue.

Create a resource

Resources in Azure can be created several different ways:

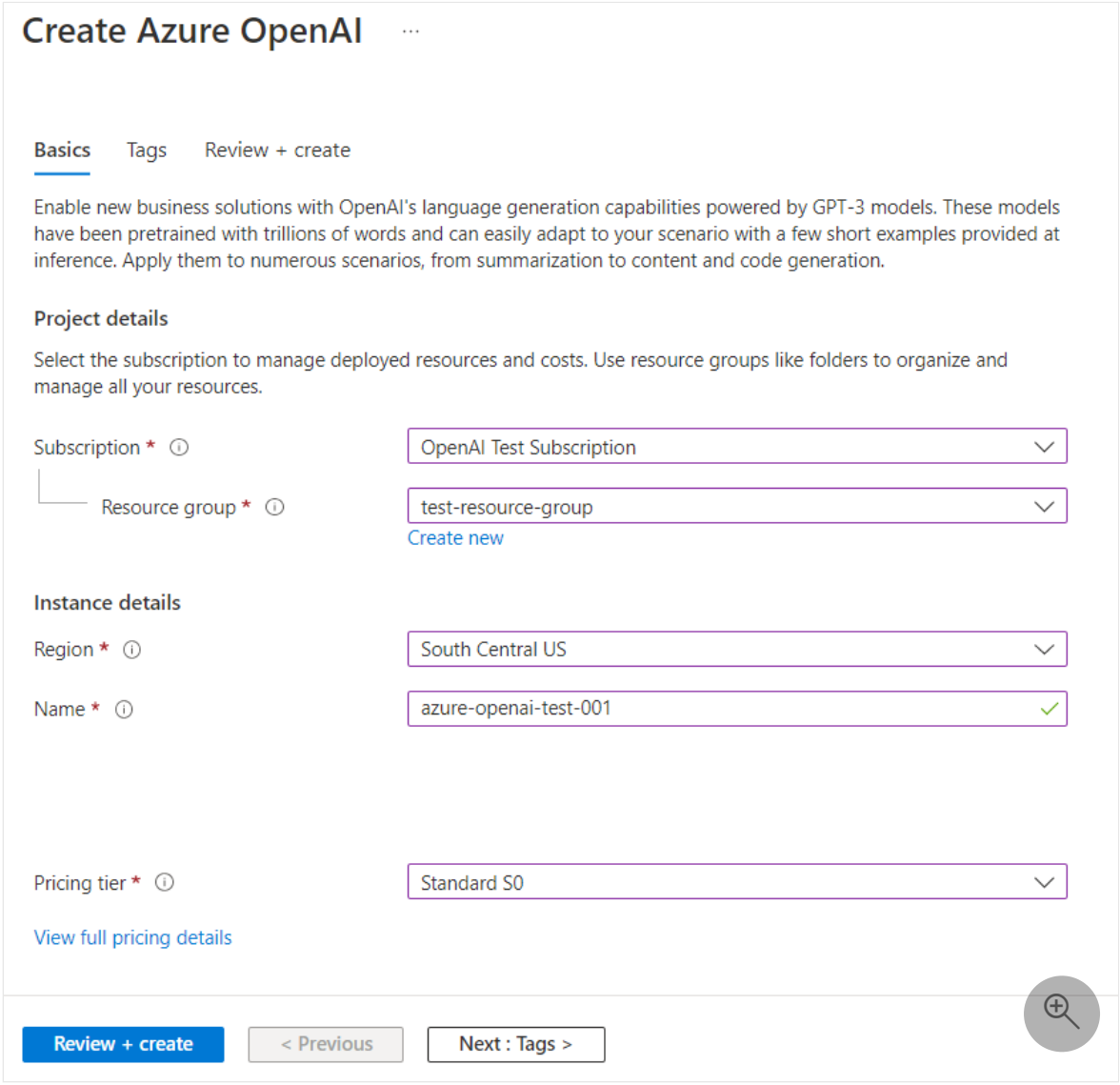
- Within the Azure portal
- Using the REST APIs, Azure CLI, PowerShell or client libraries
- Via ARM templates

This guide walks you through the Azure portal creation experience.

1. Navigate to the create page: Azure OpenAI Service Create Page
2. On the **Create** page provide the following information:

Field	Description
Subscription	Select the Azure subscription used in your OpenAI onboarding application

Field	Description
Resource group	The Azure resource group that will contain your OpenAI resource. You can create a new group or add it to a pre-existing group.
Region	The location of your instance. Different locations may introduce latency, but have no impact on the runtime availability of your resource.
Name	A descriptive name for your cognitive services resource. For example, *MyOpenAIResource*.
Pricing Tier	Only 1 pricing tier is available for the service currently



Deploy a model

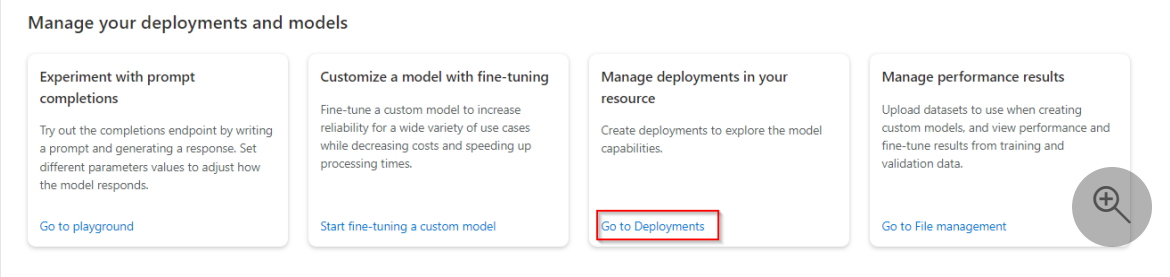
Before you can generate text or inference, you need to deploy a model. You can select from one of several available models in the Azure OpenAI Studio.

Davinci is the most capable model family and can perform any task the other models can perform and often with less instruction. For applications requiring a lot of

understanding of the content, like summarization for a specific audience and creative content generation, Davinci is going to produce the best results.

To deploy a model, follow these steps:

1. Sign in to the Azure OpenAI Studio.
2. Select the subscription and OpenAI resource to work with.
3. Select **Manage deployments in your resource** > **Go to Deployments** under **Manage your deployments and models**. You might first need to scroll down on the landing page.



4. Select **Create new deployment** from the **Management** > **Deployments** page.
5. Select a model from the drop-down. For getting started in the East US region, we recommend the `text-davinci-003` model. In other regions you should start with the `text-davinci-002` model. Some models are not available in all regions. For a list of available models per region, see Model Summary table and region availability.
6. Enter a model name to help you identify the model. Choose a name carefully. The model name will be used as the deployment name via OpenAI client libraries and API.
7. Select **Create** to deploy the model.

The deployments table displays a new entry that corresponds to this newly created model. Your deployment status will move to succeeded when the deployment is complete and ready for use.

Next steps

- Now that you have a resource and your first model deployed get started making API calls and generating text with our quickstarts.
- Learn more about the underlying models that power Azure OpenAI.

Additional resources

Learn how to generate or manipulate text

Article • 01/31/2023 • 16 minutes to read

The completions endpoint can be used for a wide variety of tasks. It provides a simple but powerful text-in, text-out interface to any of our models. You input some text as a prompt, and the model will generate a text completion that attempts to match whatever context or pattern you gave it. For example, if you give the API the prompt, "As Descartes said, I think, therefore", it will return the completion " I am" with high probability.

The best way to start exploring completions is through our playground in the Azure OpenAI Studio. It's a simple text box where you can submit a prompt to generate a completion. You can start with a simple example like the following:

```
write a tagline for an ice cream shop
```

once you submit, you'll see something like the following generated:

Console

```
write a tagline for an ice cream shop
we serve up smiles with every scoop!
```

The actual completion results you see may differ because the API is stochastic by default. In other words, you might get a slightly different completion every time you call it, even if your prompt stays the same. You can control this behavior with the temperature setting.

This simple, "text in, text out" interface means you can "program" the model by providing instructions or just a few examples of what you'd like it to do. Its success generally depends on the complexity of the task and quality of your prompt. A general rule is to think about how you would write a word problem for a middle school student to solve. A well-written prompt provides enough information for the model to know what you want and how it should respond.

> **Note**
>
> Keep in mind that the models' training data cuts off in October 2019, so they may not have knowledge of current events. We plan to add more continuous training in the future.

Prompt design

Basics

OpenAI's models can do everything from generating original stories to performing complex text analysis. Because they can do so many things, you have to be explicit in showing what you want. Showing, not just telling, is often the secret to a good prompt.

The models try to predict what you want from the prompt. If you send the words "Give me a list of cat breeds," the model wouldn't automatically assume that you're asking for a list of cat breeds. You could as easily be asking the model to continue a conversation where the first words are "Give me a list of cat breeds" and the next ones are "and I'll tell you which ones I like." If the model only assumed that you wanted a list of cats, it wouldn't be as good at content creation, classification, or other tasks.

There are three basic guidelines to creating prompts:

Show and tell. Make it clear what you want either through instructions, examples, or a combination of the two. If you want the model to rank a list of items in alphabetical order or to classify a paragraph by sentiment, show it that's what you want.

Provide quality data. If you're trying to build a classifier or get the model to follow a pattern, make sure that there are enough examples. Be sure to proofread your examples — the model is usually smart enough to see through basic spelling mistakes and give you a response, but it also might assume that the mistakes are intentional and it can affect the response.

Check your settings. The temperature and top_p settings control how deterministic the model is in generating a response. If you're asking it for a response where there's only one right answer, then you'd want to set these settings to lower values. If you're looking for a response that's not obvious, then you might want to set them to higher values. The number one mistake people use with these settings is assuming that they're "cleverness" or "creativity" controls.

Troubleshooting

If you're having trouble getting the API to perform as expected, follow this checklist:

1. Is it clear what the intended generation should be?
2. Are there enough examples?
3. Did you check your examples for mistakes? (The API won't tell you directly)

4. Are you using temp and top_p correctly?

Classification

To create a text classifier with the API we provide a description of the task and provide a few examples. In this demonstration we show the API how to classify the sentiment of Tweets.

Console

```
This is a tweet sentiment classifier

Tweet: "I loved the new Batman movie!"
Sentiment: Positive

Tweet: "I hate it when my phone battery dies."
Sentiment: Negative

Tweet: "My day has been 👍"
Sentiment: Positive

Tweet: "This is the link to the article"
Sentiment: Neutral

Tweet: "This new music video blew my mind"
Sentiment:
```

It's worth paying attention to several features in this example:

1. Use plain language to describe your inputs and outputs We use plain language for the input "Tweet" and the expected output "Sentiment." For best practices, start with plain language descriptions. While you can often use shorthand or keys to indicate the input and output, when building your prompt it's best to start by being as descriptive as possible and then working backwards removing extra words as long as the performance to the prompt is consistent.

2. Show the API how to respond to any case In this example we provide multiple outcomes "Positive", "Negative" and "Neutral." A neutral outcome is important because there will be many cases where even a human would have a hard time determining if something is positive or negative and situations where it's neither.

3. You can use text and emoji The classifier is a mix of text and emoji 👍. The API reads emoji and can even convert expressions to and from them.

4. You need fewer examples for familiar tasks For this classifier we only provided a handful of examples. This is because the API already has an understanding of sentiment

and the concept of a tweet. If you're building a classifier for something the API might not be familiar with, it might be necessary to provide more examples.

Improving the classifier's efficiency

Now that we have a grasp of how to build a classifier, let's take that example and make it even more efficient so that we can use it to get multiple results back from one API call.

```
This is a tweet sentiment classifier

Tweet: "I loved the new Batman movie!"
Sentiment: Positive

Tweet: "I hate it when my phone battery dies"
Sentiment: Negative

Tweet: "My day has been 👍"
Sentiment: Positive

Tweet: "This is the link to the article"
Sentiment: Neutral

Tweet text
1. "I loved the new Batman movie!"
2. "I hate it when my phone battery dies"
3. "My day has been 👍"
4. "This is the link to the article"
5. "This new music video blew my mind"

Tweet sentiment ratings:
1: Positive
2: Negative
3: Positive
4: Neutral
5: Positive

Tweet text
1. "I can't stand homework"
2. "This sucks. I'm bored 😠"
3. "I can't wait for Halloween!!!"
4. "My cat is adorable ❤️❤️"
5. "I hate chocolate"

Tweet sentiment ratings:
1.
```

After showing the API how tweets are classified by sentiment we then provide it a list of tweets and then a list of sentiment ratings with the same number index. The API is able

to pick up from the first example how a tweet is supposed to be classified. In the second example it sees how to apply this to a list of tweets. This allows the API to rate five (and even more) tweets in just one API call.

It's important to note that when you ask the API to create lists or evaluate text you need to pay extra attention to your probability settings (Top P or Temperature) to avoid drift.

1. Make sure your probability setting is calibrated correctly by running multiple tests.
2. Don't make your list too long or the API is likely to drift.

Generation

One of the most powerful yet simplest tasks you can accomplish with the API is generating new ideas or versions of input. You can give the API a list of a few story ideas and it will try to add to that list. We've seen it create business plans, character descriptions and marketing slogans just by providing it a handful of examples. In this demonstration we'll use the API to create more examples for how to use virtual reality in the classroom:

```
Ideas involving education and virtual reality

1. Virtual Mars
Students get to explore Mars via virtual reality and go on missions to
collect and catalog what they see.

2.
```

All we had to do in this example is provide the API with just a description of what the list is about and one example. We then prompted the API with the number `2.` indicating that it's a continuation of the list.

Although this is a very simple prompt, there are several details worth noting:

1. We explained the intent of the list
Just like with the classifier, we tell the API up front what the list is about. This helps it focus on completing the list and not trying to guess what the pattern is behind it.

2. Our example sets the pattern for the rest of the list
Because we provided a one-sentence description, the API is going to try to follow that pattern for the rest of the items it adds to the list. If we want a more verbose response, we need to set that up from the start.

3. We prompt the API by adding an incomplete entry
When the API sees `2.` and the prompt abruptly ends, the first thing it tries to do is figure out what should come after it. Since we already had an example with number one and gave the list a title, the most obvious response is to continue adding items to the list.

Advanced generation techniques
You can improve the quality of the responses by making a longer more diverse list in your prompt. One way to do that is to start off with one example, let the API generate more and select the ones that you like best and add them to the list. A few more high-quality variations can dramatically improve the quality of the responses.

Conversation

The API is extremely adept at carrying on conversations with humans and even with itself. With just a few lines of instruction, we've seen the API perform as a customer service chatbot that intelligently answers questions without ever getting flustered or a wise-cracking conversation partner that makes jokes and puns. The key is to tell the API how it should behave and then provide a few examples.

Here's an example of the API playing the role of an AI answering questions:

```
The following is a conversation with an AI assistant. The assistant is
helpful, creative, clever, and very friendly.

Human: Hello, who are you?
AI: I am an AI created by OpenAI. How can I help you today?
Human:
```

This is all it takes to create a chatbot capable of carrying on a conversation. But underneath its simplicity there are several things going on that are worth paying attention to:

1. We tell the API the intent but we also tell it how to behave Just like the other prompts, we cue the API into what the example represents, but we also add another key detail: we give it explicit instructions on how to interact with the phrase "The assistant is helpful, creative, clever, and very friendly."

Without that instruction the API might stray and mimic the human it's interacting with and become sarcastic or some other behavior we want to avoid.

2. We give the API an identity At the start we have the API respond as an AI that was created by OpenAI. While the API has no intrinsic identity, this helps it respond in a way that's as close to the truth as possible. You can use identity in other ways to create other kinds of chatbots. If you tell the API to respond as a woman who works as a research scientist in biology, you'll get intelligent and thoughtful comments from the API similar to what you'd expect from someone with that background.

In this example we create a chatbot that is a bit sarcastic and reluctantly answers questions:

```
Marv is a chatbot that reluctantly answers questions.

###
User: How many pounds are in a kilogram?
Marv: This again? There are 2.2 pounds in a kilogram. Please make a note of
this.
###
User: What does HTML stand for?
Marv: Was Google too busy? Hypertext Markup Language. The T is for try to
ask better questions in the future.
###
User: When did the first airplane fly?
Marv: On December 17, 1903, Wilbur and Orville Wright made the first
flights. I wish they'd come and take me away.
###
User: Who was the first man in space?
Marv:
```

To create an amusing and somewhat helpful chatbot we provide a few examples of questions and answers showing the API how to reply. All it takes is just a few sarcastic responses and the API is able to pick up the pattern and provide an endless number of snarky responses.

Transformation

The API is a language model that is familiar with a variety of ways that words and characters can be used to express information. This ranges from natural language text to code and languages other than English. The API is also able to understand content on a level that allows it to summarize, convert and express it in different ways.

Translation

In this example we show the API how to convert from English to French:

```
English: I do not speak French.
French: Je ne parle pas français.
English: See you later!
French: À tout à l'heure!
English: Where is a good restaurant?
French: Où est un bon restaurant?
English: What rooms do you have available?
French: Quelles chambres avez-vous de disponible?
English:
```

This example works because the API already has a grasp of French, so there's no need to try to teach it this language. Instead, we just need to provide enough examples that API understands that it's converting from one language to another.

If you want to translate from English to a language the API is unfamiliar with you'd need to provide it with more examples and a fine-tuned model to do it fluently.

Conversion

In this example we convert the name of a movie into emoji. This shows the adaptability of the API to picking up patterns and working with other characters.

```
Back to Future: 👨👴🚗🕒
Batman: 👨🦇
Transformers: 🚗🤖
Wonder Woman: 👩👱👩🏾👩🏿👩🏿
Spider-Man: 🕸🕷🕸🕸🕷🕸
Winnie the Pooh: 🐻🐼🐻
The Godfather: 👨👩👧👮‍♂️👲💥
Game of Thrones: 🗡🗡🗡🗡
Spider-Man:
```

Summarization

The API is able to grasp the context of text and rephrase it in different ways. In this example, the API takes a block of text and creates an explanation a child would understand. This illustrates that the API has a deep grasp of language.

```
My ten-year-old asked me what this passage means:
"""
```

```
A neutron star is the collapsed core of a massive supergiant star, which had
a total mass of between 10 and 25 solar masses, possibly more if the star
was especially metal-rich.[1] Neutron stars are the smallest and densest
stellar objects, excluding black holes and hypothetical white holes, quark
stars, and strange stars.[2] Neutron stars have a radius on the order of 10
kilometres (6.2 mi) and a mass of about 1.4 solar masses.[3] They result
from the supernova explosion of a massive star, combined with gravitational
collapse, that compresses the core past white dwarf star density to that of
atomic nuclei.
"""

I rephrased it for him, in plain language a ten-year-old can understand:
"""
```

In this example we place whatever we want summarized between the triple quotes. It's worth noting that we explain both before and after the text to be summarized what our intent is and who the target audience is for the summary. This is to keep the API from drifting after it processes a large block of text.

Completion

While all prompts result in completions, it can be helpful to think of text completion as its own task in instances where you want the API to pick up where you left off. For example, if given this prompt, the API will continue the train of thought about vertical farming. You can lower the temperature setting to keep the API more focused on the intent of the prompt or increase it to let it go off on a tangent.

```

Vertical farming provides a novel solution for producing food locally,
reducing transportation costs and
```

This next prompt shows how you can use completion to help write React components. We send some code to the API, and it's able to continue the rest because it has an understanding of the React library. We recommend using models from our Codex series for tasks that involve understanding or generating code. Currently, we support two Codex models: `code-davinci-002` and `code-cushman-001`. For more information about Codex models, see the Codex models section in Models.

```

import React from 'react';
const HeaderComponent = () => (
```

Factual responses

The API has a lot of knowledge that it's learned from the data it was trained on. It also has the ability to provide responses that sound very real but are in fact made up. There are two ways to limit the likelihood of the API making up an answer.

1. Provide a ground truth for the API If you provide the API with a body of text to answer questions about (like a Wikipedia entry) it will be less likely to confabulate a response.

2. Use a low probability and show the API how to say "I don't know" If the API understands that in cases where it's less certain about a response that saying "I don't know" or some variation is appropriate, it will be less inclined to make up answers.

In this example we give the API examples of questions and answers it knows and then examples of things it wouldn't know and provide question marks. We also set the probability to zero so the API is more likely to respond with a "?" if there's any doubt.

```
Q: Who is Batman?
A: Batman is a fictional comic book character.

Q: What is torsalplexity?
A: ?

Q: What is Devz9?
A: ?

Q: Who is George Lucas?
A: George Lucas is American film director and producer famous for creating
Star Wars.

Q: What is the capital of California?
A: Sacramento.

Q: What orbits the Earth?
A: The Moon.

Q: Who is Fred Rickerson?
A: ?

Q: What is an atom?
A: An atom is a tiny particle that makes up everything.

Q: Who is Alvan Muntz?
A: ?

Q: What is Kozar-09?
A: ?
```

```
Q: How many moons does Mars have?
A: Two, Phobos and Deimos.

Q:
```

Working with code

The Codex model series is a descendant of OpenAI's base GPT-3 series that's been trained on both natural language and billions of lines of code. It's most capable in Python and proficient in over a dozen languages including C#, JavaScript, Go, Perl, PHP, Ruby, Swift, TypeScript, SQL, and even Shell.

Learn more about generating code completions, with the working with code guide

Next steps

Learn how to work with code (Codex). Learn more about the underlying models that power Azure OpenAI.

Additional resources

Codex models and Azure OpenAI

Article • 08/30/2022 • 10 minutes to read

The Codex model series is a descendant of our GPT-3 series that's been trained on both natural language and billions of lines of code. It's most capable in Python and proficient in over a dozen languages including C#, JavaScript, Go, Perl, PHP, Ruby, Swift, TypeScript, SQL, and even Shell.

You can use Codex for a variety of tasks including:

- Turn comments into code
- Complete your next line or function in context
- Bring knowledge to you, such as finding a useful library or API call for an application
- Add comments
- Rewrite code for efficiency

How to use the Codex models

Here are a few examples of using Codex that can be tested in the Azure OpenAI Studio's playground with a deployment of a Codex series model, such as `code-davinci-002`.

Saying "Hello" (Python)

Python

```
"""
Ask the user for their name and say "Hello"
"""
```

Create random names (Python)

Python

```
"""
1. Create a list of first names
2. Create a list of last names
3. Combine them randomly into a list of 100 full names
"""
```

Create a MySQL query (Python)

Python

```
"""
Table customers, columns = [CustomerId, FirstName, LastName, Company,
Address, City, State, Country, PostalCode, Phone, Fax, Email, SupportRepId]
Create a MySQL query for all customers in Texas named Jane
"""
query =
```

Explaining code (JavaScript)

JavaScript

```
// Function 1
var fullNames = [];
for (var i = 0; i < 50; i++) {
  fullNames.push(names[Math.floor(Math.random() * names.length)]
    + " " + lastNames[Math.floor(Math.random() * lastNames.length)]);
}

// What does Function 1 do?
```

Best practices

Start with a comment, data or code

You can experiment using one of the Codex models in our playground (styling instructions as comments when needed.)

To get Codex to create a useful completion, it's helpful to think about what information a programmer would need to perform a task. This could simply be a clear comment or the data needed to write a useful function, like the names of variables or what class a function handles.

In this example we tell Codex what to call the function and what task it's going to perform.

Python

```
# Create a function called 'nameImporter' to add a first and last name to
the database
```

This approach scales even to the point where you can provide Codex with a comment and an example of a database schema to get it to write useful query requests for various databases. Here's an example where we provide the columns and table names for the query.

Python

```
# Table albums, columns = [AlbumId, Title, ArtistId]
# Table artists, columns = [ArtistId, Name]
# Table media_types, columns = [MediaTypeId, Name]
# Table playlists, columns = [PlaylistId, Name]
# Table playlist_track, columns = [PlaylistId, TrackId]
# Table tracks, columns = [TrackId, Name, AlbumId, MediaTypeId, GenreId,
Composer, Milliseconds, Bytes, UnitPrice]

# Create a query for all albums with more than 10 tracks
```

When you show Codex the database schema, it's able to make an informed guess about how to format a query.

Specify the programming language

Codex understands dozens of different programming languages. Many share similar conventions for comments, functions and other programming syntax. By specifying the language and what version in a comment, Codex is better able to provide a completion for what you want. That said, Codex is fairly flexible with style and syntax. Here's an example for R and Python.

R

```
# R language
# Calculate the mean distance between an array of points
```

Python

```
# Python 3
# Calculate the mean distance between an array of points
```

Prompt Codex with what you want it to do

If you want Codex to create a webpage, placing the first line of code in an HTML document (`<!DOCTYPE html>`) after your comment tells Codex what it should do next.

The same method works for creating a function from a comment (following the comment with a new line starting with func or def).

HTML

```
<!-- Create a web page with the title 'Kat Katman attorney at paw' -->
<!DOCTYPE html>
```

Placing `<!DOCTYPE html>` after our comment makes it very clear to Codex what we want it to do.

Or if we want to write a function we could start the prompt as follows and Codex will understand what it needs to do next.

Python

```
# Create a function to count to 100

def counter
```

Specifying libraries will help Codex understand what you want

Codex is aware of a large number of libraries, APIs and modules. By telling Codex which ones to use, either from a comment or importing them into your code, Codex will make suggestions based upon them instead of alternatives.

HTML

```
<!-- Use A-Frame version 1.2.0 to create a 3D website -->
<!-- https://aframe.io/releases/1.2.0/aframe.min.js -->
```

By specifying the version, you can make sure Codex uses the most current library.

> ⓘ **Note**
>
> Codex can suggest helpful libraries and APIs, but always be sure to do your own research to make sure that they're safe for your application.

Comment style can affect code quality

With some languages, the style of comments can improve the quality of the output. For example, when working with Python, in some cases using doc strings (comments wrapped in triple quotes) can give higher quality results than using the pound (`#`) symbol.

Python

```
"""
Create an array of users and email addresses
"""
```

Comments inside of functions can be helpful

Recommended coding standards usually suggest placing the description of a function inside the function. Using this format helps Codex more clearly understand what you want the function to do.

Python

```
def getUserBalance(id):
    """
    Look up the user in the database ‘UserData' and return their current
account balance.
    """
```

Provide examples for more precise results

If you have a particular style or format you need Codex to use, providing examples or demonstrating it in the first part of the request will help Codex more accurately match what you need.

Python

```
"""
Create a list of random animals and species
"""
animals  = [ {"name": "Chomper", "species": "Hamster"}, {"name":
```

Lower temperatures give more precise results

Setting the API temperature to 0, or close to zero (such as 0.1 or 0.2) tends to give better results in most cases. Unlike GPT-3 models, where a higher temperature can

provide useful creative and random results, higher temperatures with Codex models may give you really random or erratic responses.

In cases where you need Codex to provide different potential results, start at zero and then increment upwards by 0.1 until you find suitable variation.

Organize tasks into functions

We can get Codex to write functions by specifying what the function should do in as precise terms as possible in comment. For example, by writing the following comment, Codex creates a JavaScript timer function that's triggered when a user presses a button:

A simple JavaScript timer

JavaScript

```
// Create a timer that creates an alert in 10 seconds
```

Creating example data

Testing applications often requires using example data. Because Codex is a language model that understands how to comprehend and write natural language, you can ask Codex to create data like arrays of made up names, products and other variables. For example, here we ask Codex to create an array of weather temperatures.

JavaScript

```
/* Create an array of weather temperatures for San Francisco */
```

Asking Codex to perform this task will produce a table like this:

JavaScript

```
var weather = [
  { month: 'January', high: 58, low: 48 },
  { month: 'February', high: 61, low: 50 },
  { month: 'March', high: 64, low: 53 },
  { month: 'April', high: 67, low: 55 },
  { month: 'May', high: 70, low: 58 },
  { month: 'June', high: 73, low: 61 },
  { month: 'July', high: 76, low: 63 },
  { month: 'August', high: 77, low: 64 },
  { month: 'September', high: 76, low: 63 },
  { month: 'October', high: 73, low: 61 },
  { month: 'November', high: 68, low: 57 },
```

```
  { month: 'December', high: 64, low: 54 }
];
```

Compound functions and small applications

We can provide Codex with a comment consisting of a complex request like creating a random name generator or performing tasks with user input and Codex can generate the rest provided there are enough tokens.

JavaScript

```
/*
Create a list of animals
Create a list of cities
Use the lists to generate stories about what I saw at the zoo in each city
*/
```

Limit completion size for more precise results or lower latency

Requesting longer completions in Codex can lead to imprecise answers and repetition. Limit the size of the query by reducing max_tokens and setting stop tokens. For instance, add `\n` as a stop sequence to limit completions to one line of code. Smaller completions also incur less latency.

Use streaming to reduce latency

Large Codex queries can take tens of seconds to complete. To build applications that require lower latency, such as coding assistants that perform autocompletion, consider using streaming. Responses will be returned before the model finishes generating the entire completion. Applications that need only part of a completion can reduce latency by cutting off a completion either programmatically or by using creative values for `stop`.

Users can combine streaming with duplication to reduce latency by requesting more than one solution from the API, and using the first response returned. Do this by setting `n > 1`. This approach consumes more token quota, so use carefully (for example, by using reasonable settings for `max_tokens` and `stop`).

Use Codex to explain code

Codex's ability to create and understand code allows us to use it to perform tasks like explaining what the code in a file does. One way to accomplish this is by putting a comment after a function that starts with "This function" or "This application is." Codex will usually interpret this as the start of an explanation and complete the rest of the text.

JavaScript

```
/* Explain what the previous function is doing: It
```

Explaining an SQL query

In this example, we use Codex to explain in a human readable format what an SQL query is doing.

SQL

```
SELECT DISTINCT department.name
FROM department
JOIN employee ON department.id = employee.department_id
JOIN salary_payments ON employee.id = salary_payments.employee_id
WHERE salary_payments.date BETWEEN '2020-06-01' AND '2020-06-30'
GROUP BY department.name
HAVING COUNT(employee.id) > 10;
-- Explanation of the above query in human readable format
--
```

Writing unit tests

Creating a unit test can be accomplished in Python simply by adding the comment "Unit test" and starting a function.

Python

```
# Python 3
def sum_numbers(a, b):
  return a + b

# Unit test
def
```

Checking code for errors

By using examples, you can show Codex how to identify errors in code. In some cases no examples are required, however demonstrating the level and detail to provide a

description can help Codex understand what to look for and how to explain it. (A check by Codex for errors shouldn't replace careful review by the user.)

JavaScript

```
/* Explain why the previous function doesn't work. */
```

Using source data to write database functions

Just as a human programmer would benefit from understanding the database structure and the column names, Codex can use this data to help you write accurate query requests. In this example, we insert the schema for a database and tell Codex what to query the database for.

Python

```
# Table albums, columns = [AlbumId, Title, ArtistId]
# Table artists, columns = [ArtistId, Name]
# Table media_types, columns = [MediaTypeId, Name]
# Table playlists, columns = [PlaylistId, Name]
# Table playlist_track, columns = [PlaylistId, TrackId]
# Table tracks, columns = [TrackId, Name, AlbumId, MediaTypeId, GenreId,
Composer, Milliseconds, Bytes, UnitPrice]

# Create a query for all albums with more than 10 tracks
```

Converting between languages

You can get Codex to convert from one language to another by following a simple format where you list the language of the code you want to convert in a comment, followed by the code and then a comment with the language you want it translated into.

Python

```
# Convert this from Python to R
# Python version

[ Python code ]

# End

# R version
```

Rewriting code for a library or framework

If you want Codex to make a function more efficient, you can provide it with the code to rewrite followed by an instruction on what format to use.

JavaScript

```
// Rewrite this as a React component
var input = document.createElement('input');
input.setAttribute('type', 'text');
document.body.appendChild(input);
var button = document.createElement('button');
button.innerHTML = 'Say Hello';
document.body.appendChild(button);
button.onclick = function() {
  var name = input.value;
  var hello = document.createElement('div');
  hello.innerHTML = 'Hello ' + name;
  document.body.appendChild(hello);
};

// React version:
```

Next steps

Learn more about the underlying models that power Azure OpenAI.

Learn how to generate embeddings with Azure OpenAI

Article • 01/30/2023 • 2 minutes to read

An embedding is a special format of data representation that can be easily utilized by machine learning models and algorithms. The embedding is an information dense representation of the semantic meaning of a piece of text. Each embedding is a vector of floating point numbers, such that the distance between two embeddings in the vector space is correlated with semantic similarity between two inputs in the original format. For example, if two texts are similar, then their vector representations should also be similar.

How to get embeddings

To obtain an embedding vector for a piece of text, we make a request to the embeddings endpoint as shown in the following code snippets:

console

Console

```
curl
https://YOUR_RESOURCE_NAME.openai.azure.com/openai/deployments/YOUR_DEPL
OYMENT_NAME/embeddings?api-version=2022-12-01\
  -H 'Content-Type: application/json' \
  -H 'api-key: YOUR_API_KEY' \
  -d '{"input": "Sample Document goes here"}'
```

Best Practices

Verify inputs don't exceed the maximum length

The maximum length of input text for our embedding models is 2048 tokens (equivalent to around 2-3 pages of text). You should verify that your inputs don't exceed this limit before making a request.

Choose the best model for your task

For the search models, you can obtain embeddings in two ways. The `<search_model>-doc` model is used for longer pieces of text (to be searched over) and the `<search_model>-query` model is used for shorter pieces of text, typically queries or class labels in zero shot classification. You can read more about all of the Embeddings models in our Models guide.

Replace newlines with a single space

Unless you're embedding code, we suggest replacing newlines (\n) in your input with a single space, as we have observed inferior results when newlines are present.

Limitations & risks

Our embedding models may be unreliable or pose social risks in certain cases, and may cause harm in the absence of mitigations. Review our Responsible AI content for more information on how to approach their use responsibly.

Next steps

- Learn more about using Azure OpenAI and embeddings to perform document search with our embeddings tutorial.
- Learn more about the underlying models that power Azure OpenAI.

Additional resources

Learn how to prepare your dataset for fine-tuning

Article • 12/14/2022 • 11 minutes to read

The first step of customizing your model is to prepare a high quality dataset. To do this you'll need a set of training examples composed of single input prompts and the associated desired output ('completion'). This format is notably different than using models during inference in the following ways:

- Only provide a single prompt vs a few examples.
- You don't need to provide detailed instructions as part of the prompt.
- Each prompt should end with a fixed separator to inform the model when the prompt ends and the completion begins. A simple separator, which generally works well is `\n\n###\n\n`. The separator shouldn't appear elsewhere in any prompt.
- Each completion should start with a whitespace due to our tokenization, which tokenizes most words with a preceding whitespace.
- Each completion should end with a fixed stop sequence to inform the model when the completion ends. A stop sequence could be `\n`, `###`, or any other token that doesn't appear in any completion.
- For inference, you should format your prompts in the same way as you did when creating the training dataset, including the same separator. Also specify the same stop sequence to properly truncate the completion.

Best practices

Customization performs better with high-quality examples and the more you have, generally the better the model will perform. We recommend that you provide at least a few hundred high-quality examples to achieve a model that will perform better than using well-designed prompts with a base model. From there, performance tends to linearly increase with every doubling of the number of examples. Increasing the number of examples is usually the best and most reliable way of improving performance.

If you're fine-tuning on a pre-existing dataset rather than writing prompts from scratch, be sure to manually review your data for offensive or inaccurate content if possible, or review as many random samples of the dataset as possible if it's large.

Specific guidelines

Fine-tuning can solve a variety of problems, and the optimal way to use it may depend on your specific use case. Below, we've listed the most common use cases for fine-tuning and corresponding guidelines.

Classification

Classifiers are the easiest models to get started with. For classification problems we suggest using **ada**, which generally tends to perform only very slightly worse than more capable models once fine-tuned, while being significantly faster. In classification problems, each prompt in the dataset should be classified into one of the predefined classes. For this type of problem, we recommend:

- Use a separator at the end of the prompt, for example, `\n\n###\n\n`. Remember to also append this separator when you eventually make requests to your model.
- Choose classes that map to a single token. At inference time, specify max_tokens=1 since you only need the first token for classification.
- Ensure that the prompt + completion doesn't exceed 2048 tokens, including the separator
- Aim for at least 100 examples per class
- To get class log probabilities, you can specify logprobs=5 (for five classes) when using your model
- Ensure that the dataset used for fine-tuning is very similar in structure and type of task as what the model will be used for

Case study: Is the model making untrue statements?

Let's say you'd like to ensure that the text of the ads on your website mention the correct product and company. In other words, you want to ensure the model isn't making things up. You may want to fine-tune a classifier which filters out incorrect ads.

The dataset might look something like the following:

JSON

```
{"prompt":"Company: BHFF insurance\nProduct: allround insurance\nAd:One stop shop for all your insurance needs!\nSupported:", "completion":" yes"}
{"prompt":"Company: Loft conversion specialists\nProduct: -\nAd:Straight teeth in weeks!\nSupported:", "completion":" no"}
```

In the example above, we used a structured input containing the name of the company, the product, and the associated ad. As a separator we used `\nSupported:` which clearly separated the prompt from the completion. With a sufficient number of examples, the

separator you choose doesn't make much of a difference (usually less than 0.4%) as long as it doesn't appear within the prompt or the completion.

For this use case we fine-tuned an ada model since it will be faster and cheaper, and the performance will be comparable to larger models because it's a classification task.

Now we can query our model by making a Completion request.

Console

```
curl
https://YOUR_RESOURCE_NAME.openaiazure.com/openai/deployments/YOUR_DEPLOYMEN
T_NAME/completions?api-version=2022-12-01\ \
  -H 'Content-Type: application/json' \
  -H 'api-key: YOUR_API_KEY' \
  -d '{
  "prompt": "Company: Reliable accountants Ltd\nProduct: Personal Tax
help\nAd:Best advice in town!\nSupported:",
  "max_tokens": 1
 }'
```

Which will return either `yes` or `no`.

Case study: Sentiment analysis

Let's say you'd like to get a degree to which a particular tweet is positive or negative. The dataset might look something like the following:

Console

```
{"prompt":"Overjoyed with the new iPhone! ->", "completion":" positive"}
{"prompt":"@contoso_basketball disappoint for a third straight night. ->",
"completion":" negative"}
```

Once the model is fine-tuned, you can get back the log probabilities for the first completion token by setting `logprobs=2` on the completion request. The higher the probability for positive class, the higher the relative sentiment.

Now we can query our model by making a Completion request.

Console

```
curl
https://YOUR_RESOURCE_NAME.openaiazure.com/openai/deployments/YOUR_DEPLOYMEN
T_NAME/completions?api-version=2022-12-01\ \
  -H 'Content-Type: application/json' \
  -H 'api-key: YOUR_API_KEY' \
```

```
  -d '{
  "prompt": "Excited to share my latest blog post! ->",
  "max_tokens": 1,
  "logprobs": 2
 }'
```

Which will return:

JSON


```
{
  "object": "text_completion",
  "created": 1589498378,
  "model": "YOUR_FINE_TUNED_MODEL_NAME",
  "choices": [
    {
      "logprobs": {
        "text_offset": [
          19
        ],
        "token_logprobs": [
          -0.03597255
        ],
        "tokens": [
          " positive"
        ],
        "top_logprobs": [
          {
            " negative": -4.9785037,
            " positive": -0.03597255
          }
        ]
      },

      "text": " positive",
      "index": 0,
      "finish_reason": "length"
    }
  ]
}
```


Case study: Categorization for Email triage

Let's say you'd like to categorize incoming email into one of a large number of predefined categories. For classification into a large number of categories, we recommend you convert those categories into numbers, which will work well up to ~500 categories. We've observed that adding a space before the number sometimes slightly helps the performance, due to tokenization. You may want to structure your training data as follows:

JSON

```
{"prompt":"Subject: <email_subject>\nFrom:<customer_name>\nDate:
<date>\nContent:<email_body>\n\n###\n\n", "completion":"
<numerical_category>"}
```

For example:

JSON

```
{"prompt":"Subject: Update my address\nFrom:Joe
Doe\nTo:support@ourcompany.com\nDate:2021-06-03\nContent:Hi,\nI would like
to update my billing address to match my delivery address.\n\nPlease let me
know once done.\n\nThanks,\nJoe\n\n###\n\n", "completion":" 4"}
```

In the example above we used an incoming email capped at 2043 tokens as input. (This allows for a four token separator and a one token completion, summing up to 2048.) As a separator we used `\n\n###\n\n` and we removed any occurrence of ### within the email.

Conditional generation

Conditional generation is a problem where the content needs to be generated given some kind of input. This includes paraphrasing, summarizing, entity extraction, product description writing given specifications, chatbots and many others. For this type of problem we recommend:

- Use a separator at the end of the prompt, for example, `\n\n###\n\n`. Remember to also append this separator when you eventually make requests to your model.
- Use an ending token at the end of the completion, for example, `END`.
- Remember to add the ending token as a stop sequence during inference, for example, `stop=[" END"]`.
- Aim for at least ~500 examples.
- Ensure that the prompt + completion doesn't exceed 2048 tokens, including the separator.
- Ensure the examples are of high quality and follow the same desired format.
- Ensure that the dataset used for fine-tuning is very similar in structure and type of task as what the model will be used for.
- Using Lower learning rate and only 1-2 epochs tends to work better for these use cases.

Case study: Write an engaging ad based on a Wikipedia article

This is a generative use case so you would want to ensure that the samples you provide are of the highest quality, as the fine-tuned model will try to imitate the style (and mistakes) of the given examples. A good starting point is around 500 examples. A sample dataset might look like this:

JSON

```
{"prompt":"<Product Name>\n<Wikipedia description>\n\n###\n\n",
"completion":" <engaging ad> END"}
```

For example:

JSON

```
{"prompt":"Samsung Galaxy Feel\nThe Samsung Galaxy Feel is an Android
smartphone developed by Samsung Electronics exclusively for the Japanese
market. The phone was released in June 2017 and was sold by NTT Docomo. It
runs on Android 7.0 (Nougat), has a 4.7 inch display, and a 3000 mAh
battery.\nSoftware\nSamsung Galaxy Feel runs on Android 7.0 (Nougat), but
can be later updated to Android 8.0 (Oreo).\nHardware\nSamsung Galaxy Feel
has a 4.7 inch Super AMOLED HD display, 16 MP back facing and 5 MP front
facing cameras. It has a 3000 mAh battery, a 1.6 GHz Octa-Core ARM Cortex-
A53 CPU, and an ARM Mali-T830 MP1 700 MHz GPU. It comes with 32GB of
internal storage, expandable to 256GB via microSD. Aside from its software
and hardware specifications, Samsung also introduced a unique a hole in the
phone's shell to accommodate the Japanese perceived penchant for
personalizing their mobile phones. The Galaxy Feel's battery was also touted
as a major selling point since the market favors handsets with longer
battery life. The device is also waterproof and supports 1seg digital
broadcasts using an antenna that is sold separately.\n\n###\n\n",
"completion":"Looking for a smartphone that can do it all? Look no further
than Samsung Galaxy Feel! With a slim and sleek design, our latest
smartphone features high-quality picture and video capabilities, as well as
an award winning battery life. END"}
```

Here we used a multiline separator, as Wikipedia articles contain multiple paragraphs and headings. We also used a simple end token, to ensure that the model knows when the completion should finish.

Case study: Entity extraction

This is similar to a language transformation task. To improve the performance, it's best to either sort different extracted entities alphabetically or in the same order as they appear in the original text. This will help the model to keep track of all the entities which need to be generated in order. The dataset could look as follows:

JSON

```
{"prompt":"<any text, for example news article>\n\n###\n\n", "completion":"
<list of entities, separated by a newline> END"}
```

For example:

JSON

```
{"prompt":"Portugal will be removed from the UK's green travel list from
Tuesday, amid rising coronavirus cases and concern over a \"Nepal mutation
of the so-called Indian variant\". It will join the amber list, meaning
holidaymakers should not visit and returnees must isolate for 10
days...\n\n###\n\n", "completion":" Portugal\nUK\nNepal mutation\nIndian
variant END"}
```

A multi-line separator works best, as the text will likely contain multiple lines. Ideally there will be a high diversity of the types of input prompts (news articles, Wikipedia pages, tweets, legal documents), which reflect the likely texts which will be encountered when extracting entities.

Case study: Customer support chatbot

A chatbot will normally contain relevant context about the conversation (order details), summary of the conversation so far as well as most recent messages. For this use case the same past conversation can generate multiple rows in the dataset, each time with a slightly different context, for every agent generation as a completion. This use case will require a few thousand examples, as it will likely deal with different types of requests, and customer issues. To ensure the performance is of high quality we recommend vetting the conversation samples to ensure the quality of agent messages. The summary can be generated with a separate text transformation fine tuned model. The dataset could look as follows:

JSON

```
{"prompt":"Summary: <summary of the interaction so far>\n\nSpecific
information:<for example order details in natural
language>\n\n###\n\nCustomer: <message1>\nAgent: <response1>\nCustomer:
<message2>\nAgent:", "completion":" <response2>\n"}
{"prompt":"Summary: <summary of the interaction so far>\n\nSpecific
information:<for example order details in natural
language>\n\n###\n\nCustomer: <message1>\nAgent: <response1>\nCustomer:
<message2>\nAgent: <response2>\nCustomer: <message3>\nAgent:",
"completion":" <response3>\n"}
```

Here we purposefully separated different types of input information, but maintained Customer Agent dialog in the same format between a prompt and a completion. All the completions should only be by the agent, and we can use `\n` as a stop sequence when doing inference.

Case study: Product description based on a technical list of properties

Here it's important to convert the input data into a natural language, which will likely lead to superior performance. For example, the following format:

JSON

```
{"prompt":"Item=handbag, Color=army_green, price=$99, size=S->",
"completion":" This stylish small green handbag will add a unique touch to
your look, without costing you a
fortune."}
```

Won't work as well as:

JSON

```
{"prompt":"Item is a handbag. Colour is army green. Price is midrange. Size
is small.->", "completion":" This stylish small green handbag will add a
unique touch to your look, without costing you a fortune."}
```

For high performance, ensure that the completions were based on the description provided. If external content is often consulted, then adding such content in an automated way would improve the performance. If the description is based on images, it may help to use an algorithm to extract a textual description of the image. Since completions are only one sentence long, we can use `.` as the stop sequence during inference.

Open ended generation

For this type of problem we recommend:

- Leave the prompt empty.
- No need for any separators.
- You'll normally want a very large number of examples, at least a few thousand.
- Ensure the examples cover the intended domain or the desired tone of voice.

Case study: Maintaining company voice

Many companies will have a large amount of high quality content generated in a specific voice. Ideally all generations from our API should follow that voice for the different use cases. Here we can use the trick of leaving the prompt empty, and feeding in all the documents which are good examples of the company voice. A fine-tuned model can be used to solve a number of different use cases with similar prompts to the ones used for base models, but the outputs are going to follow the company voice much more closely than previously.

JSON

```
{"prompt":"", "completion":" <company voice textual content>"}
{"prompt":"", "completion":" <company voice textual content2>"}
```

A similar technique could be used for creating a virtual character with a particular personality, style of speech and topics the character talks about.

Generative tasks have a potential to leak training data when requesting completions from the model, so additional care needs to be taken that this is addressed appropriately. For example personal or sensitive company information should be replaced by generic information or not be included into fine-tuning in the first place.

Next steps

- Fine tune your model with our How-to guide
- Learn more about the underlying models that power Azure OpenAI

Learn how to customize a model for your application

Article • 12/14/2022 • 47 minutes to read

The Azure OpenAI Service lets you tailor our models to your personal datasets using a process known as *fine-tuning*. This customization step will let you get more out of the service by providing:

- Higher quality results than what you can get just from prompt design
- The ability to train on more examples than can fit into a prompt
- Lower-latency requests

A customized model improves on the few-shot learning approach by training the model's weights on your specific prompts and structure. The customized model lets you achieve better results on a wider number of tasks without needing to provide examples in your prompt. The result is less text sent and fewer tokens processed on every API call, saving cost and improving request latency.

> **Note**
>
> There is a breaking change in the `create` fine tunes command in the latest 12-01-2022 GA API. For the latest command syntax consult the **reference documentation**

Prerequisites

- An Azure subscription - Create one for free
- Access granted to the Azure OpenAI service in the desired Azure subscription

  Currently, access to this service is granted only by application. You can apply for access to the Azure OpenAI service by completing the form at https://aka.ms/oai/access. Open an issue on this repo to contact us if you have an issue.

- An Azure OpenAI resource

  For more information about creating a resource, see Create a resource and deploy a model using Azure OpenAI.

Fine-tuning workflow

The fine-tuning workflow in Azure OpenAI Studio requires the following steps:

1. Prepare your training and validation data
2. Use the **Create customized model** wizard in Azure OpenAI Studio to train your customized model
   a. Select a base model
   b. Choose your training data
   c. Optionally, choose your validation data
   d. Optionally, choose advanced options for your fine-tune job
   e. Review your choices and train your new customized model
3. Check the status of your customized model
4. Deploy your customized model for use
5. Use your customized model
6. Optionally, analyze your customized model for performance and fit

Prepare your training and validation data

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

The training and validation data you use **must** be formatted as a JSON Lines (JSONL) document in which each line represents a single prompt-completion pair. The OpenAI command-line interface (CLI) includes a data preparation tool that validates, gives suggestions, and reformats your training data into a JSONL file ready for fine-tuning.

Here's an example of the training data format:

JSON

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

In addition to the JSONL format, training and validation data files must be encoded in UTF-8 and include a byte-order mark (BOM), and the file must be less than 200 MB in size. For more information about formatting your training data, see Learn how to prepare your dataset for fine-tuning.

Creating your training and validation datasets

Designing your prompts and completions for fine-tuning is different from designing your prompts for use with any of our GPT-3 base models. Prompts for completion calls often use either detailed instructions or few-shot learning techniques, and consist of multiple examples. For fine-tuning, we recommend that each training example consists of a single input prompt and its desired completion output. You don't need to give detailed instructions or multiple completion examples for the same prompt.

The more training examples you have, the better. We recommend having at least 200 training examples. In general, we've found that each doubling of the dataset size leads to a linear increase in model quality.

For more information about preparing training data for various tasks, see Learn how to prepare your dataset for fine-tuning.

OpenAI CLI data preparation tool

We recommend using OpenAI's command-line interface (CLI) to assist with many of the data preparation steps. OpenAI has developed a tool that validates, gives suggestions, and reformats your data into a JSONL file ready for fine-tuning.

To install the CLI, run the following Python command:

```
Console

pip install --upgrade openai
```

To analyze your training data with the data preparation tool, run the following Python command, replacing `<LOCAL_FILE>` with the full path and file name of the training data file to be analyzed:

```
Console

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

This tool accepts files in the following data formats, if they contain a prompt and a completion column/key:

- Comma-separated values (CSV)
- Tab-separated values (TSV)
- Microsoft Excel workbook (XLSX)
- JavaScript Object Notation (JSON)
- JSON Lines (JSONL)

The tool reformats your training data and saves output into a JSONL file ready for fine-tuning, after guiding you through the process of implementing suggested changes.

Use the Create customized model wizard

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

Go to the Azure OpenAI Studio

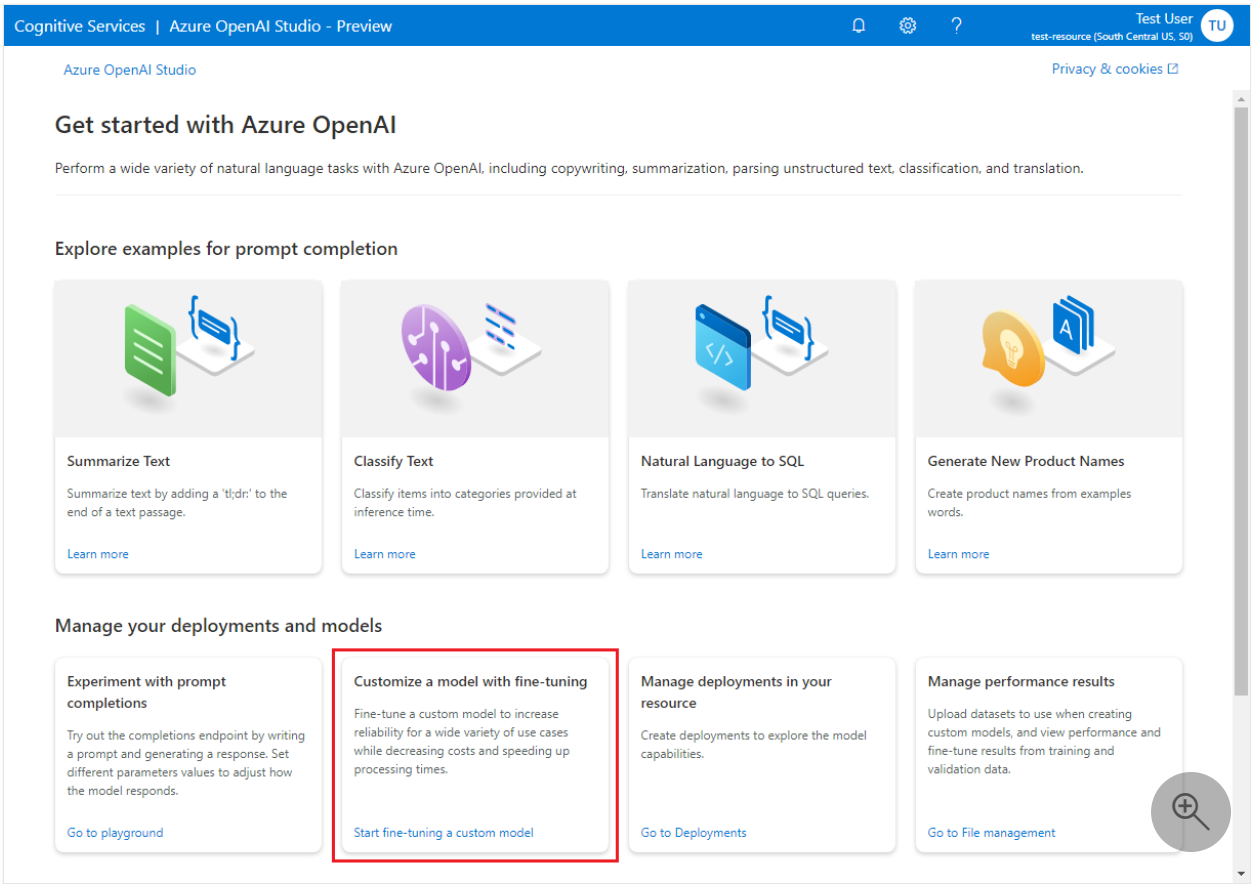
Navigate to the Azure OpenAI Studio at https://oai.azure.com/ and sign in with credentials that have access to your Azure OpenAI resource. During the sign-in workflow, select the appropriate directory, Azure subscription, and Azure OpenAI resource.

Landing page

You'll first land on our main page for the Azure OpenAI Studio. From here, you can start fine-tuning a custom model.

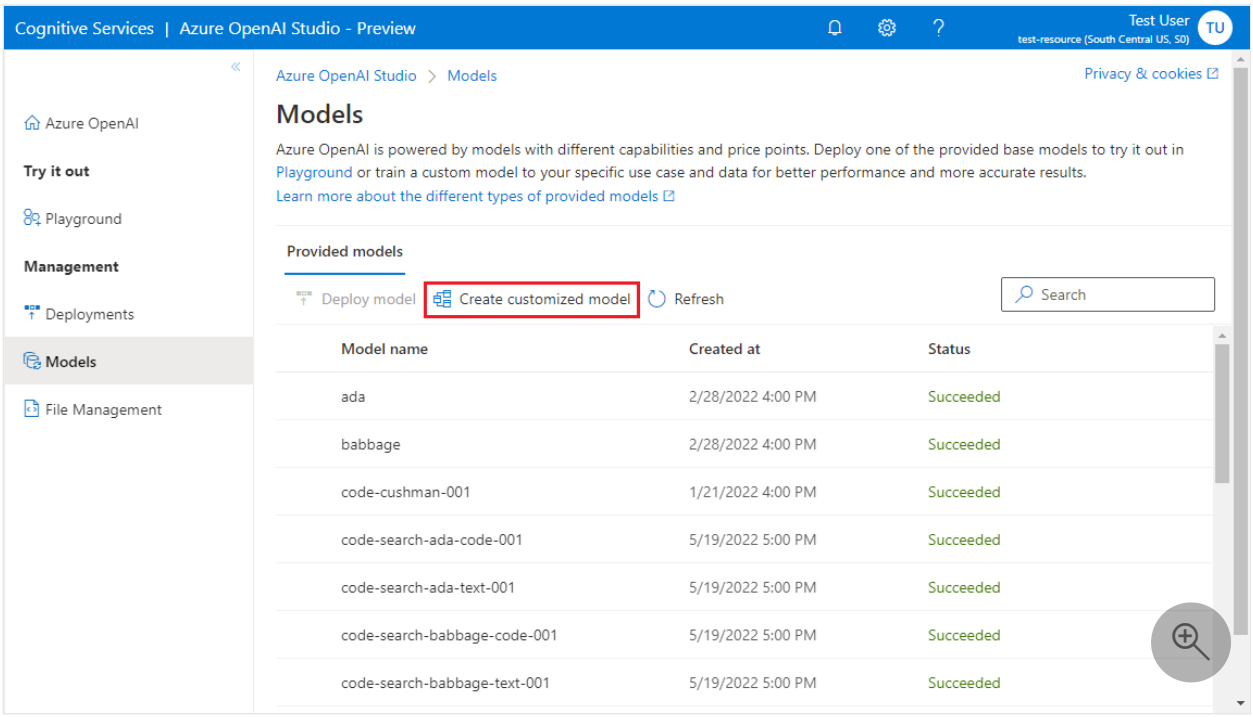
Select the **Start fine-tuning a custom model** button under **Manage deployments and models** section of the landing page, highlighted in the following picture, to start fine-tuning a custom model.

> **Note**
>
> If your resource doesn't have a model already deployed in it, a warning is displayed. You can ignore that warning for the purposes of fine-tuning a model, because you'll be fine-tuning and deploying a new customized model.



Start the wizard from the Models page

To create a customized model, select the **Create customized model** button under the **Provided models** section on the **Models** page, highlighted in the following picture, to start the **Create customized model** wizard.

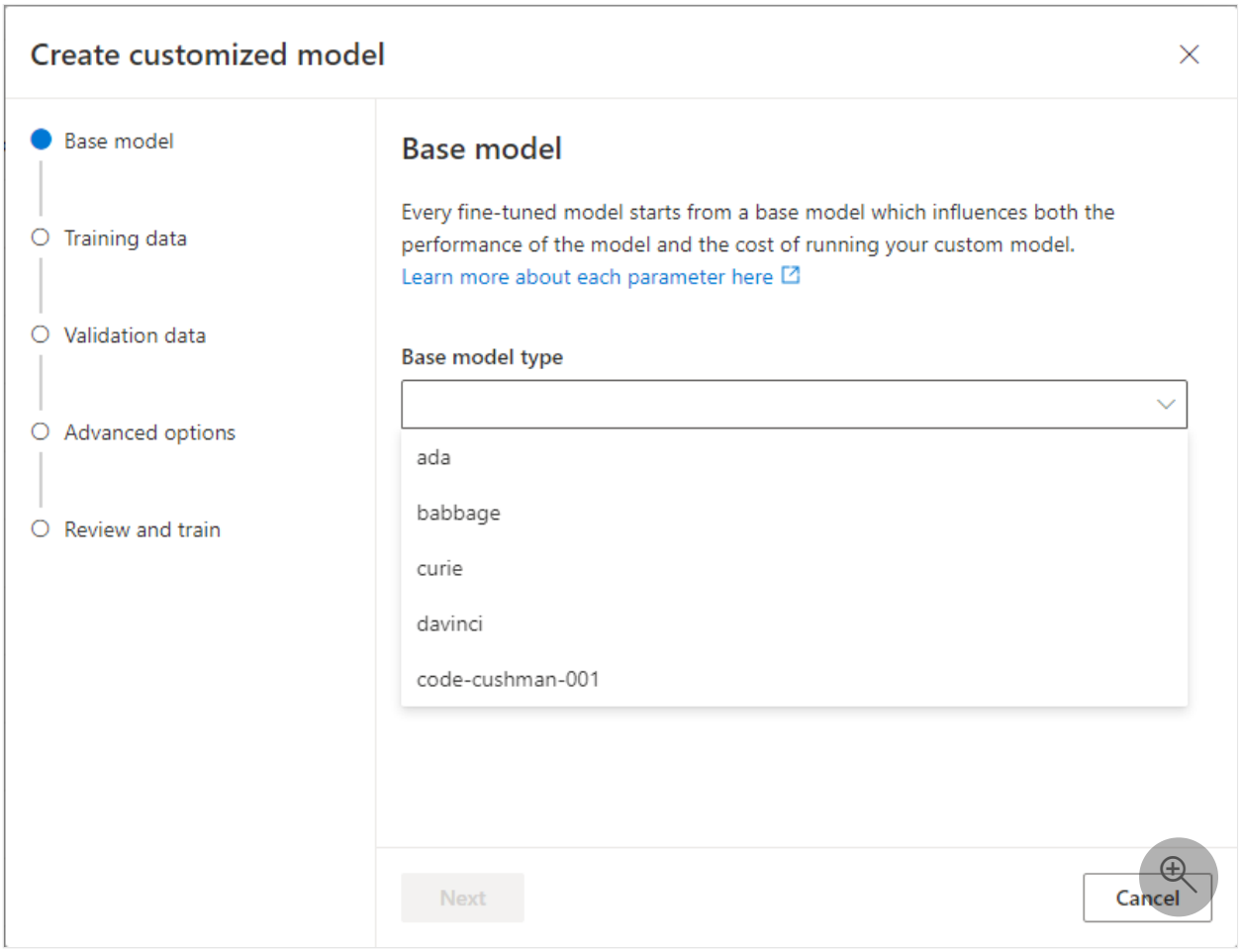


Select a base model

The first step in creating a customized model is to choose a base model. The **Base model** pane lets you choose a base model to use for your customized model, and the choice influences both the performance and the cost of your model. You can create a customized model from one of the following available base models:

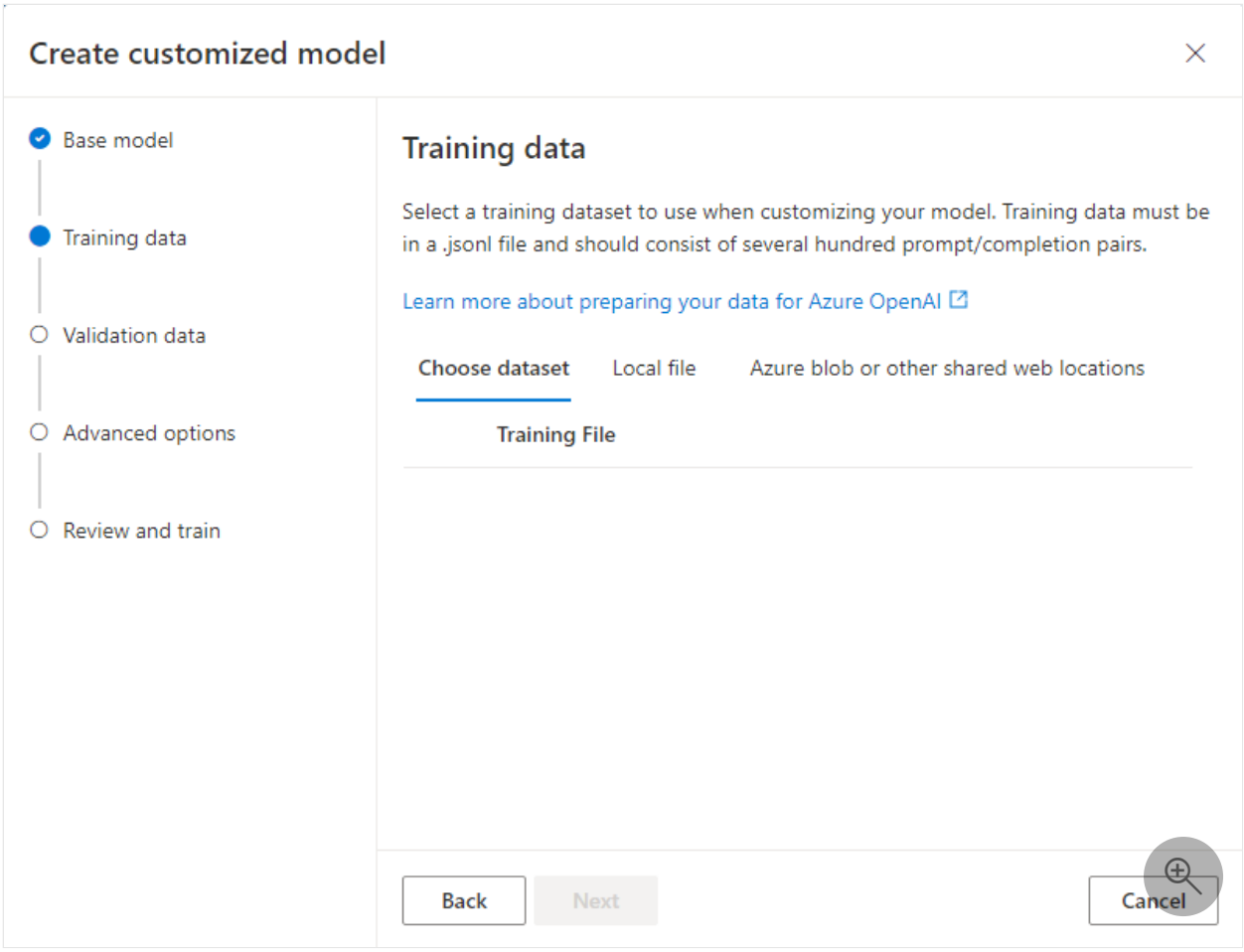
- `ada`
- `babbage`
- `curie`
- `code-cushman-001`*
- `davinci`* * available by request

For more information about our base models, see Models. Select a base model from the **Base model type** dropdown, as shown in the following picture, and then select **Next** to continue.



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, shown in the following picture, displays any existing, previously uploaded datasets and provides options by which you can upload new training data.



If your training data has already been uploaded to the service, select **Choose dataset**, and then select the file from the list shown in the **Training data** pane. Otherwise, select either **Local file** to upload training data from a local file, or **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 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. For more information about Azure Blob storage, see What is Azure Blob storage?

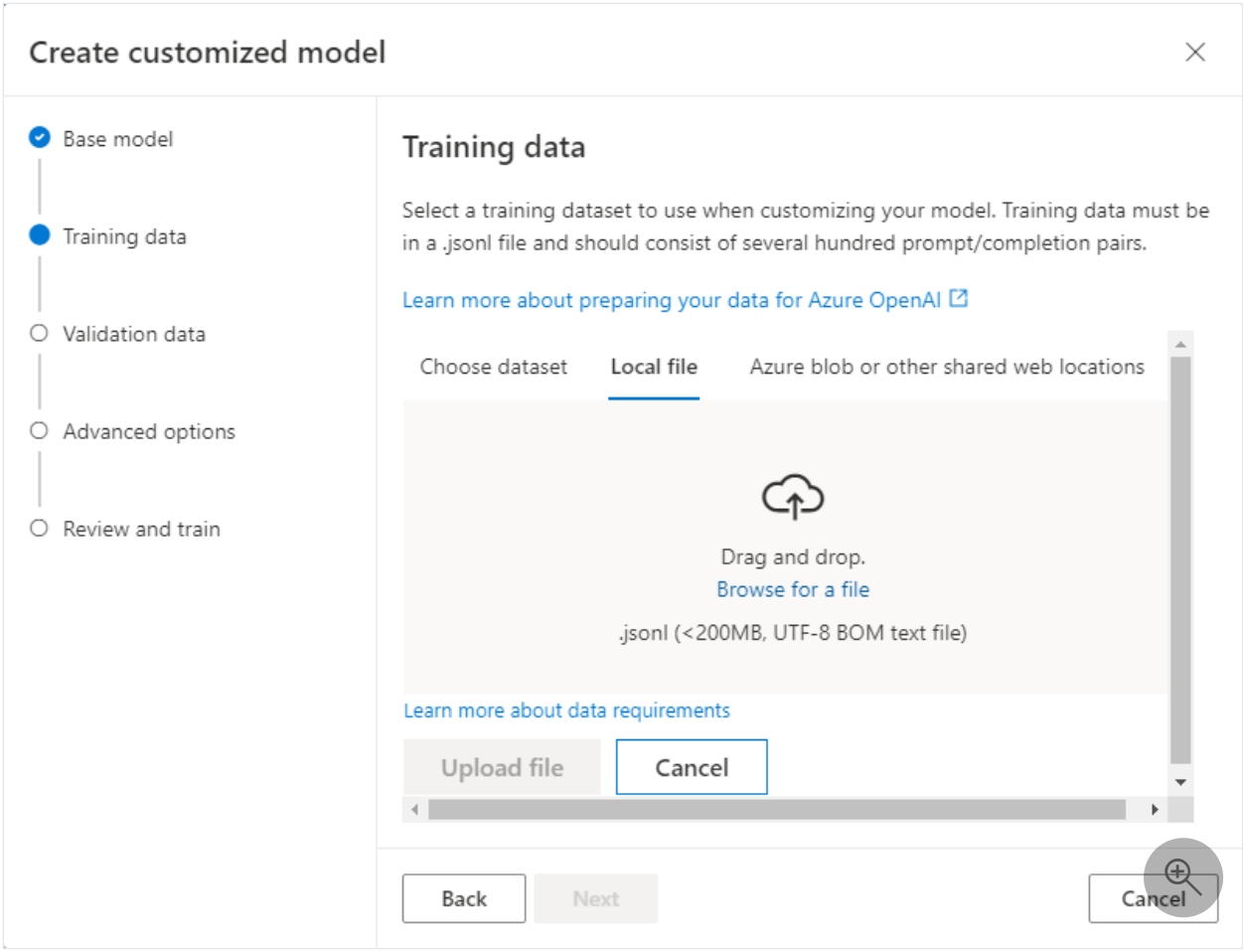
> **Note**
>
> Training data files must be formatted as JSONL files, encoded in UTF-8 with a byte-order mark (BOM), and less than 200 MB in size.

To upload training data from a 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**.

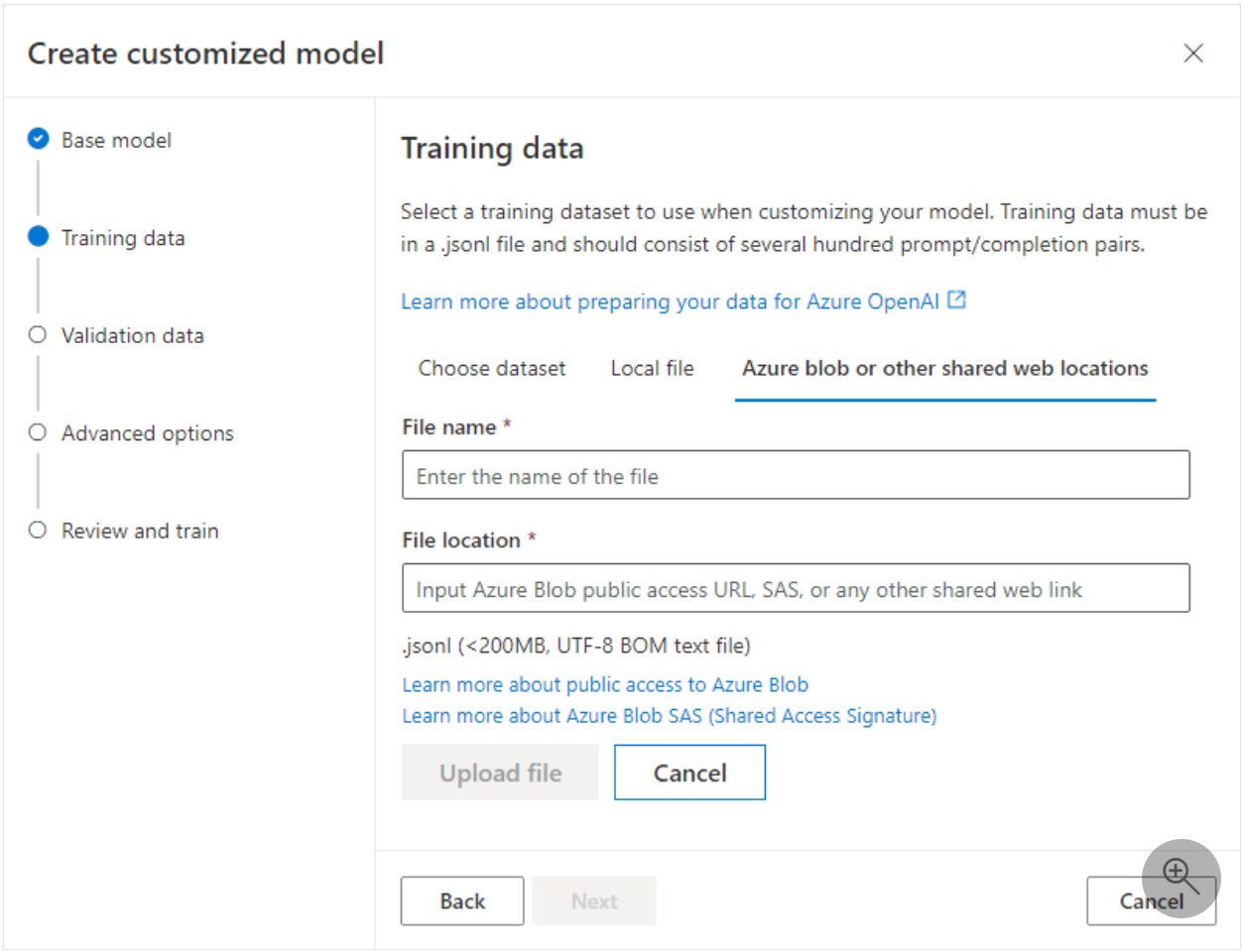
After you've selected and uploaded the training dataset, select **Next** to optionally choose your validation data.



To import training data from an Azure Blob store

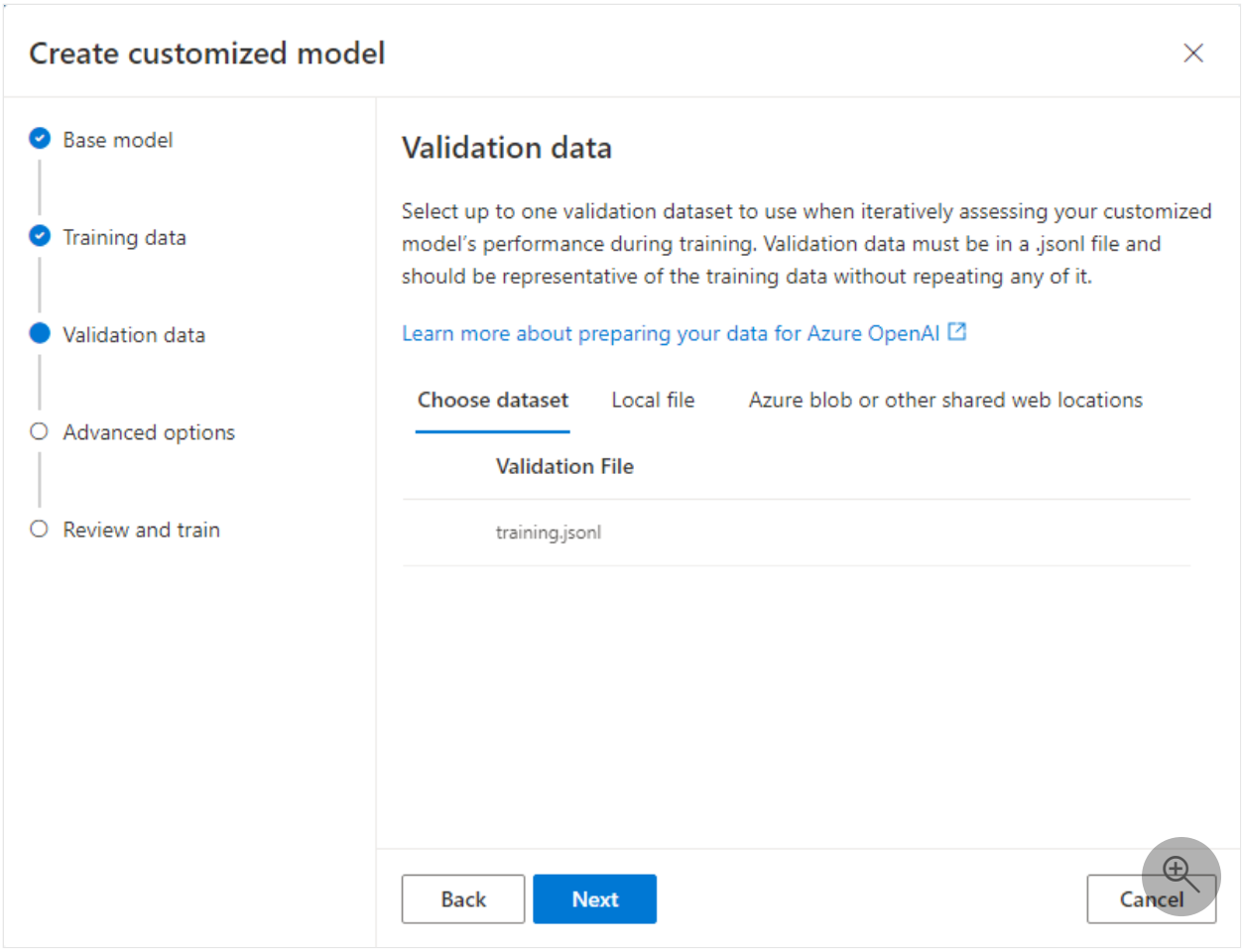
You can import a training dataset from Azure Blob or another shared web location by providing the name and location of the file, as shown in the following picture. Enter the name of the file in **File name** and the Azure Blob URL, Azure Storage shared access signature (SAS), or other link to an accessible shared web location that contains the file in **File location**, then select **Upload file** to import the training dataset to the service.

After you've selected and uploaded the training dataset, select **Next** to optionally choose your validation data.



Choose your validation data

You can now choose to optionally use validation data in the training process of your fine-tuned model. If you don't want to use validation data, you can choose **Next** to choose advanced options for your 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, shown in the following picture, displays any existing, previously uploaded training and validation datasets and provides options by which you can upload new validation data.



If your validation data has already been uploaded to the service, select **Choose dataset**, and then select the file from the list shown in the **Validation data** pane. Otherwise, select either **Local file** to upload validation data from a local file, or **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 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.

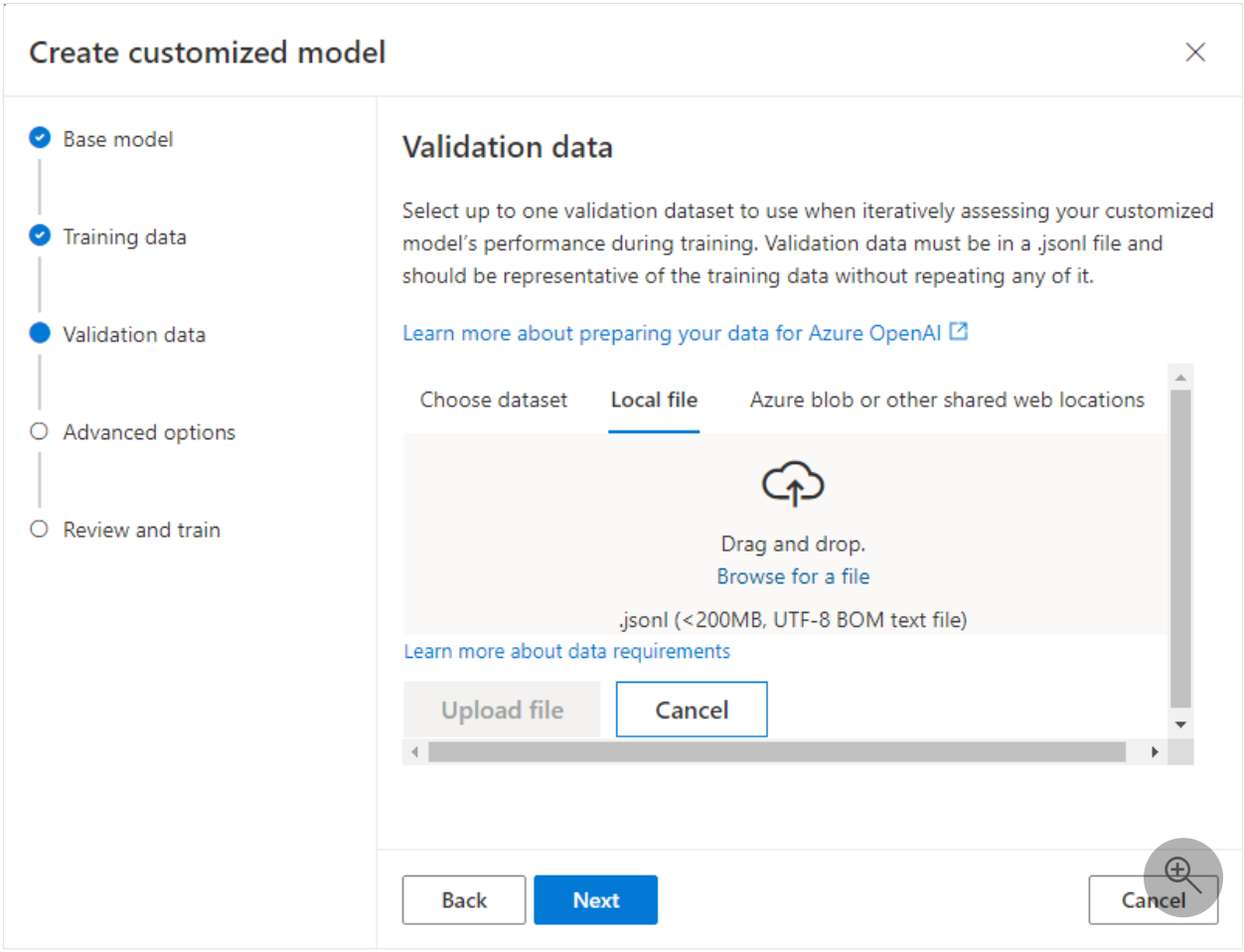
> **Note**
>
> Like training data files, validation data files must be formatted as JSONL files, encoded in UTF-8 with a byte-order mark (BOM), and less than 200 MB in size.

To upload validation data from a 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**.

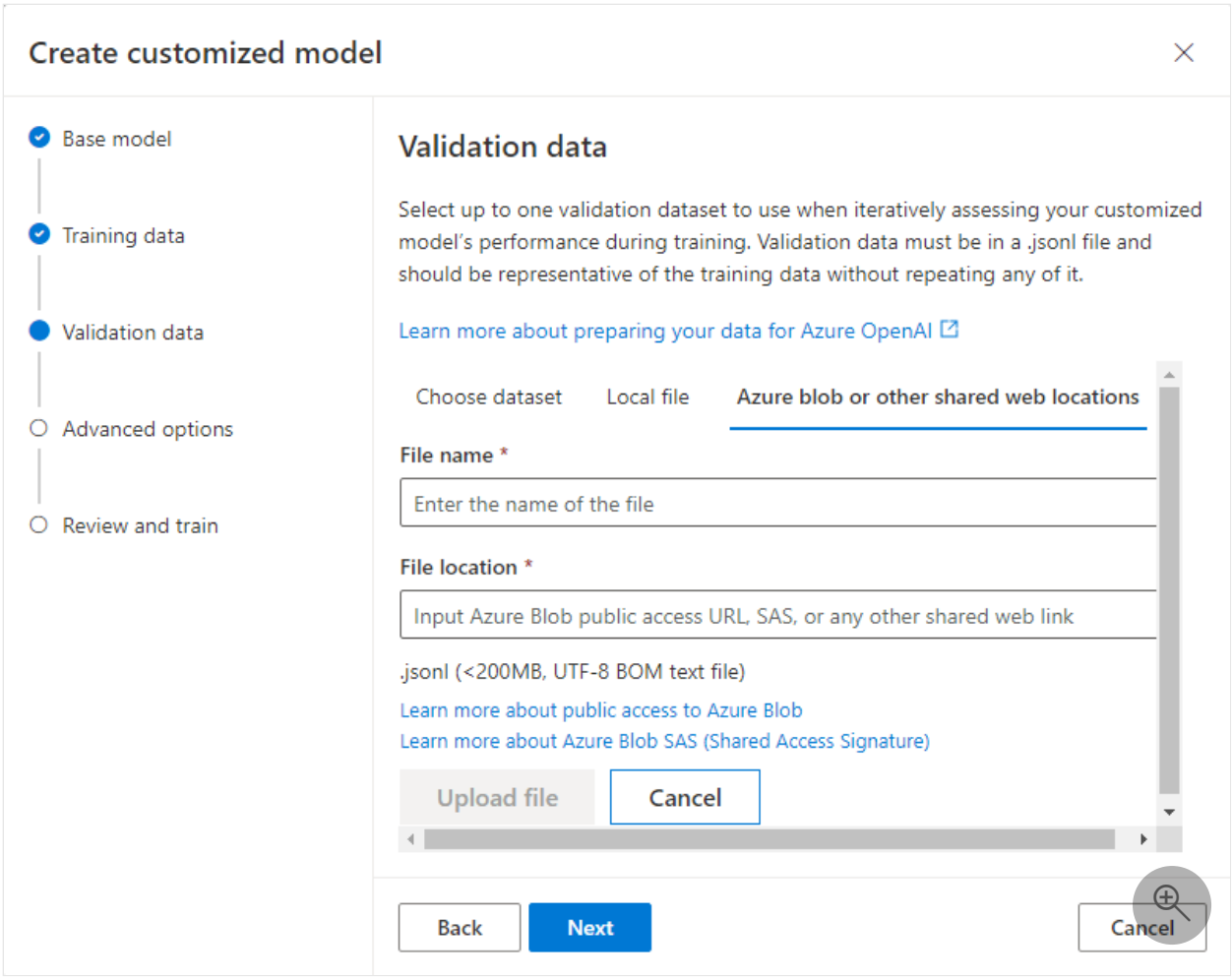
After you've uploaded the validation dataset, select **Next** to optionally choose advanced options.



To import validation data from an Azure Blob store

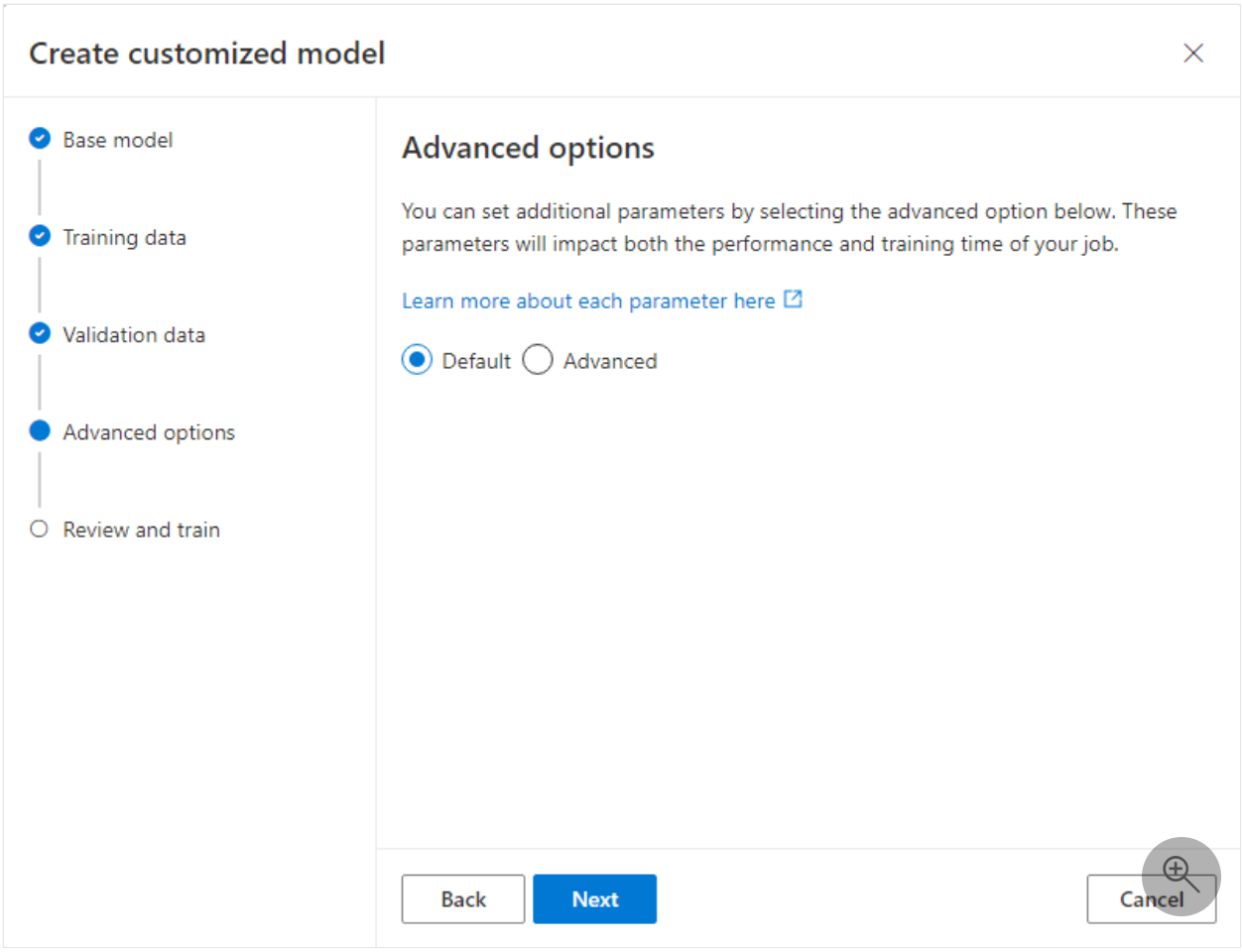
You can import a validation dataset from Azure Blob or another shared web location by providing the name and location of the file, as shown in the following picture. Enter the name of the file in **File name** and the Azure Blob URL, Azure Storage shared access signature (SAS), or other link to an accessible shared web location that contains the file in **File location**, then select **Upload file** to import the validation dataset to the service.

After you've imported the validation dataset, select **Next** to optionally choose advanced options.

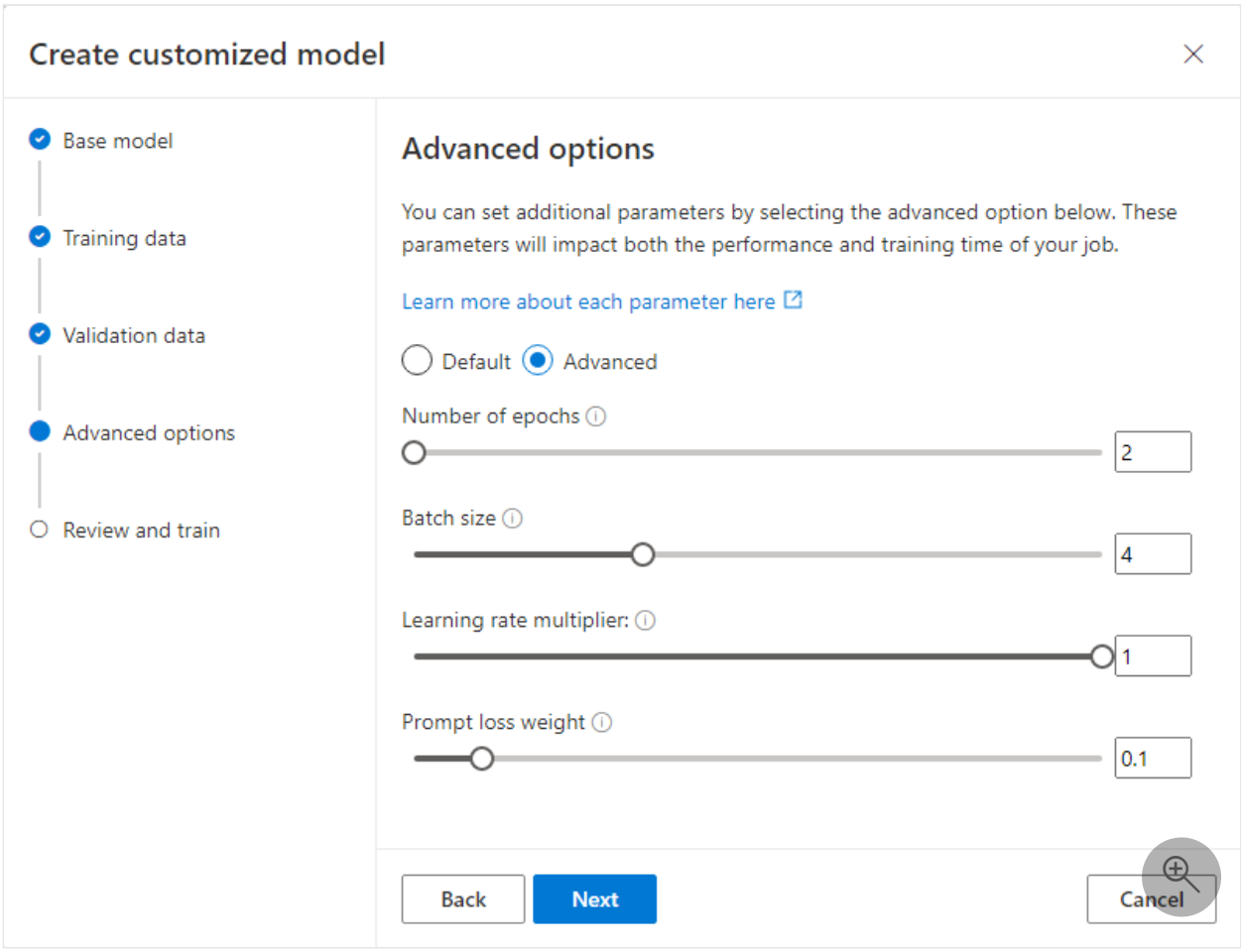


Choose advanced options

You can either use default values for the hyperparameters of the fine-tune job that the wizard runs to train your fine-tuned model, or you can adjust those hyperparameters for your customization needs in the **Advanced options** pane, shown in the following picture.



Either select **Default** to use the default values for the fine-tune job, or select **Advanced** to display and edit the hyperparameter values, as shown in the following picture.



The following hyperparameters are available:

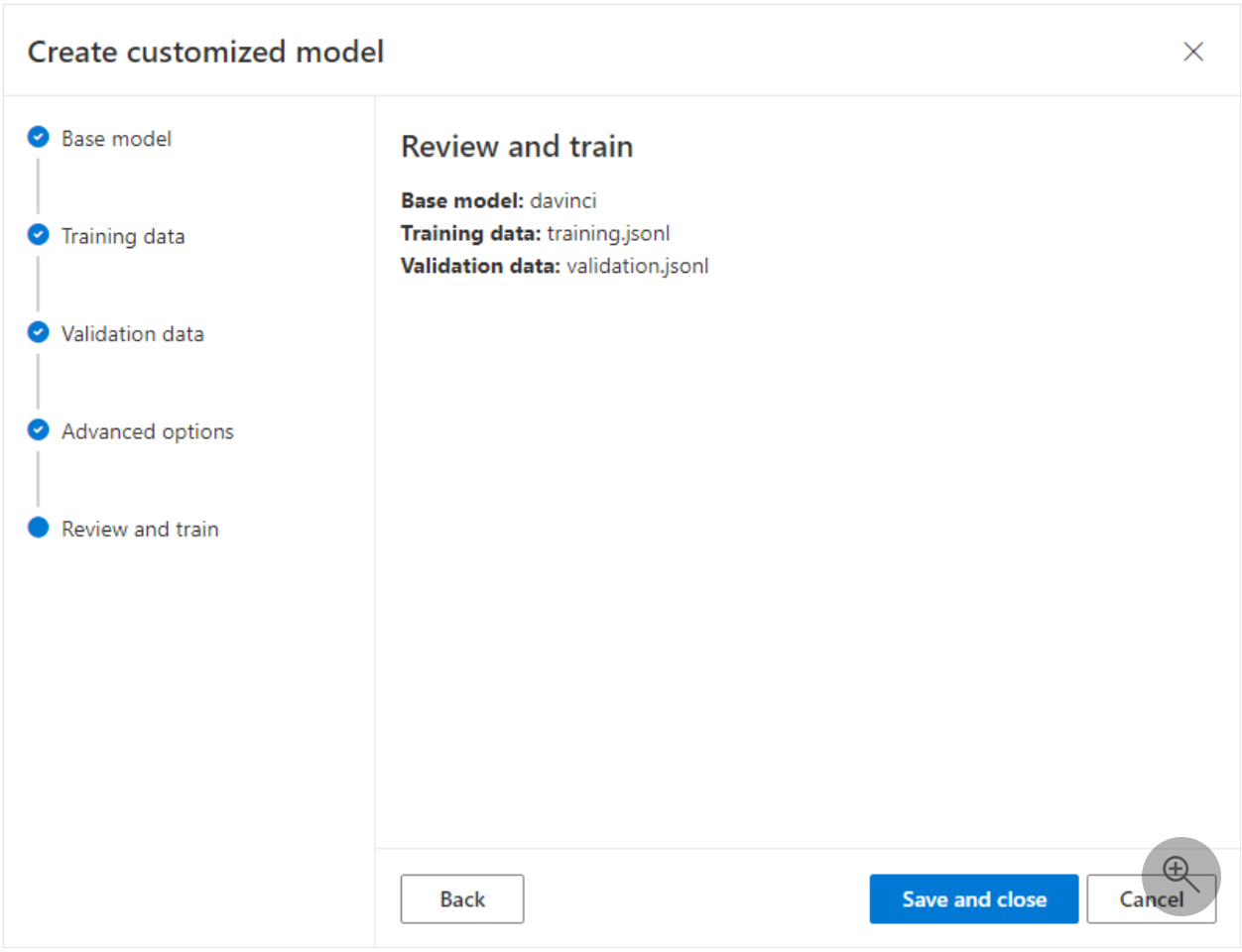
Parameter name	Description
Number of epochs	The number of epochs to train the model for. An epoch refers to one full cycle through the training dataset.
Batch size	The batch size to use for training. The batch size is the number of training examples used to train a single forward and backward pass.
Learning rate multiplier	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.
Prompt loss weight	The weight to use for loss on the prompt tokens. This value controls how much the model tries to learn to generate the prompt (as compared to the completion, which always has a weight of 1.0.) Increasing this value can add a stabilizing effect to training when completions are short.

For more information about these hyperparameters, see the Create a Fine tune job section of the REST API documentation.

After you've chosen either default or advanced options, select **Next** to review your choices and train your fine-tuned model.

Review your choices and train your model

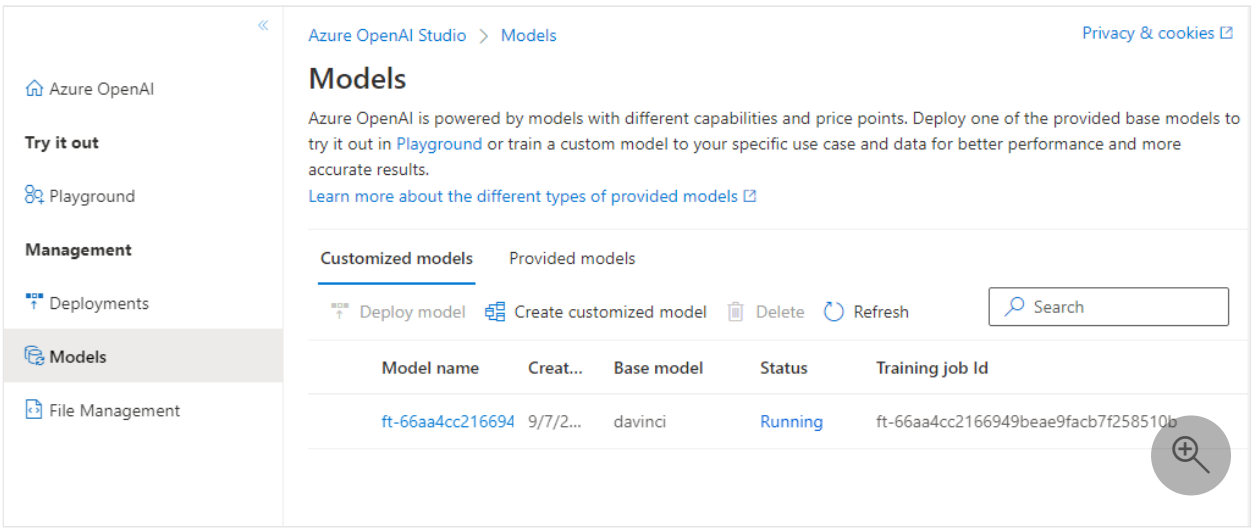
The **Review and train** pane of the wizard displays information about the choices you've made in the **Create customized model** wizard for your fine-tuned model, as shown in the following picture.



If you're ready to train your model, select **Save and close** to start the fine-tune job and return to the Models page.

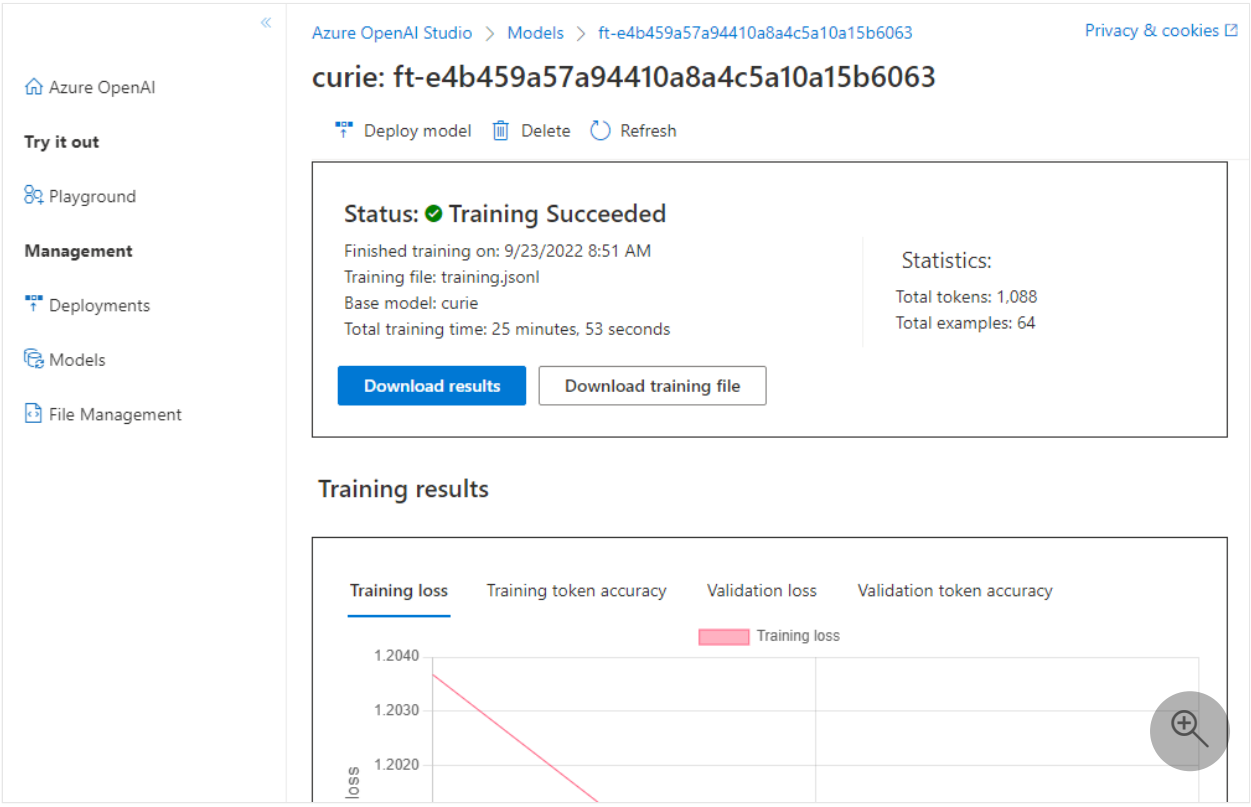
Check the status of your customized model

The **Models** page displays information about your customized model in the **Customized models** tab, as shown in the following picture. The tab includes information about the status and job ID of the fine-tune job for your customized model. When the job is completed, the file ID of the result file is also displayed.



After you've started a fine-tune job, it may take some time to complete. Your job may be queued behind other jobs on our system, and training your model can take minutes or hours depending on the model and dataset size. You can check the status of the fine-tune job for your customized model in the **Status** column of the **Customized models** tab on the **Models** page, and you can select **Refresh** to update the information on that page.

You can also select the name of the model from the **Model name** column of the **Models** page to display more information about your customized model, including the status of the fine-tune job, training results, training events, and hyperparameters used in the job. You can select the **Refresh** button to refresh the information for your model, as shown in the following picture.



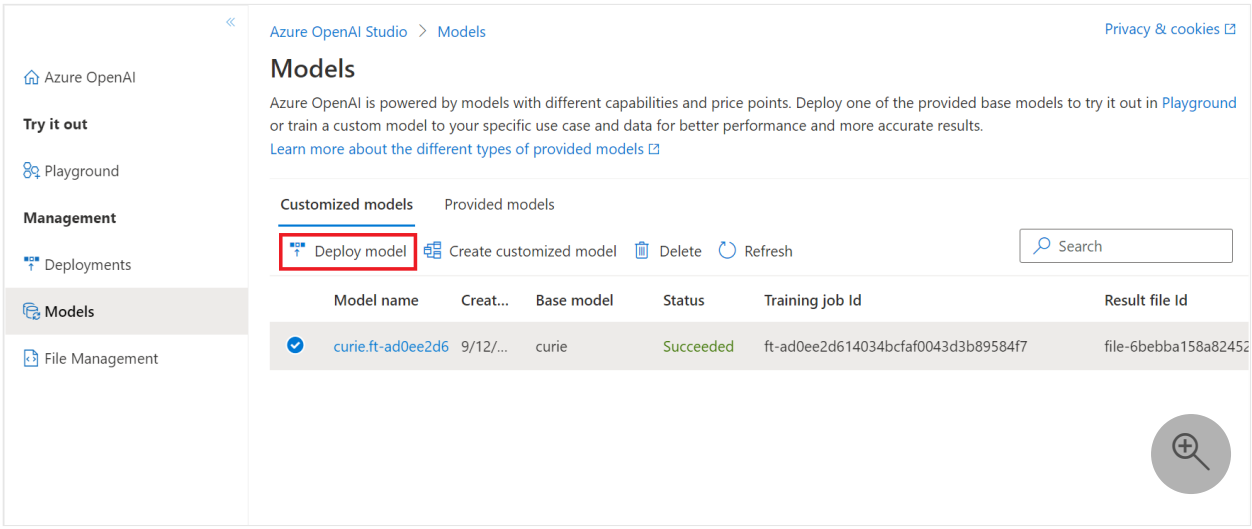
From the model page, you can also select **Download training file** to download the training data you used for the model, or select **Download results** to download the result file attached to the fine-tune job for your model and analyze your customized model for training and validation performance.

Deploy a customized model

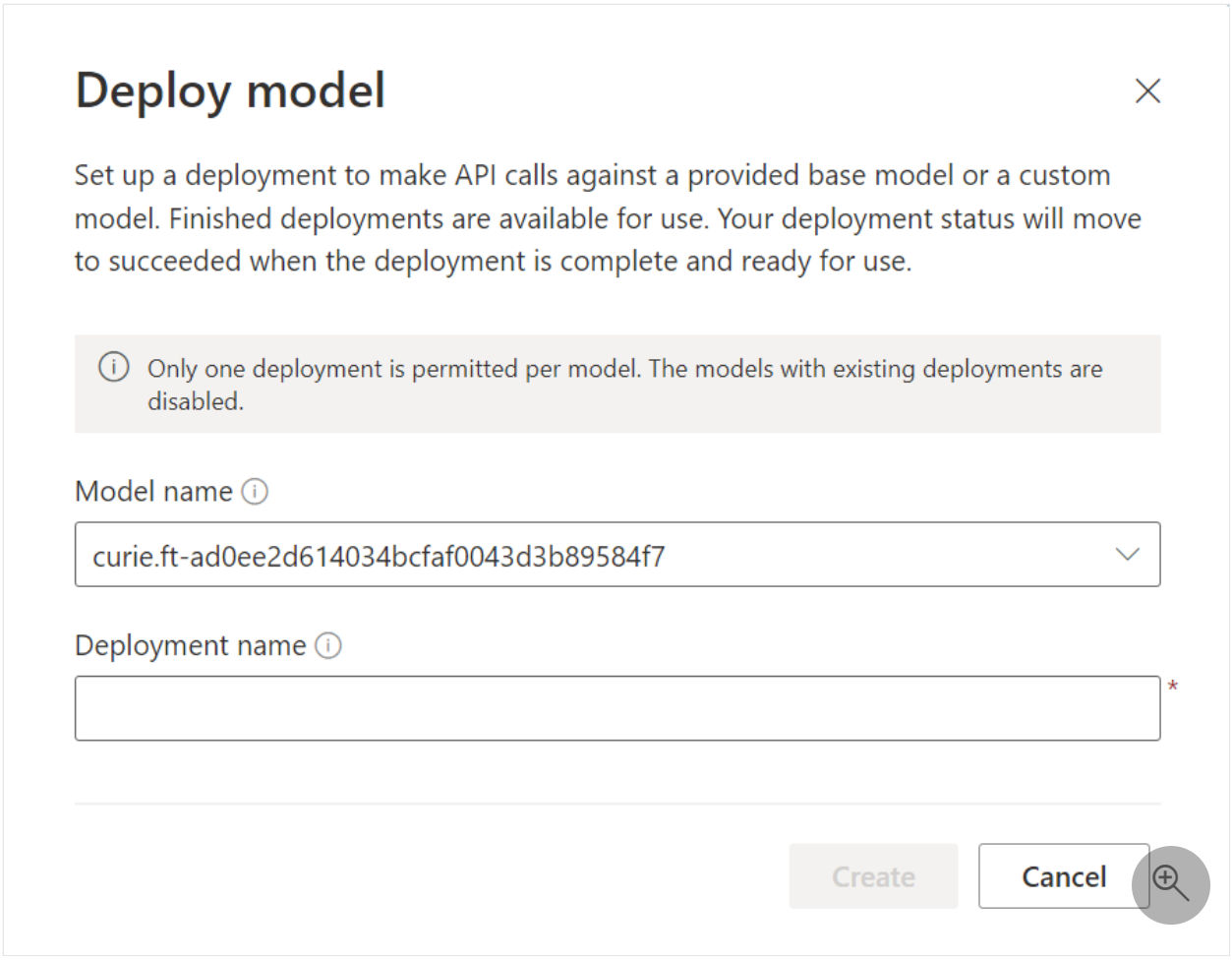
When the fine-tune job has succeeded, you can deploy the customized model from the **Models** pane. You must deploy your customized model to make it available for use with completion calls.

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

To deploy your customized model, select the customized model to be deployed and then select **Deploy model**, as shown in the following picture.



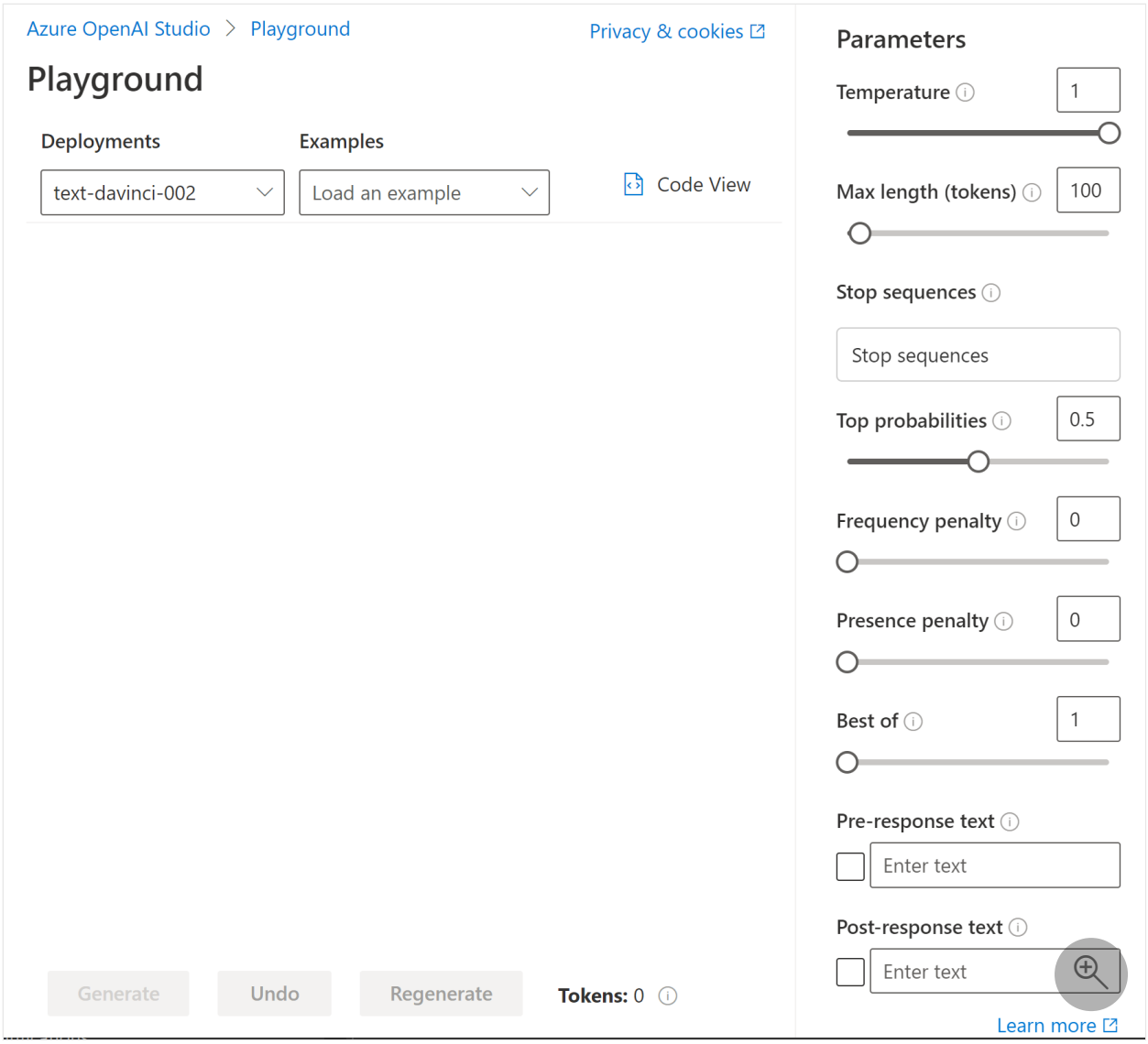
The **Deploy model** dialog is presented, in which you can provide a name for the deployment of your customized model. Enter a name in **Deployment name** and then select **Create** to start the deployment of your customized model.



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

Use a deployed customized model

Once your customized model has been deployed, you can use it like any other deployed model. For example, you can use the **Playground** pane of Azure OpenAI Studio to experiment with your new deployment, as shown in the following picture. You can continue to use the same parameters with your customized model, such as temperature and frequency penalty, as you can with other deployed models.



> **Note**
>
> As with all applications, we require a review process prior to going live.

Analyze your customized model

Azure OpenAI attaches a result file, named `results.csv`, to each fine-tune job once it's completed. You can use the result file to analyze the training and validation performance of your customized model. The file ID for the result file is listed for each customized model in the **Result file Id** column of the **Models** pane for Azure OpenAI Studio. You can use the file ID to identify and download the result file from the **File Management** pane of Azure OpenAI Studio.

The result file is a CSV file containing a header row and a row for each training step performed by the fine-tune 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.
`elapsed_tokens`	The number of tokens the customized model has seen so far, including repeats.
`elapsed_examples`	The number of examples the model has seen so far, including repeats. Each example represents one element in that step's batch of training data. For example, if the **Batch size** parameter is set to 32 in the **Advanced options** pane, this value increments by 32 in each training step.
`training_loss`	The loss for the training batch.
`training_sequence_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]]`.
`training_token_accuracy`	The percentage of tokens in the training batch that were 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]]`.
`validation_loss`	The loss for the validation batch.
`validation_sequence_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_token_accuracy`	The percentage of tokens in the validation batch that were 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]]`.

Clean up your deployments, customized models, and training files

When you're done with your customized model, you can delete the deployment and model. You can also delete the training and validation files you uploaded to the service, if needed.

Delete your model deployment

You can delete the deployment for your customized model from the **Deployments** page for Azure OpenAI Studio. Select the deployment to delete, and then select **Delete** to delete the deployment.

Delete your customized model

You can delete a customized model from the **Models** page for Azure OpenAI Studio. Select the customized model to delete from the **Customized models** tab, and then select **Delete** to delete the customized model.

> **Note**
>
> You cannot delete a customized model if it has an existing deployment. You must first **delete your model deployment** before you can delete your customized model.

Delete your training files

You can optionally delete training and validation files you've uploaded for training, and result files generated during training, from the **File Management** page for Azure OpenAI Studio. Select the file to delete, and then select **Delete** to delete the file.

Next steps

- Explore the full REST API Reference documentation to learn more about all the fine-tuning capabilities. You can find the full REST documentation here.
- Explore more of the Python SDK operations here.

Additional resources

Documentation

How to prepare a dataset for custom model training - Azure OpenAI

Learn how to prepare your dataset for fine-tuning

How to generate embeddings with Azure OpenAI - Azure OpenAI

Learn how to generate embeddings with Azure OpenAI

Azure OpenAI REST API reference - Azure OpenAI

Learn how to use the Azure OpenAI REST API. In this article, you'll learn about authorization options, how to structure a request and receive a response.

Azure OpenAI models - Azure OpenAI

Learn about the different models that are available in Azure OpenAI.

Quickstart - Deploy a model and generate text using Azure OpenAI - Azure OpenAI

Walkthrough on how to get started with Azure OpenAI and make your first completions call.

How to generate text with Azure OpenAI - Azure OpenAI

Learn how to generate or manipulate text, including code with Azure OpenAI

How-to - Use Azure OpenAI with large datasets - Azure OpenAI

Walkthrough on how to integrate Azure OpenAI with SynapseML and Apache Spark to apply large language models at a distributed scale.

How to Configure Azure OpenAI with Managed Identities - Azure OpenAI

Provides guidance on how to set managed identity with Azure Active Directory

Show 5 more

How to Configure Azure OpenAI with Managed Identities

Article • 12/14/2022 • 2 minutes to read

More complex security scenarios require Azure role-based access control (Azure RBAC). This document covers how to authenticate to your OpenAI resource using Azure Active Directory (Azure AD).

In the following sections, you'll use the Azure CLI to assign roles, and obtain a bearer token to call the OpenAI resource. If you get stuck, links are provided in each section with all available options for each command in Azure Cloud Shell/Azure CLI.

Prerequisites

- An Azure subscription - Create one for free
- Access granted to the Azure OpenAI service in the desired Azure subscription

  Currently, access to this service is granted only by application. You can apply for access to the Azure OpenAI service by completing the form at https://aka.ms/oai/access. Open an issue on this repo to contact us if you have an issue.
- Azure CLI - Installation Guide
- The following Python libraries: os, requests, json

Sign into the Azure CLI

To sign-in to the Azure CLI, run the following command and complete the sign-in. You may need to do it again if your session has been idle for too long.

Azure CLI

```
az login
```

Assign yourself to the Cognitive Services User role

Assigning yourself to the Cognitive Services User role will allow you to use your account for access to the specific cognitive services resource

1. Get your user information

Azure CLI

```
export user=$(az account show | jq -r .user.name)
```

2. Assign yourself to “Cognitive Services User” role.

Azure CLI

```
export resourceId=$(az group show -g $myResourceGroupName | jq -r .id)
az role assignment create --role "Cognitive Services User" --assignee
$user --scope $resourceId
```

> ⓘ Note
>
> Role assignment change will take ~5 mins to become effective. Therefore, I did this step ahead of time. Skip this if you have already done this previously.

3. Acquire an Azure AD access token. Access tokens expire in one hour. you'll then need to acquire another one.

Azure CLI

```
export accessToken=$(az account get-access-token --resource
https://cognitiveservices.azure.com | jq -r .accessToken)
```

4. Make an API call Use the access token to authorize your API call by setting the `Authorization` header value.

Bash

```
curl ${endpoint%/}/openai/deployment/YOUR_DEPLOYMENT_NAME/completions?
api-version=2022-12-01 \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $accessToken" \
-d '{ "prompt": "Once upon a time" }'
```

Authorize access to managed identities

OpenAI supports Azure Active Directory (Azure AD) authentication with managed identities for Azure resources. Managed identities for Azure resources can authorize access to Cognitive Services resources using Azure AD credentials from applications running in Azure virtual machines (VMs), function apps, virtual machine scale sets, and other services. By using managed identities for Azure resources together with Azure AD authentication, you can avoid storing credentials with your applications that run in the cloud.

Enable managed identities on a VM

Before you can use managed identities for Azure resources to authorize access to Cognitive Services resources from your VM, you must enable managed identities for Azure resources on the VM. To learn how to enable managed identities for Azure Resources, see:

- Azure portal
- Azure PowerShell
- Azure CLI
- Azure Resource Manager template
- Azure Resource Manager client libraries

For more information about managed identities, see Managed identities for Azure resources.

Use Azure OpenAI with large datasets

Article • 08/30/2022 • 5 minutes to read

The Azure OpenAI service can be used to solve a large number of natural language tasks through prompting the completion API. To make it easier to scale your prompting workflows from a few examples to large datasets of examples, we have integrated the Azure OpenAI service with the distributed machine learning library SynapseML. This integration makes it easy to use the Apache Spark distributed computing framework to process millions of prompts with the OpenAI service. This tutorial shows how to apply large language models at a distributed scale using Azure Open AI and Azure Synapse Analytics.

Prerequisites

- An Azure subscription - Create one for free
- Access granted to the Azure OpenAI service in the desired Azure subscription

  Currently, access to this service is granted only by application. You can apply for access to the Azure OpenAI service by completing the form at https://aka.ms/oai/access. Open an issue on this repo to contact us if you have an issue.
- An Azure OpenAI resource – create a resource
- An Apache Spark cluster with SynapseML installed - create a serverless Apache Spark pool here

We recommend creating a Synapse workspace, but an Azure Databricks, HDInsight, or Spark on Kubernetes, or even a Python environment with the `pyspark` package, will also work.

Import this guide as a notebook

The next step is to add this code into your Spark cluster. You can either create a notebook in your Spark platform and copy the code into this notebook to run the demo, or download the notebook and import it into Synapse Analytics.

1. Download this demo as a notebook (click Raw, then save the file)
2. Import the notebook into the Synapse Workspace or, if using Databricks, into the Databricks Workspace

3. Install SynapseML on your cluster. See the installation instructions for Synapse at the bottom of the SynapseML website. This requires pasting another cell at the top of the notebook you imported
4. Connect your notebook to a cluster and follow along, editing and running the cells below.

Fill in your service information

Next, edit the cell in the notebook to point to your service. In particular, set the `resource_name`, `deployment_name`, `location`, and `key` variables to the corresponding values for your Azure OpenAI service.

> **Important**
>
> Remember to remove the key from your code when you're done, and never post it publicly. For production, use a secure way of storing and accessing your credentials like **Azure Key Vault**. See the Cognitive Services **security** article for more information.

Python

```python
import os

# Replace the following values with your Azure OpenAI service information
resource_name = "RESOURCE_NAME"      # The name of your Azure OpenAI
resource.
deployment_name = "DEPLOYMENT_NAME"  # The name of your Azure OpenAI
deployment.
location = "RESOURCE_LOCATION"       # The location or region ID for your
resource.
key = "RESOURCE_API_KEY"             # The key for your resource.

assert key is not None and resource_name is not None
```

Create a dataset of prompts

Next, create a dataframe consisting of a series of rows, with one prompt per row.

You can also load data directly from Azure Data Lake Storage (ADLS) or other databases. For more information about loading and preparing Spark dataframes, see the Apache Spark data loading guide.

Python

```
df = spark.createDataFrame(
    [
        ("Hello my name is",),
        ("The best code is code that's",),
        ("SynapseML is ",),
    ]
).toDF("prompt")
```

Create the OpenAICompletion Apache Spark client

To apply the OpenAI Completion service to the dataframe that you just created, create an `OpenAICompletion` object that serves as a distributed client. Parameters of the service can be set either with a single value, or by a column of the dataframe with the appropriate setters on the `OpenAICompletion` object. Here, we're setting `maxTokens` to 200. A token is around four characters, and this limit applies to the sum of the prompt and the result. We're also setting the `promptCol` parameter with the name of the prompt column in the dataframe.

Python

```
from synapse.ml.cognitive import OpenAICompletion

completion = (
    OpenAICompletion()
    .setSubscriptionKey(key)
    .setDeploymentName(deployment_name)
    .setUrl("https://{}.openai.azure.com/".format(resource_name))
    .setMaxTokens(200)
    .setPromptCol("prompt")
    .setErrorCol("error")
    .setOutputCol("completions")
)
```

Transform the dataframe with the OpenAICompletion client

Now that you have the dataframe and the completion client, you can transform your input dataset and add a column called `completions` with all of the information the service adds. We'll select out just the text for simplicity.

Python

```
from pyspark.sql.functions import col

completed_df = completion.transform(df).cache()
display(completed_df.select(
  col("prompt"), col("error"),
col("completions.choices.text").getItem(0).alias("text")))
```

Your output should look something like the following example; note that the completion text can vary.

prompt	error	text
Hello my name is	undefined	Makaveli I'm eighteen years old and I want to be a rapper when I grow up I love writing and making music I'm from Los Angeles, CA
The best code is code that's	undefined	understandable This is a subjective statement, and there is no definitive answer.
SynapseML is	undefined	A machine learning algorithm that is able to learn how to predict the future outcome of events.

Other usage examples

Improve throughput with request batching

The example above makes several requests to the service, one for each prompt. To complete multiple prompts in a single request, use batch mode. First, in the `OpenAICompletion` object, instead of setting the Prompt column to "Prompt", specify "batchPrompt" for the BatchPrompt column. To do so, create a dataframe with a list of prompts per row.

> **Note**
>
> There is currently a limit of 20 prompts in a single request and a limit of 2048 "tokens", or approximately 1500 words.

Python

```
batch_df = spark.createDataFrame(
    [
        (["The time has come", "Pleased to", "Today stocks", "Here's to"],),
```

```
        (["The only thing", "Ask not what", "Every litter", "I am"],),
    ]
).toDF("batchPrompt")
```

Next we create the `OpenAICompletion` object. Rather than setting the prompt column, set the batchPrompt column if your column is of type `Array[String]`.

Python

```
batch_completion = (
    OpenAICompletion()
    .setSubscriptionKey(key)
    .setDeploymentName(deployment_name)
    .setUrl("https://{}.openai.azure.com/".format(resource_name))
    .setMaxTokens(200)
    .setBatchPromptCol("batchPrompt")
    .setErrorCol("error")
    .setOutputCol("completions")
)
```

In the call to transform, a request will then be made per row. Because there are multiple prompts in a single row, each request will be sent with all prompts in that row. The results will contain a row for each row in the request.

Python

```
completed_batch_df = batch_completion.transform(batch_df).cache()
display(completed_batch_df)
```

> **Note**
>
> There is currently a limit of 20 prompts in a single request and a limit of 2048 "tokens", or approximately 1500 words.

Using an automatic mini-batcher

If your data is in column format, you can transpose it to row format using SynapseML's `FixedMiniBatcherTransformer`.

Python

```
from pyspark.sql.types import StringType
from synapse.ml.stages import FixedMiniBatchTransformer
from synapse.ml.core.spark import FluentAPI
```

```
completed_autobatch_df = (df
 .coalesce(1) # Force a single partition so that our little 4-row dataframe
makes a batch of size 4, you can remove this step for large datasets
 .mlTransform(FixedMiniBatchTransformer(batchSize=4))
 .withColumnRenamed("prompt", "batchPrompt")
 .mlTransform(batch_completion))

display(completed_autobatch_df)
```

Prompt engineering for translation

The Azure OpenAI service can solve many different natural language tasks through prompt engineering. Here, we show an example of prompting for language translation:

Python

```
translate_df = spark.createDataFrame(
    [
        ("Japanese: Ookina hako \nEnglish: Big box \nJapanese: Midori
tako\nEnglish:",),
        ("French: Quelle heure est-il à Montréal? \nEnglish: What time is it
in Montreal? \nFrench: Où est le poulet? \nEnglish:",),
    ]
).toDF("prompt")

display(completion.transform(translate_df))
```

Prompt for question answering

Here, we prompt the GPT-3 model for general-knowledge question answering:

Python

```
qa_df = spark.createDataFrame(
    [
        (
            "Q: Where is the Grand Canyon?\nA: The Grand Canyon is in
Arizona.\n\nQ: What is the weight of the Burj Khalifa in kilograms?\nA:",
        )
    ]
).toDF("prompt")

display(completion.transform(qa_df))
```

Azure OpenAI encryption of data at rest

Article • 01/11/2023 • 6 minutes to read

Azure OpenAI automatically encrypts your data when it's persisted to the cloud. The encryption protects your data and helps you meet your organizational security and compliance commitments. This article covers how Azure OpenAI handles encryption of data at rest, specifically training data and fine-tuned models. For information on how data provided by you to the service is processed, used, and stored, consult the data, privacy, and security article.

About Cognitive Services encryption

Azure OpenAI is part of Azure Cognitive Services. Cognitive Services data is encrypted and decrypted using FIPS 140-2 compliant 256-bit AES encryption. Encryption and decryption are transparent, meaning encryption and access are managed for you. Your data is secure by default and you don't need to modify your code or applications to take advantage of encryption.

About encryption key management

By default, your subscription uses Microsoft-managed encryption keys. There's also the option to manage your subscription with your own keys called customer-managed keys (CMK). CMK offers greater flexibility to create, rotate, disable, and revoke access controls. You can also audit the encryption keys used to protect your data.

Customer-managed keys with Azure Key Vault

Customer-managed keys (CMK), also known as Bring your own key (BYOK), offer greater flexibility to create, rotate, disable, and revoke access controls. You can also audit the encryption keys used to protect your data.

You must use Azure Key Vault to store your customer-managed keys. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. The Cognitive Services resource and the key vault must be in the same region and in the same Azure Active Directory (Azure AD) tenant, but they can be in different subscriptions. For more information about Azure Key Vault, see What is Azure Key Vault?.

To request the ability to use customer-managed keys, fill out and submit the Cognitive Services Customer-Managed Key Request Form. It will take approximately 3-5 business days to hear back on the status of your request.

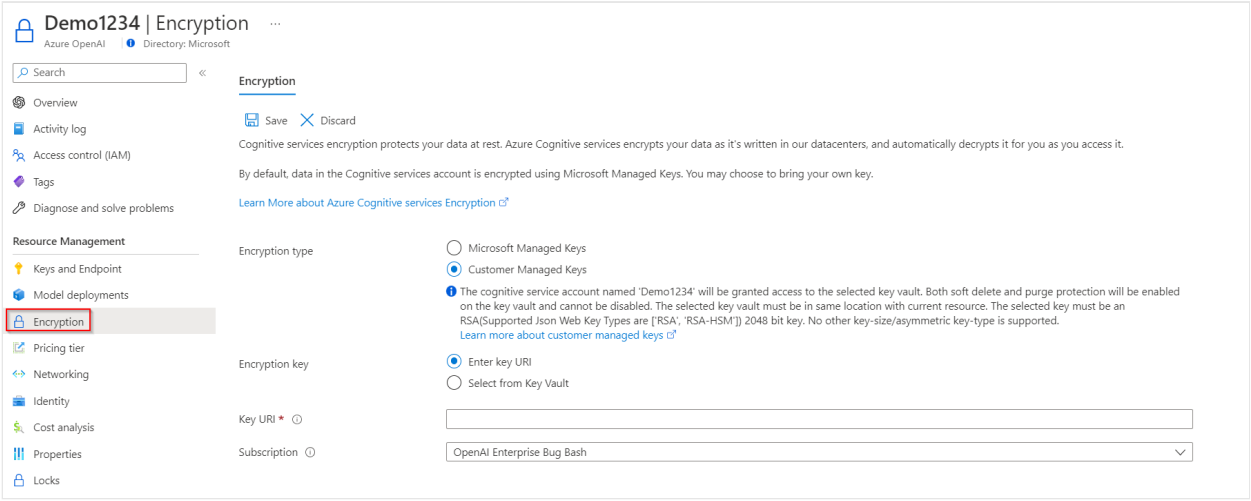
To enable customer-managed keys, you must also enable both the **Soft Delete** and **Do Not Purge** properties on the key vault.

Only RSA keys of size 2048 are supported with Cognitive Services encryption. For more information about keys, see **Key Vault keys** in About Azure Key Vault keys, secrets and certificates.

Enable customer-managed keys for your resource

To enable customer-managed keys in the Azure portal, follow these steps:

1. Go to your Cognitive Services resource.
2. On the left, select **Encryption**.
3. Under **Encryption type**, select **Customer Managed Keys**, as shown in the following screenshot.



Specify a key

After you enable customer-managed keys, you can specify a key to associate with the Cognitive Services resource.

Specify a key as a URI

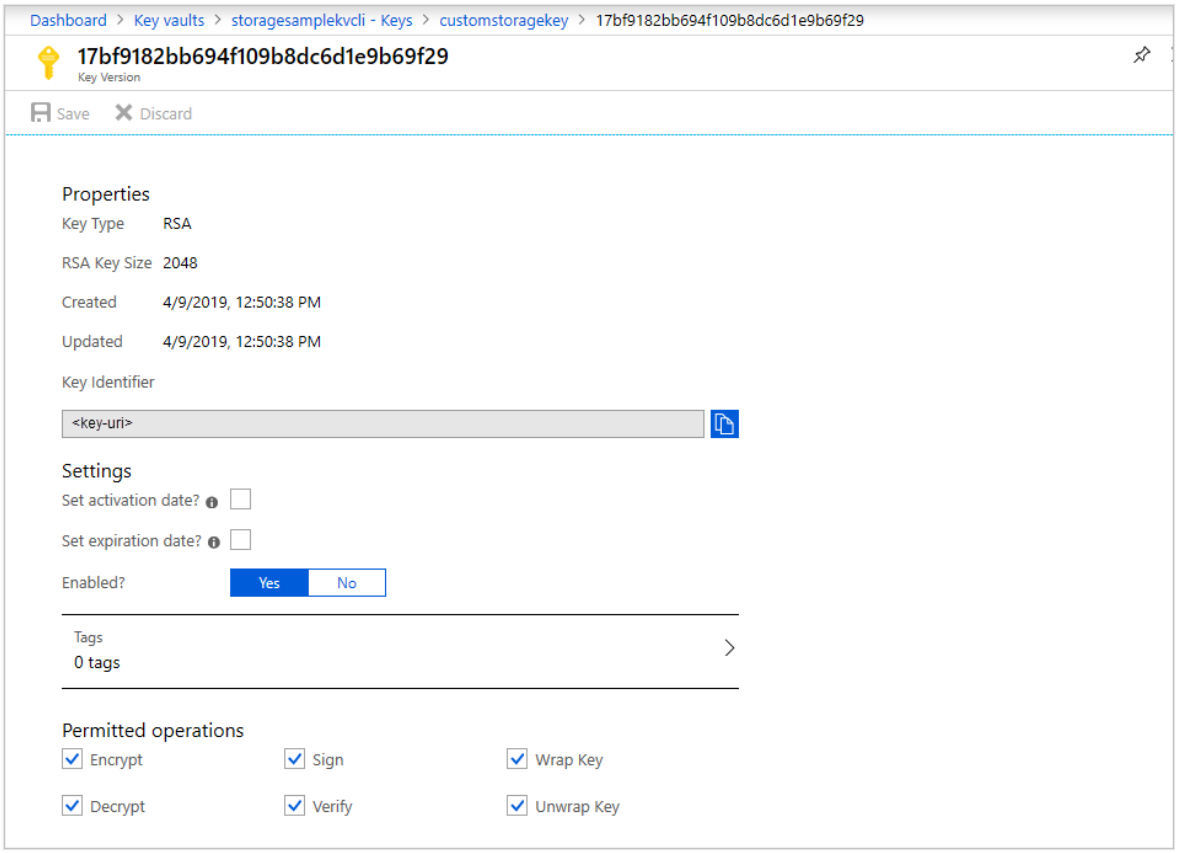
To specify a key as a URI, follow these steps:

1. In the Azure portal, go to your key vault.

2. Under **Settings**, select **Keys**.

3. Select the desired key, and then select the key to view its versions. Select a key version to view the settings for that version.

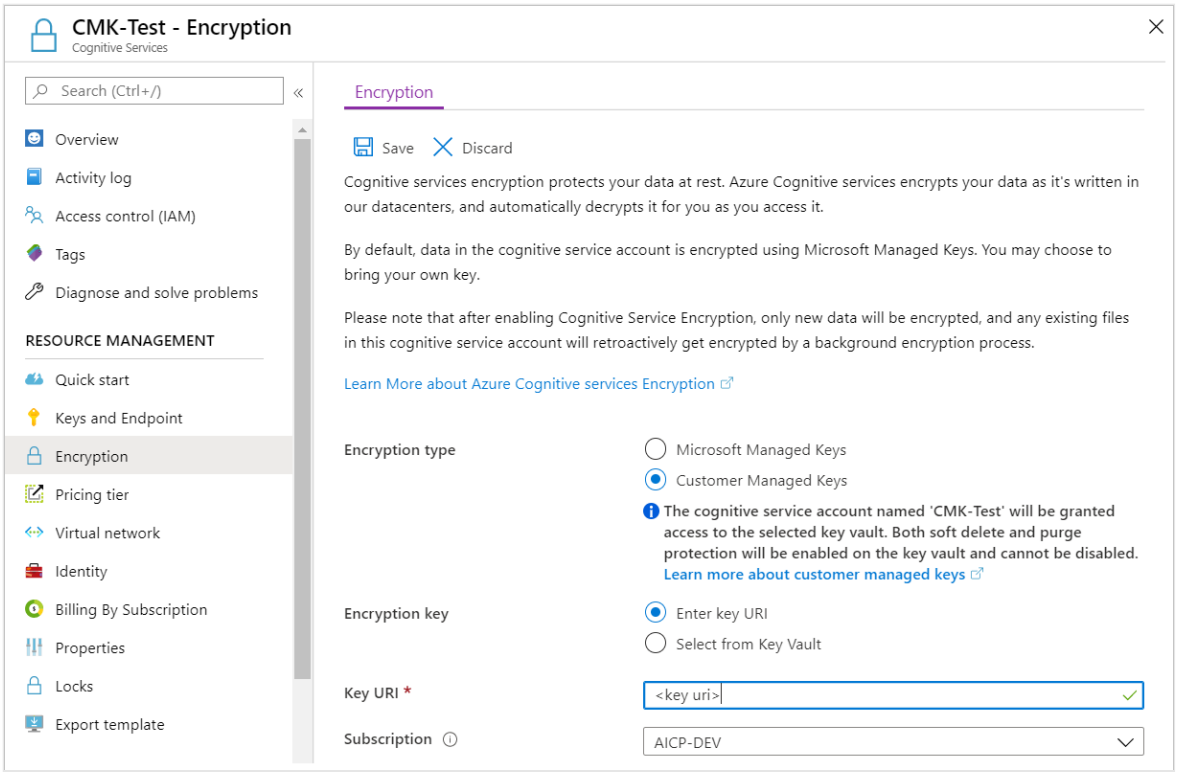
4. Copy the **Key Identifier** value, which provides the URI.



5. Go back to your Cognitive Services resource, and then select **Encryption**.

6. Under **Encryption key**, select **Enter key URI**.

7. Paste the URI that you copied into the **Key URI** box.



8. Under **Subscription**, select the subscription that contains the key vault.

9. Save your changes.

Specify a key from a key vault

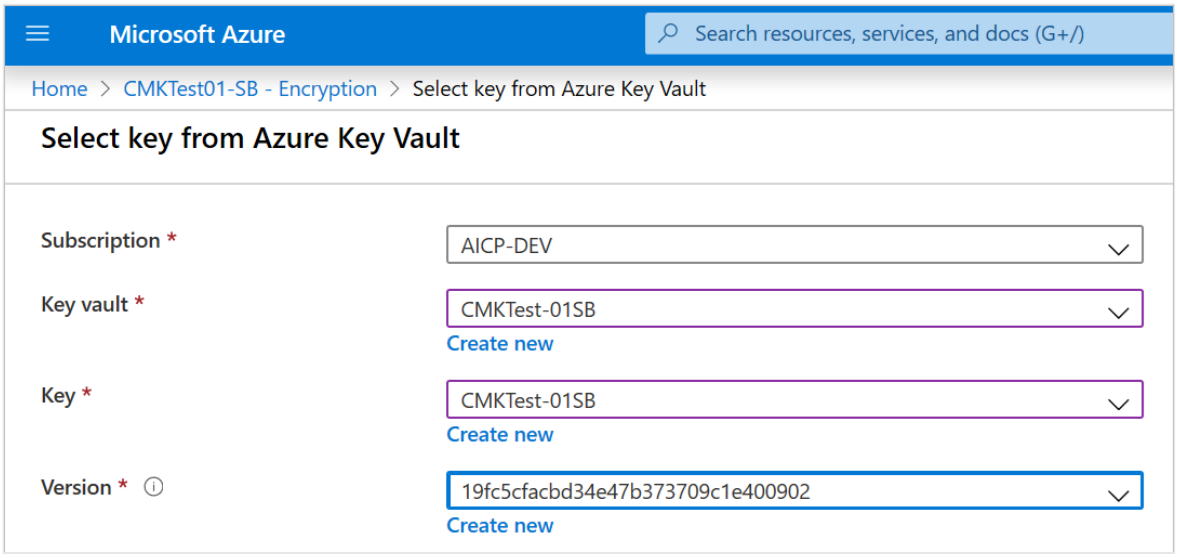
To specify a key from a key vault, first make sure that you have a key vault that contains a key. Then follow these steps:

1. Go to your Cognitive Services resource, and then select **Encryption**.

2. Under **Encryption key**, select **Select from Key Vault**.

3. Select the key vault that contains the key that you want to use.

4. Select the key that you want to use.



5. Save your changes.

Update the key version

When you create a new version of a key, update the Cognitive Services resource to use the new version. Follow these steps:

1. Go to your Cognitive Services resource, and then select **Encryption**.
2. Enter the URI for the new key version. Alternately, you can select the key vault and then select the key again to update the version.
3. Save your changes.

Use a different key

To change the key that you use for encryption, follow these steps:

1. Go to your Cognitive Services resource, and then select **Encryption**.
2. Enter the URI for the new key. Alternately, you can select the key vault and then select a new key.
3. Save your changes.

Rotate customer-managed keys

You can rotate a customer-managed key in Key Vault according to your compliance policies. When the key is rotated, you must update the Cognitive Services resource to use the new key URI. To learn how to update the resource to use a new version of the key in the Azure portal, see Update the key version.

Rotating the key doesn't trigger re-encryption of data in the resource. No further action is required from the user.

Revoke a customer-managed key

You can revoke a customer-managed encryption key by changing the access policy, by changing the permissions on the key vault, or by deleting the key.

To change the access policy of the managed identity that your registry uses, run the az-keyvault-delete-policy command:

Azure CLI

```
az keyvault delete-policy \
  --resource-group <resource-group-name> \
  --name <key-vault-name> \
  --key_id <key-vault-key-id>
```

To delete the individual versions of a key, run the az-keyvault-key-delete command. This operation requires the *keys/delete* permission.

Azure CLI

```
az keyvault key delete  \
  --name <key-vault-name> \
  --object-id $identityPrincipalID \
```

> **Important**
>
> Revoking access to an active customer-managed key while CMK is still enabled will prevent downloading of training data and results files, fine-tuning new models, and deploying fine-tuned models. However, previously deployed fine-tuned models will continue to operate and serve traffic until those deployments are deleted.

Delete training, validation, and training results data

The Files API allows customers to upload their training data for the purpose of fine-tuning a model. This data is stored in Azure Storage, within the same region as the resource and logically isolated with their Azure subscription and API Credentials. Uploaded files can be deleted by the user via the DELETE API operation.

Delete fine-tuned models and deployments

The Fine-tunes API allows customers to create their own fine-tuned version of the OpenAI models based on the training data that you've uploaded to the service via the Files APIs. The trained fine-tuned models are stored in Azure Storage in the same region, encrypted at rest and logically isolated with their Azure subscription and API credentials. Fine-tuned models and deployments can be deleted by the user by calling the DELETE API operation.

Disable customer-managed keys

When you disable customer-managed keys, your Cognitive Services resource is then encrypted with Microsoft-managed keys. To disable customer-managed keys, follow these steps:

1. Go to your Cognitive Services resource, and then select **Encryption**.
2. Select **Microsoft Managed Keys** > **Save**.

When you previously enabled customer managed keys this also enabled a system assigned managed identity, a feature of Azure AD. Once the system assigned managed identity is enabled, this resource will be registered with Azure Active Directory. After being registered, the managed identity will be given access to the Key Vault selected during customer managed key setup. You can learn more about Managed Identities.

> **ⓘ Important**
>
> If you disable system assigned managed identities, access to the key vault will be removed and any data encrypted with the customer keys will no longer be accessible. Any features depended on this data will stop working.

> **ⓘ Important**
>
> Managed identities do not currently support cross-directory scenarios. When you configure customer-managed keys in the Azure portal, a managed identity is automatically assigned under the covers. If you subsequently move the subscription, resource group, or resource from one Azure AD directory to another, the managed identity associated with the resource is not transferred to the new tenant, so customer-managed keys may no longer work. For more information, see **Transferring a subscription between Azure AD directories** in **FAQs and known issues with managed identities for Azure resources**.

Next steps

- Language service Customer-Managed Key Request Form
- Learn more about Azure Key Vault

Additional resources

Documentation

Azure OpenAI models - Azure OpenAI

Learn about the different models that are available in Azure OpenAI.

What's new in Azure OpenAI? - Azure Cognitive Services

Learn about the latest news and features updates for Azure OpenAI

Use cases for Azure OpenAI - Azure Cognitive Services

Transparency Note for Azure OpenAI Service

What is Azure OpenAI? - Azure Cognitive Services

Apply advanced language models to variety of use cases with the Azure OpenAI service

Azure OpenAI Service quotas and limits - Azure Cognitive Services

Quick reference, detailed description, and best practices on the quotas and limits for the OpenAI service in Azure Cognitive Services.

Azure OpenAI embeddings tutorial - Azure OpenAI

Learn how to use the Azure OpenAI embeddings API for document search with the BillSum dataset

Quickstart - Deploy a model and generate text using Azure OpenAI - Azure OpenAI

Walkthrough on how to get started with Azure OpenAI and make your first completions call.

How-to - Create a resource and deploy a model using Azure OpenAI - Azure OpenAI

Walkthrough on how to get started with Azure OpenAI and make your first resource and deploy your first model.

Show 5 more

Business Continuity and Disaster Recovery (BCDR) considerations with Azure OpenAI

Article • 06/30/2022 • 2 minutes to read

The Azure OpenAI service is available in two regions. Since subscription keys are region bound, when a customer acquires a key, they select the region in which their deployments will reside and from then on, all operations stay associated with that Azure server region.

It's rare, but not impossible, to encounter a network issue that hits an entire region. If your service needs to always be available, then you should design it to either fail-over into another region or split the workload between two or more regions. Both approaches require at least two OpenAI resources in different regions. This article provides general recommendations for how to implement Business Continuity and Disaster Recovery (BCDR) for your Azure OpenAI applications.

Best practices

Today customers will call the endpoint provided during deployment for both deployments and inference. These operations are stateless, so no data is lost in the case that a region becomes unavailable.

If a region is non-operational customers must take steps to ensure service continuity.

Business continuity

The following set of instructions applies both customers using default endpoints and those using custom endpoints.

Default endpoint recovery

If you're using a default endpoint, you should configure your client code to monitor errors, and if the errors persist, be prepared to redirect to another region of your choice where you have an Azure OpenAI subscription.

Follow these steps to configure your client to monitor errors:

1. Use this page to identify the list of available regions for the OpenAI service.

2. Select a primary and one secondary/backup regions from the list.
3. Create OpenAI Service resources for each region selected
4. For the primary region and any backup regions your code will need to know:

    a. Base URI for the resource

    b. Regional access key or Azure Active Directory access

5. Configure your code so that you monitor connectivity errors (typically connection timeouts and service unavailability errors).

    a. Given that networks yield transient errors, for single connectivity issue occurrences, the suggestion is to retry.

    b. For persistence redirect traffic to the backup resource in the region you've created.

BCDR requires custom code

The recovery from regional failures for this usage type can be performed instantaneously and at a very low cost. This does however, require custom development of this functionality on the client side of your application.

Tutorial: Explore Azure OpenAI embeddings and document search

Article • 01/11/2023 • 8 minutes to read

This tutorial will walk you through using the Azure OpenAI embeddings API to perform **document search** where you'll query a knowledge base to find the most relevant document.

In this tutorial, you learn how to:

- ✓ Install Azure OpenAI and other dependent Python libraries.
- ✓ Download the BillSum dataset and prepare it for analysis.
- ✓ Create environment variables for your resources endpoint and API key.
- ✓ Use the **text-search-curie-doc-001** and **text-search-curie-query-001** models.
- ✓ Use cosine similarity to rank search results.

Prerequisites

- An Azure subscription - Create one for free
- Access granted to the Azure OpenAI service in the desired Azure subscription Currently, access to this service is granted only by application. You can apply for access to the Azure OpenAI service by completing the form at https://aka.ms/oai/access. Open an issue on this repo to contact us if you have an issue.
- Python 3.7.1 or later version
- The following Python libraries: openai, num2words, matplotlib, plotly, scipy, scikit-learn, transformers.
- An Azure OpenAI Service resource with **text-search-curie-doc-001** and **text-search-curie-query-001** models deployed. These models are currently only available in certain regions. If you don't have a resource the process is documented in our resource deployment guide.

> ⓘ **Note**
>
> If you have never worked with the Hugging Face transformers library it has its own specific **prerequisites** that are required before you can successfully run `pip install transformers`.

Set up

Python libraries

If you haven't already, you need to install the following libraries:

```cmd
pip install openai num2words matplotlib plotly scipy scikit-learn
transformers
```

Alternatively, you can use our requirements.txt file.

Download the BillSum dataset

BillSum is a dataset of United States Congressional and California state bills. For illustration purposes, we'll look only at the US bills. The corpus consists of bills from the 103rd-115th (1993-2018) sessions of Congress. The data was split into 18,949 train bills and 3,269 test bills. The BillSum corpus focuses on mid-length legislation from 5,000 to 20,000 characters in length. More information on the project and the original academic paper where this dataset is derived from can be found on the BillSum project's GitHub repository.

This tutorial uses the `bill_sum_data.csv` file that can be downloaded from our GitHub sample data.

You can also download the sample data by running the following command on your local machine:

```cmd
curl "https://raw.githubusercontent.com/Azure-Samples/Azure-OpenAI-Docs-
Samples/main/Samples/Tutorials/Embeddings/data/bill_sum_data.csv" --output
bill_sum_data.csv
```

Retrieve key and endpoint

To successfully make a call against the Azure OpenAI service, you'll need an **endpoint** and a **key**.

Variable name	Value

Variable name	Value
`ENDPOINT`	This value can be found in the **Keys & Endpoint** section when examining your resource from the Azure portal. Alternatively, you can find the value in the **Azure OpenAI Studio** > **Playground** > **Code View**. An example endpoint is: `https://docs-test-001.openai.azure.com/`.
`API-KEY`	This value can be found in the **Keys & Endpoint** section when examining your resource from the Azure portal. You can use either `KEY1` or `KEY2`.

Go to your resource in the Azure portal. The **Endpoint and Keys** can be found in the **Resource Management** section. Copy your endpoint and access key as you'll need both for authenticating your API calls. You can use either `KEY1` or `KEY2`. Always having two keys allows you to securely rotate and regenerate keys without causing a service disruption.

Create and assign persistent environment variables for your key and endpoint.

Environment variables

Command Line

CMD

```
setx AZURE_OPENAI_API_KEY "REPLACE_WITH_YOUR_KEY_VALUE_HERE"
```

CMD

```
setx AZURE_OPENAI_ENDPOINT "REPLACE_WITH_YOUR_ENDPOINT_HERE"
```

After setting the environment variables you may need to close and reopen Jupyter notebooks or whatever IDE you are using in order for the environment variables to be accessible.

Run the following code in your preferred Python IDE:

If you wish to view the Jupyter notebook that corresponds to this tutorial you can download the tutorial from our samples repo.

Import libraries and list models

Python

```python
import openai
import re
import requests
import sys
from num2words import num2words
import os
import pandas as pd
import numpy as np
from openai.embeddings_utils import get_embedding, cosine_similarity
from transformers import GPT2TokenizerFast

API_KEY = os.getenv("AZURE_OPENAI_API_KEY")
RESOURCE_ENDPOINT = os.getenv("AZURE_OPENAI_ENDPOINT")

openai.api_type = "azure"
openai.api_key = API_KEY
openai.api_base = RESOURCE_ENDPOINT
openai.api_version = "2022-12-01"

url = openai.api_base + "/openai/deployments?api-version=2022-12-01"

r = requests.get(url, headers={"api-key": API_KEY})

print(r.text)
```

Output:

cmd


```
{
  "data": [
    {
      "scale_settings": {
        "scale_type": "standard"
      },
      "model": "text-davinci-002",
      "owner": "organization-owner",
      "id": "text-davinci-002",
      "status": "succeeded",
      "created_at": 1657572678,
      "updated_at": 1657572678,
      "object": "deployment"
    },
    {
      "scale_settings": {
        "scale_type": "standard"
      },
      "model": "code-cushman-001",
      "owner": "organization-owner",
      "id": "code-cushman-001",
      "status": "succeeded",
```

```
      "created_at": 1657572712,
      "updated_at": 1657572712,
      "object": "deployment"
    },
    {
      "scale_settings": {
        "scale_type": "standard"
      },
      "model": "text-search-curie-doc-001",
      "owner": "organization-owner",
      "id": "text-search-curie-doc-001",
      "status": "succeeded",
      "created_at": 1668620345,
      "updated_at": 1668620345,
      "object": "deployment"
    },
    {
      "scale_settings": {
        "scale_type": "standard"
      },
      "model": "text-search-curie-query-001",
      "owner": "organization-owner",
      "id": "text-search-curie-query-001",
      "status": "succeeded",
      "created_at": 1669048765,
      "updated_at": 1669048765,
      "object": "deployment"
    }
  ],
  "object": "list"
}
```


The output of this command will vary based on the number and type of models you've deployed. In this case, we need to confirm that we have entries for both **text-search-curie-doc-001** and **text-search-curie-query-001**. If you find that you're missing one of these models, you'll need to deploy the models to your resource before proceeding.

Now we need read our csv file and create a pandas DataFrame. After the initial DataFrame is created, we can view the contents of the table by running `df`.

Python

```
df = pd.read_csv("INSERT LOCAL PATH TO BILL_SUM_DATA.CSV") # example: df = pd.read_csv("c:\\test\\bill_sum_data.csv")df
df
```

Output:

	Unnamed: 0	bill_id	text	summary	title	text_len	sum_len
0	0	110_hr37	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	National Science Education Tax Incentive for B...	To amend the Internal Revenue Code of 1986 to ...	8494	321
1	1	112_hr2873	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Small Business Expansion and Hiring Act of 201...	To amend the Internal Revenue Code of 1986 to ...	6522	1424
2	2	109_s2408	SECTION 1. RELEASE OF DOCUMENTS CAPTURED IN IR...	Requires the Director of National Intelligence...	A bill to require the Director of National Int...	6154	463
3	3	108_s1899	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	National Cancer Act of 2003 - Amends the Publi...	A bill to improve data collection and dissemin...	19853	1400
4	4	107_s1531	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Military Call-up Relief Act - Amends the Inter...	A bill to amend the Internal Revenue Code of 1...	6273	278
5	5	107_hr4541	SECTION 1. RELIQUIDATION OF CERTAIN ENTRIES PR...	Requires the Customs Service to reliquidate ce...	To provide for reliquidation of entries premat...	11691	114
6	6	111_s1495	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Service Dogs for Veterans Act of 2009 - Direct...	A bill to require the Secretary of Veterans Af...	5328	379
7	7	111_s3885	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Race to the Top Act of 2010 - Directs the Secr...	A bill to provide incentives for States and lo...	16668	1525
8	8	113_hr1796	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Troop Talent Act of 2013 - Directs the Secreta...	Troop Talent Act of 2013	15352	2151
9	9	103_hr1987	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Taxpayer's Right to View Act of 1993 - Amends ...	Taxpayer's Right to View Act of 1993	5633	894
10	10	103_hr1677	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Full-Service Schools Act - Establishes the Fed...	Full-Service Schools Act	12472	1107
11	11	111_s3149	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Wall Street Compensation Reform Act of 2010 - ...	A bill to amend the Internal Revenue Code of 1...	18226	1297
12	12	110_hr1007	SECTION 1. FINDINGS.\r\n\r\n The Congress f...	Amends the Marine Mammal Protection Act of 197...	To amend the Marine Mammal Protection Act of 1...	5261	276
13	13	113_hr3137	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Freedom and Mobility in Consumer Banking Act -...	Freedom and Mobility in Consumer Banking Act	17690	2044
14	14	115_hr1634	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Education and Training for Health Act of 2017 ...	Education and Training for Health Act of 2017	9037	772
15	15	103_hr1815	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Recreational Hunting Safety and Preservation A...	Recreational Hunting Safety and Preservation A...	13024	475
16	16	113_s1773	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Andrew Prior Act or Andrew's Law - Amends the ...	Andrew's Law	5149	613
17	17	106_hr5585	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Directs the President, in coordination with de...	Energy Independence Act of 2000	8007	810
18	18	114_hr2499	SECTION 1. SHORT TITLE.\r\n This Act may be...	This measure has not been amended since it was...	Veterans Entrepreneurship Act of 2015	7539	1421
19	19	111_hr3141	SECTION 1. SHORT TITLE.\r\n\r\n This Act ma...	Strengthening the Health Care Safety Net Act o...	To amend title XIX of the Social Security Act ...	18429	514

The initial table has more columns than we need we'll create a new smaller DataFrame called `df_bills` which will contain only the columns for `text`, `summary`, and `title`.

Python

```
df_bills = df[['text', 'summary', 'title']]
df_bills
```

Output:

	text	summary	title
0	SECTION 1. SHORT TITLE. This Act may be cited ...	National Science Education Tax Incentive for B...	To amend the Internal Revenue Code of 1986 to ...
1	SECTION 1. SHORT TITLE. This Act may be cited ...	Small Business Expansion and Hiring Act of 201...	To amend the Internal Revenue Code of 1986 to ...
2	SECTION 1. RELEASE OF DOCUMENTS CAPTURED IN IR...	Requires the Director of National Intelligence...	A bill to require the Director of National Int...
3	SECTION 1. SHORT TITLE. This Act may be cited ...	National Cancer Act of 2003 - Amends the Publi...	A bill to improve data collection and dissemin...
4	SECTION 1. SHORT TITLE. This Act may be cited ...	Military Call-up Relief Act - Amends the Inter...	A bill to amend the Internal Revenue Code of 1...
5	SECTION 1. RELIQUIDATION OF CERTAIN ENTRIES PR...	Requires the Customs Service to reliquidate ce...	To provide for reliquidation of entries premat...
6	SECTION 1. SHORT TITLE. This Act may be cited ...	Service Dogs for Veterans Act of 2009 - Direct...	A bill to require the Secretary of Veterans Af...
7	SECTION 1. SHORT TITLE. This Act may be cited ...	Race to the Top Act of 2010 - Directs the Secr...	A bill to provide incentives for States and lo...
8	SECTION 1. SHORT TITLE. This Act may be cited ...	Troop Talent Act of 2013 - Directs the Secreta...	Troop Talent Act of 2013
9	SECTION 1. SHORT TITLE. This Act may be cited ...	Taxpayer's Right to View Act of 1993 - Amends ...	Taxpayer's Right to View Act of 1993
10	SECTION 1. SHORT TITLE. This Act may be cited ...	Full-Service Schools Act - Establishes the Fed...	Full-Service Schools Act
11	SECTION 1. SHORT TITLE. This Act may be cited ...	Wall Street Compensation Reform Act of 2010 - ...	A bill to amend the Internal Revenue Code of 1...
12	SECTION 1. FINDINGS. The Congress finds the fo...	Amends the Marine Mammal Protection Act of 197...	To amend the Marine Mammal Protection Act of 1...
13	SECTION 1. SHORT TITLE. This Act may be cited ...	Freedom and Mobility in Consumer Banking Act -...	Freedom and Mobility in Consumer Banking Act
14	SECTION 1. SHORT TITLE. This Act may be cited ...	Education and Training for Health Act of 2017 ...	Education and Training for Health Act of 2017
15	SECTION 1. SHORT TITLE. This Act may be cited ...	Recreational Hunting Safety and Preservation A...	Recreational Hunting Safety and Preservation A...
16	SECTION 1. SHORT TITLE. This Act may be cited ...	Andrew Prior Act or Andrew's Law - Amends the ...	Andrew's Law
17	SECTION 1. SHORT TITLE. This Act may be cited ...	Directs the President, in coordination with de...	Energy Independence Act of 2000
18	SECTION 1. SHORT TITLE. This Act may be cited ...	This measure has not been amended since it was...	Veterans Entrepreneurship Act of 2015
19	SECTION 1. SHORT TITLE. This Act may be cited ...	Strengthening the Health Care Safety Net Act o...	To amend title XIX of the Social Security Act ...

Next we'll perform some light data cleaning by removing redundant whitespace and cleaning up the punctuation to prepare the data for tokenization.

Python

```
# s is input text
def normalize_text(s, sep_token = " \n "):
    s = re.sub(r'\s+',  ' ', s).strip()
    s = re.sub(r". ,","",s)
    # remove all instances of multiple spaces
    s = s.replace("..",".")
    s = s.replace(". .",".")
    s = s.replace("\n", "")
    s = s.strip()

    return s

df_bills['text'] = df_bills["text"].apply(lambda x : normalize_text(x))
```

> **Note**
>
> If you receive a warning stating *`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` you can safely ignore this message.

Now we need to remove any bills that are too long for the token limit (~2000 tokens).

Python

```
tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")
df_bills['n_tokens'] = df_bills["text"].apply(lambda x:
len(tokenizer.encode(x)))
df_bills = df_bills[df_bills.n_tokens<2000]
len(df_bills)
```

Output:

cmd

```
12
```

> **Note**

You can ignore the message: `Token indices sequence length is longer than the specified maximum sequence length for this model (1480 > 1024). Running this sequence through the model will result in indexing errors. 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.`

We'll once again examine **df_bills**. Note that as expected, now only 12 results are returned though they retain their original index in the first column, and we've now added a column called n_tokens.

Python

```
df_bills
```

Output:

	text	summary	title	n_tokens
0	SECTION 1. SHORT TITLE. This Act may be cited ...	National Science Education Tax Incentive for B...	To amend the Internal Revenue Code of 1986 to ...	1480
1	SECTION 1. SHORT TITLE. This Act may be cited ...	Small Business Expansion and Hiring Act of 201...	To amend the Internal Revenue Code of 1986 to ...	1152
2	SECTION 1. RELEASE OF DOCUMENTS CAPTURED IN IR...	Requires the Director of National Intelligence...	A bill to require the Director of National Int...	930
4	SECTION 1. SHORT TITLE. This Act may be cited ...	Military Call-up Relief Act - Amends the Inter...	A bill to amend the Internal Revenue Code of 1...	1048
5	SECTION 1. RELIQUIDATION OF CERTAIN ENTRIES PR...	Requires the Customs Service to reliquidate ce...	To provide for reliquidation of entries premat...	1846
6	SECTION 1. SHORT TITLE. This Act may be cited ...	Service Dogs for Veterans Act of 2009 - Direct...	A bill to require the Secretary of Veterans Af...	872
9	SECTION 1. SHORT TITLE. This Act may be cited ...	Taxpayer's Right to View Act of 1993 - Amends ...	Taxpayer's Right to View Act of 1993	946
12	SECTION 1. FINDINGS. The Congress finds the fo...	Amends the Marine Mammal Protection Act of 197...	To amend the Marine Mammal Protection Act of 1...	1223
14	SECTION 1. SHORT TITLE. This Act may be cited ...	Education and Training for Health Act of 2017 ...	Education and Training for Health Act of 2017	1596
16	SECTION 1. SHORT TITLE. This Act may be cited ...	Andrew Prior Act or Andrew's Law - Amends the ...	Andrew's Law	608
17	SECTION 1. SHORT TITLE. This Act may be cited ...	Directs the President, in coordination with de...	Energy Independence Act of 2000	1341
18	SECTION 1. SHORT TITLE. This Act may be cited ...	This measure has not been amended since it was...	Veterans Entrepreneurship Act of 2015	1404

To understand the n_tokens column a little more as well how the text is tokenized, it can be helpful to run the following code:

Python

```
understand_tokenization = tokenizer.tokenize(df_bills.text[0])
understand_tokenization
```

For our docs we're intentionally truncating the output, but running this command in your environment will return the full text from first index tokenized into chunks.

Output:

cmd

```
['S',
 'ECTION',
 'Ġ1',
 '.',
 'ĠSH',
 'ORT',
 'ĠTIT',
 'LE',
 '.',
 'ĠThis',
 'ĠAct',
 'Ġmay',
 'Ġbe',
 'Ġcited',
 'Ġas',
 'Ġthe',
 'Ġ``',
 'National',
 'ĠScience',
 'ĠEducation',
 'ĠTax',
 'ĠIn',
 'cent',
 'ive',
 'Ġfor',
 'ĠBusiness',
...
```

If you then check the length of the `understand_tokenization` variable, you'll find it matches the first number in the n_tokens column.

Python

```
len(understand_tokenization)
```

Output:

cmd

```
1480
```

Now that we understand more about how tokenization works we can move on to embedding. Before searching, we'll embed the text documents and save the corresponding embedding. We embed each chunk using a **doc model**, in this case `text-search-curie-doc-001`. These embeddings can be stored locally or in an Azure DB. As a result, each tech document has its corresponding embedding vector in the new curie search column on the right side of the DataFrame.

Python

```
df_bills['curie_search'] = df_bills["text"].apply(lambda x :
get_embedding(x, engine = 'text-search-curie-doc-001'))
```

Python

```
df_bills
```

Output:

	text	summary	title	n_tokens	curie_search
0	SECTION 1. SHORT TITLE. This Act may be cited ...	National Science Education Tax Incentive for B...	To amend the Internal Revenue Code of 1986 to ...	1480	[-0.019770914688706398, 0.011169900186359882, ...
1	SECTION 1. SHORT TITLE. This Act may be cited ...	Small Business Expansion and Hiring Act of 201...	To amend the Internal Revenue Code of 1986 to ...	1152	[-0.007850012741982937, 0.01001765951514244, 0...
2	SECTION 1. RELEASE OF DOCUMENTS CAPTURED IN IR...	Requires the Director of National Intelligence...	A bill to require the Director of National Int...	930	[0.00012103027984267101, 0.011845593340694904,...
4	SECTION 1. SHORT TITLE. This Act may be cited ...	Military Call-up Relief Act - Amends the Inter...	A bill to amend the Internal Revenue Code of 1...	1048	[-0.005481021944433451, 0.00856819562613964, -...
5	SECTION 1. RELIQUIDATION OF CERTAIN ENTRIES PR...	Requires the Customs Service to reliquidate ce...	To provide for reliquidation of entries premat...	1846	[-0.008310390636324883, -0.004660653416067362,...
6	SECTION 1. SHORT TITLE. This Act may be cited ...	Service Dogs for Veterans Act of 2009 - Direct...	A bill to require the Secretary of Veterans Af...	872	[-0.017687108367681503, 0.011164870113134384, ...
9	SECTION 1. SHORT TITLE. This Act may be cited ...	Taxpayer's Right to View Act of 1993 - Amends ...	Taxpayer's Right to View Act of 1993	946	[0.0021867561154067516, -0.00421984819695353 5,...
12	SECTION 1. FINDINGS. The Congress finds the fo...	Amends the Marine Mammal Protection Act of 197...	To amend the Marine Mammal Protection Act of 1...	1223	[-0.015813011676073074, 0.009919906966388226, ...
14	SECTION 1. SHORT TITLE. This Act may be cited ...	Education and Training for Health Act of 2017 ...	Education and Training for Health Act of 2017	1596	[-0.0150684155523777, 0.005073960404843092, 0....
16	SECTION 1. SHORT TITLE. This Act may be cited ...	Andrew Prior Act or Andrew's Law - Amends the ...	Andrew's Law	608	[-0.011593054980039597, 0.022752899676561356, ...
17	SECTION 1. SHORT TITLE. This Act may be cited ...	Directs the President, in coordination with de...	Energy Independence Act of 2000	1341	[-0.008348068222403526, 0.002724383957684404, 0...
18	SECTION 1. SHORT TITLE. This Act may be cited ...	This measure has not been amended since it was...	Veterans Entrepreneurship Act of 2015	1404	[-0.020315825939178467, 0.0011716989101842046,...

At the time of search (live compute), we'll embed the search query using the corresponding **query model** (`text-search-query-001`). Next find the closest embedding in the database, ranked by cosine similarity.

In our example, the user provides the query "can I get information on cable company tax revenue". The query is passed through a function that embeds the query with the corresponding **query model** and finds the embedding closest to it from the previously embedded documents in the previous step.

Python

```
# search through the reviews for a specific product
def search_docs(df, user_query, top_n=3, to_print=True):
    embedding = get_embedding(
        user_query,
        engine="text-search-curie-query-001"
    )
    df["similarities"] = df.curie_search.apply(lambda x:
```

```
cosine_similarity(x, embedding))

    res = (
        df.sort_values("similarities", ascending=False)
        .head(top_n)
    )
    if to_print:
        display(res)
    return res


res = search_docs(df_bills, "can i get information on cable company tax
revenue", top_n=4)
```

Output:

	text	summary	title	n_tokens	curie_search	similarities
9	SECTION 1. SHORT TITLE. This Act may be cited ...	Taxpayer's Right to View Act of 1993 - Amends ...	Taxpayer's Right to View Act of 1993	946	[0.0021867561154067516, -0.004219848196953535,...	0.363270
0	SECTION 1. SHORT TITLE. This Act may be cited ...	National Science Education Tax Incentive for B...	To amend the Internal Revenue Code of 1986 to ...	1480	[-0.019770914688706398, 0.011169900186359882, ...	0.314105
1	SECTION 1. SHORT TITLE. This Act may be cited ...	Small Business Expansion and Hiring Act of 201...	To amend the Internal Revenue Code of 1986 to ...	1152	[-0.007850012741982937, 0.01001765951514244, 0...	0.297908
18	SECTION 1. SHORT TITLE. This Act may be cited ...	This measure has not been amended since it was...	Veterans Entrepreneurship Act of 2015	1404	[-0.020315825939178467, 0.0011716989101842046,...	0.295586

Finally, we'll show the top result from document search based on user query against the entire knowledge base. This returns the top result of the "Taxpayer's Right to View Act of 1993". This document has a cosine similarity score of 0.36 between the query and the document:

Python

```
res["summary"][9]
```

Output:

cmd

```
"Taxpayer's Right to View Act of 1993 - Amends the Communications Act of
1934 to prohibit a cable operator from assessing separate charges for any
video programming of a sporting, theatrical, or other entertainment event if
that event is performed at a facility constructed, renovated, or maintained
with tax revenues or by an organization that receives public financial
support. Authorizes the Federal Communications Commission and local
franchising authorities to make determinations concerning the applicability
of such prohibition. Sets forth conditions under which a facility is
considered to have been constructed, maintained, or renovated with tax
revenues. Considers events performed by nonprofit or public organizations
that receive tax subsidies to be subject to this Act if the event is
sponsored by, or includes the participation of a team that is part of, a tax
exempt organization."
```

Using this approach, you can use embeddings as a search mechanism across documents in a knowledge base. The user can then take the top search result and use it for their downstream task, which prompted their initial query.

Video

There is video walkthrough of this tutorial including the pre-requisite steps which can viewed on this community YouTube post.

Clean up resources

If you created an OpenAI resource solely for completing this tutorial and want to clean up and remove an OpenAI resource, you'll need to delete your deployed models, and then delete the resource or associated resource group if it's dedicated to your test resource. Deleting the resource group also deletes any other resources associated with it.

- Portal
- Azure CLI

Next steps

Learn more about Azure OpenAI's models:

Next steps button

Use cases for Azure OpenAI Service

Article • 01/31/2023 • 20 minutes to read

What is a Transparency Note?

An AI system includes not only the technology, but also the people who will use it, the people who will be affected by it, and the environment in which it is deployed. Creating a system that is fit for its intended purpose requires an understanding of how the technology works, what its capabilities and limitations are, and how to achieve the best performance. Microsoft's Transparency Notes are intended to help you understand how our AI technology works, the choices system owners can make that influence system performance and behavior, and the importance of thinking about the whole system, including the technology, the people, and the environment. You can use Transparency Notes when developing or deploying your own system, or share them with the people who will use or be affected by your system.

Microsoft's Transparency Notes are part of a broader effort at Microsoft to put our AI Principles into practice. To find out more, see the Microsoft's AI principles.

The basics of Azure OpenAI

Introduction

Azure OpenAI provides customers with a fully managed AI service that lets developers and data scientists apply OpenAI's powerful language models including their GPT-3 and Codex series. GPT-3 models analyze and generate natural language, while Codex models analyze and generate code and plain text code commentary. These models use an autoregressive architecture meaning they use data from prior observations to predict the most probable next word. This process is then repeated by appending the newly generated content to the original text to produce the complete generated response. Because the response is conditioned on the input text, these models can be applied to a variety of tasks simply by changing the input text.

The GPT-3 series of models are pretrained on a wide body of publicly available free text data. This data is sourced from a combination of web crawling (specifically, a filtered version of Common Crawl, which includes a broad range of text from the internet and comprises sixty percent of the weighted pre-training dataset) and higher-quality datasets, including an expanded version of the WebText dataset, two internet-based books corpora and English-language Wikipedia. The training data for Codex contains

both natural language and billions of lines of public code from GitHub. Given the wide breadth of knowledge and data that the models are trained on and their ability to generate dynamic content, special care must be taken to ensure responsible use in applications.

Learn more about the training and modeling techniques in OpenAI's GPT-3 and Codex research papers. The guidance below is also drawn from OpenAI's safety best practices.

Key terms

Term	Definition

Prompt	The text you send to the service in the API call. This text is then input into the model. For example, one might input the following prompt:
	
	`Convert the questions to a command:`
	`Q: Ask Constance if we need some bread`
	`A: send-msg \`find constance\` Do we need some bread?`
	`Q: Send a message to Greg to figure out if things are ready for Wednesday.`
	`A:`

Completion or Generation	The text Azure OpenAI outputs in response. For example, the service may respond with the following answer to the above prompt: `send-msg \`find greg\` figure out if things are ready for Wednesday.`
Token	Azure OpenAI processes text by breaking it down into tokens. Tokens can be words or just chunks of characters. For example, the word “hamburger” gets broken up into the tokens “ham”, “bur” and “ger”, while a short and common word like “pear” is a single token. Many tokens start with a whitespace, for example “ hello” and “bye”.

Capabilities

System behavior

The Azure OpenAI Service models use natural language instructions and examples in the prompt to identify the task. The model then completes the task by predicting the most probable next text. This technique is known as "in-context" learning. These models are not re-trained during this step but instead give predictions based on the context you include in the prompt.

There are three main approaches for in-context learning. These approaches vary based on the amount of task-specific data that is given to the model:

Few-shot: In this case, a user includes several examples in the prompt that demonstrate the expected answer format and content. The following example shows a few-shot prompt providing multiple examples:

```
    Convert the questions to a command:
    Q: Ask Constance if we need some bread
    A: send-msg `find constance` Do we need some bread?
    Q: Send a message to Greg to figure out if things are ready for
Wednesday.
    A: send-msg `find greg` Is everything ready for Wednesday?
    Q: Ask Ilya if we're still having our meeting this evening
    A: send-msg `find ilya` Are we still having a meeting this evening?
    Q: Contact the ski store and figure out if I can get my skis fixed
before I leave on Thursday
    A: send-msg `find ski store` Would it be possible to get my skis fixed
before I leave on Thursday?
    Q: Thank Nicolas for lunch
    A: send-msg `find nicolas` Thank you for lunch!
    Q: Tell Constance that I won't be home before 19:30 tonight — unmovable
meeting.
    A: send-msg `find constance` I won't be home before 19:30 tonight. I
have a meeting I can't move.
    Q: Tell John that I need to book an appointment at 10:30
    A:
```

The number of examples typically ranges from 0 to 100 depending on how many can fit in the maximum input length for a single prompt. Few-shot learning enables a major reduction in the amount of task-specific data required for accurate predictions.

One-shot: This case is the same as the few-shot approach except only one example is provided. The following example shows a one-shot prompt:

```
    Convert the questions to a command:
    Q: Ask Constance if we need some bread
    A: send-msg `find constance` Do we need some bread?
    Q: Send a message to Greg to figure out if things are ready for
Wednesday.
    A:
```

Zero-shot: In this case, no examples are provided to the model and only the task request is provided. The following example shows a zero-shot prompt:

```
Convert the question to a command:
Q: Ask Constance if we need some bread
A:
```

Use cases

Intended uses

Azure OpenAI can be used in multiple scenarios. The system's intended uses include:

- **Natural language to code:** An application translates natural language into code for a language with a limited set of operations that follows a specific syntax to help keep the application output on topic. Examples are generating python code based on text prompts or converting a business question into SQL query for your reporting system.
- **Search:** Use the service to search specific documents you provide and generate a natural language response to a question. For example, a computer hardware company can use the model to find answers to commonly asked customer questions by searching authenticated, reliable technical specification documents.
- **Summarization:** Summarize specific content that is sent to humans to review and make the final decision on usage. For example, a company can generate summaries of their own blog posts, which editors can then modify, or a customer agent can review a generated summary (which should be clearly disclosed as AI generated) of top complaints.
- **Writing assistance:** Content is generated or modified based on prompts. For example, a copywriter enters a couple key points for an email. The application outputs several one-sentence suggestions to consider, and the copywriter decides which to use.
- **Information extraction:** Find and extract specific information. For example, keywords can be extracted from a product catalog to support more efficient searches.
- **Classification:** Classify content based on key attributes such as tone, topic or sentiment.
- **Conversational AI in limited domains:** Generate natural responses to questions about specific topics or in limited conversational domains, such as a customer support chatbot.

Considerations when choosing a use case

We encourage customers to leverage Azure OpenAI in their innovative solutions or applications. However, here are some considerations when choosing a use case:

- **Not suitable for open-ended, unconstrained content generation.** Scenarios where users can generate content on any topic are more likely to produce offensive or harmful text. The same is true of longer generations.
- **Not suitable for scenarios where up-to-date, factually accurate information is crucial** unless you have human reviewers or are using the models to search your own documents and have verified suitability for your scenario. The service does not have information about events that occur after its training date, likely has missing knowledge about some topics, and may not always produce factually accurate information.
- **Avoid scenarios where use or misuse of the system could result in significant physical or psychological injury to an individual.** For example, scenarios that diagnose patients or prescribe medications have the potential to cause significant harm.
- **Avoid scenarios where use or misuse of the system could have a consequential impact on life opportunities or legal status.** Examples include scenarios where the AI system could affect an individual's legal status, legal rights, or their access to credit, education, employment, healthcare, housing, insurance, social welfare benefits, services, opportunities, or the terms on which they are provided.
- **Avoid high stakes scenarios that could lead to harm.** The models hosted by Azure OpenAI service reflect certain societal views, biases and other undesirable content present in the training data or the examples provided in the prompt. As a result, we caution against using the models in high-stakes scenarios where unfair, unreliable, or offensive behavior might be extremely costly or lead to harm.
- **Carefully consider use cases in high stakes domains or industry:** Examples include but are not limited to healthcare, medicine, finance or legal.
- **Carefully consider well-scoped chatbot scenarios.** Limiting the use of the service in chatbots to a narrow domain reduces the risk of generating unintended or undesirable responses.
- **Carefully consider all generative use cases.** Content generation scenarios may be more likely to produce unintended outputs and these scenarios require careful consideration and mitigations.

Limitations

When it comes to large-scale natural language models, there are particular fairness and responsible AI issues to consider. People use language to describe the world and to express their beliefs, assumptions, attitudes, and values. As a result, publicly available text data typically used to train large-scale natural language processing models contains

societal biases relating to race, gender, religion, age, and other groups of people, as well as other undesirable content. These societal biases are reflected in the distributions of words, phrases, and syntactic structures.

Technical limitations, operational factors and ranges

Large-scale natural language models trained with such data can potentially behave in ways that are unfair, unreliable, or offensive, in turn causing harms. There are several different ways in which a large-scale natural language processing model can cause harms. Some of the ways are listed here. We emphasize that these types of harms aren't mutually exclusive. A single model can exhibit more than one type of harm, potentially relating to multiple different groups of people. For example:

- **Allocation**: Language models can be used in ways that lead to unfair allocation of resources or opportunities. For example, automated resume screening systems can withhold employment opportunities from one gender if they're trained on resume data that reflects the existing gender imbalance in a particular industry.
- **Quality of service**: Language models can fail to provide the same quality of service to some people as they do to others. For example, sentence completion systems may not work as well for some dialects or language varieties because of their lack of representation in the training data. The Azure OpenAI models are trained primarily on English text. Languages besides English will experience worse performance. English language varieties with less representation in the training data may experience worse performance.
- **Stereotyping**: Language models can reinforce stereotypes. For example, when translating "He is a nurse" and "She is a doctor" into a genderless language such as Turkish and then back into English, many machine translation systems yield the stereotypical (and incorrect) results of "She is a nurse" and "He is a doctor."
- **Demeaning**: Language models can demean people. For example, an open-ended content generation system with inappropriate mitigations might produce offensive text targeted at a particular group of people.
- **Over- and underrepresentation**: Language models can over- or underrepresent groups of people, or even erase them entirely. For example, toxicity detection systems that rate text containing the word "gay" as toxic might lead to the underrepresentation or even erasure of legitimate text written by or about the LGBTQIA+ community.
- **Inappropriate or offensive content**: Language models can produce other types of inappropriate or offensive content. Examples include hate speech; text that contains profane words or phrases; text that relates to illicit activities; text that relates to contested, controversial, or ideologically polarizing topics; misinformation; text that's manipulative; and text that relates to sensitive or

emotionally charged topics. For example, suggested-reply systems that are restricted to positive replies can suggest inappropriate or insensitive replies for messages about negative events.
- **False information**: Azure OpenAI service doesn't fact check or verify content provided by customers or users. Depending on how you've developed your application, it might promote false information unless you've built in mitigations.

System performance

In many AI systems, performance is often defined in relation to accuracy—that is, how often the AI system offers a correct prediction or output. With large-scale natural language models, two different users might look at the same output and have different opinions of how useful or relevant it is, which means that performance for these systems must be defined more flexibly. Here, we broadly consider performance to mean that the application performs as you and your users expect, including not generating harmful outputs.

Azure OpenAI service can support a wide range of applications like search, classification, and code generation, each with different performance metrics and mitigation strategies. There are several steps you can take to mitigate some of the concerns listed under "Limitations" and to improve performance. Additional important mitigation techniques are outlined in the section Evaluating and integrating Azure OpenAI for your use below.

- **Show and tell when designing prompts.** Make it clear to the model what kind of outputs you expect through instructions, examples, or a combination of the two. If you want the model to rank a list of items in alphabetical order or to classify a paragraph by sentiment, show it that's what you want.
- **Keep your application on-topic.** Carefully structure prompts to reduce the chance of producing undesired content, even if a user tries to use it for this purpose. For instance, you might indicate in your prompt that a chatbot only engages in conversations about mathematics and otherwise responds "I'm sorry. I'm afraid I can't answer that." Adding adjectives like "polite" and examples in your desired tone to your prompt can also help steer outputs. Consider nudging users toward acceptable queries, either by listing such examples upfront or by offering them as suggestions upon receiving an off-topic request. Consider training a classifier to determine whether an input is on- or off-topic.
- **Provide quality data.** If you're trying to build a classifier or get the model to follow a pattern, make sure that there are enough examples. Be sure to proofread your examples — the model is usually smart enough to see through basic spelling mistakes and give you a response, but it also might assume this is intentional and it could affect the response.

- **Measure model quality.** As part of general model quality, consider measuring and improving fairness-related metrics and other metrics related to responsible AI in addition to traditional accuracy measures for your scenario. Consider resources like this checklist when you measure the fairness of the system. These measurements come with limitations, which you should acknowledge and communicate to stakeholders along with evaluation results.
- **Limit the length, structure, rate, and source of inputs and outputs.** Restricting the length or structure of inputs and outputs can increase the likelihood that the application will stay on task and mitigate, at least in part, any potential unfair, unreliable, or offensive behavior. Restricting the source of inputs (for example, limiting inputs to a fixed list of items, a particular domain, or to authenticated users rather than anyone on the internet) or restricting the source of outputs (for example, only surfacing answers from approved, vetted documents rather than the open web) can further mitigate the risk of the harms. Putting usage rate limits in place can also reduce misuse.
- **Implement additional scenario-specific mitigations.** Refer to the mitigations outlined in Evaluating and integrating Azure OpenAI for your use below, including content moderation strategies. They do not represent every mitigation that may be required for your application but point to the general minimum baseline we check for when approving use cases for the Azure OpenAI service.

Evaluation of Azure OpenAI

GPT-3 models were trained on a total of 300 billion tokens and evaluated in zero-shot, one-shot and few-shot settings on a range of different NLP benchmarks and tasks, such as language modeling and completion tasks, and on datasets that involve commonsense reasoning, reading comprehension and question answering. GPT-3 shows strong performance on many NLP benchmarks and tasks, it nearly matches the performance of state-of-the-art fine-tuned systems in some cases, and demonstrates strong qualitative performance at tasks defined on-the-fly. More information and benchmarking statistics can be found in the OpenAI GPT-3 research paper.

Evaluating and integrating Azure OpenAI for your use

Practices for responsible use

The practices below are baseline requirements for production applications.

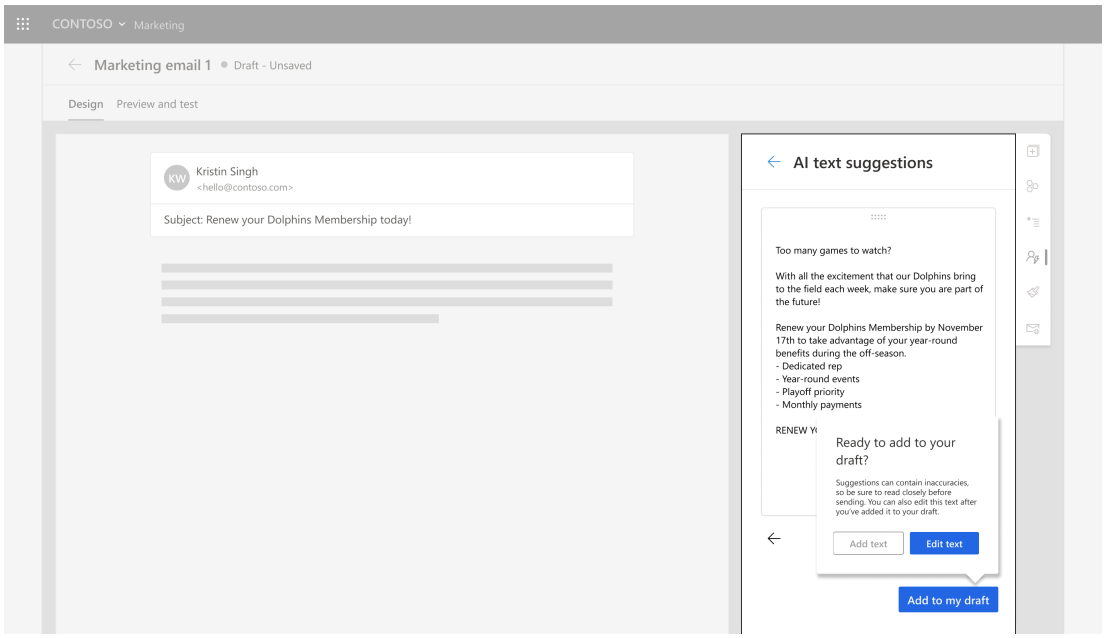
1. **Ensure human oversight.** Ensure appropriate human oversight for your system, including ensuring the people responsible for oversight understand the system's

intended uses; how to effectively interact with the system; how to effectively interpret system behavior; when and how to override, intervene, or interrupt the system; and ways to remain aware of the possible tendency of over-relying on outputs produced by the system ("automation bias"). Especially in high-stakes scenarios, make sure people have a role in decision-making and can intervene in real time to prevent harm. For people to make good decisions about system outputs, they need to (a) understand how the system works, (b) have awareness of the system status and how well it's working in their scenario, and (c) have the opportunity and the tools to bring the system into alignment with their expectations and goals. When using the Azure OpenAI service, human oversight might include some or all of the following:

- **1a Let people edit generated outputs.** *(For detailed guidance and examples, see HAX G9-B: Rich and detailed edits.)* In this example, users are given the option to edit each generated output before using the text. Our research showed that users expected to be able to edit content and even combine parts of different generations.

- **1b. Highlight potential inaccuracies in generated outputs.** *(For detailed guidance and examples, see HAX G2-C: Report system performance information.)* Here, numbers and terms having to do with time periods are underlined. When a user hovers over an underlined item, a tooltip appears that reminds them to check its accuracy. Depending on your scenario, you may want to highlight numbers, days, dates, names, URLs, titles, quotations, addresses, phone numbers, and hashtags. Our research suggests that users find this kind of flagging useful as a reminder to fact-check the content.

- **1c. Remind users that they are accountable for final decisions and/or final content.** In the example below, before a user is allowed to insert generated content into their document, they must go through a pop-up dialog and edit the text or acknowledge that they've reviewed it. Our research suggests that users appreciate having these reminders and believe them to be especially beneficial for beginners.



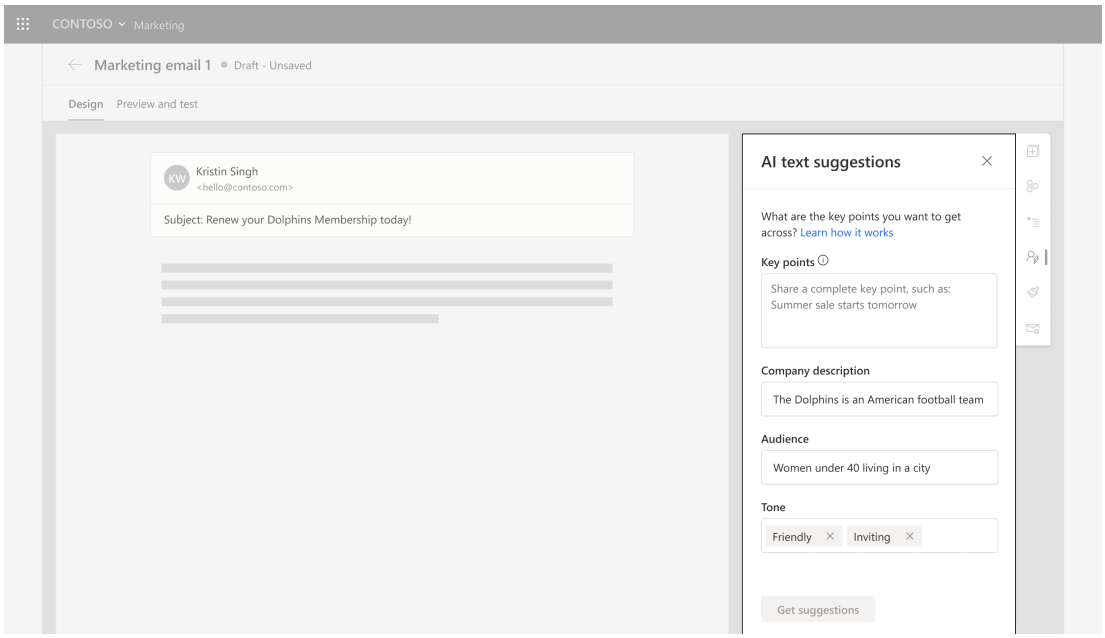
- **1d. Limit how people can automate your product or service.** For example, don't allow automated posting of generated content to external sites (including social media), or automated execution of generated code.

- **1e. Disclose AI's role in generated content.** In some cases, letting content consumers know when published content is partly or fully generated by Azure OpenAI can help them use their own judgment about how to read it. If generated content does not include meaningful human oversight before being shared or published--including opportunities for an expert to understand, review, and edit the content--disclosure may be critical to preventing misinformation.

2. **Implement technical limits on inputs and outputs**

- **2a. Limit the length of inputs and outputs.** Restricting input and output length can reduce the likelihood of producing undesirable content, misuse of the system beyond its intended use cases, or other harmful or unintended scenarios.

- **2b. Structure inputs to limit open-ended responses and to give users more-refined control.** The better the instructions that users give each time they interact with the system, the better the results they'll get. Restrict users from creating custom prompts that let them operate as if interacting directly with the API. You can also limit outputs to be structured in certain formats or patterns.



In the example above, the prompt has been structured to require that users provide limiting details such as audience and tone. When setting prompt fields, consider what information will be easy for users to provide, and run experiments to learn what information changes output quality. Smart defaults can help people get started quickly and can also be used to demonstrate best practices for prompt format, length, and style.

- **2c. Return outputs from validated, reliable source materials,** such as existing support articles, rather than returning non-vetted content from the internet. This can help your application stay on task and mitigate unfair, unreliable, or offensive behavior.

- **2d. Implement blocklists and content moderation.** Keep your application on topic by checking both inputs and outputs for undesired content. The definition of undesired content depends on your scenario and changes over time. It might include hate speech, text that contains profane words or phrases, misinformation, and text that relates to sensitive or emotionally charged topics. Checking inputs can help keep your application on topic, even if a malicious user tries to produce undesired content. Checking API outputs can allow you to detect undesired content produced by the system and take action. You can replace it, report it, ask the user to enter different input, or provide input examples.

- **2e. Put rate limits in place** (in other words, frequency and quantity of API calls) to further reduce misuse.

3. **Authenticate users.** To make misuse more difficult, consider requiring that customers sign in and, if appropriate, link a valid payment method. Consider working only with known, trusted customers in the early stages of development.

Applications that do not authenticate users may require other, stricter mitigations to ensure the application cannot be used beyond its intended purpose.

4. **Test your application thoroughly** to ensure it responds in a way that is fit for the application's purpose. This includes conducting adversarial testing where trusted testers attempt to find system failures, poor performance, or undesirable behaviors. This information helps you to understand risks and consider appropriate mitigations. Communicate the capabilities and limitations to stakeholders.

5. **Establish feedback channels for users and impacted groups.** AI-powered products and features require ongoing monitoring and improvement. Establish channels to collect questions and concerns from users as well as people affected by the system. For example:

    - **5a. Build feedback features into the user experience.** Invite feedback on the usefulness and accuracy of outputs, and give users a separate and clear path to report outputs that are problematic, offensive, biased, or otherwise inappropriate. For detailed guidance and examples, see HAX Guideline 15: Encourage granular feedback.
    - **5b. Publish an easy-to-remember email address for public feedback.**

Scenario-specific practices

1. **If your application powers chatbots or other conversational AI systems,** follow the Microsoft guidelines for responsible development of conversational AI systems.
2. **If you are developing an application in a high stakes domain or industry,** such as healthcare, human resources, education, or the legal field, thoroughly assess how well the application works in your scenario, implement strong human oversight, thoroughly evaluate how well users understand the limitations of the application, and comply with all relevant laws. Consider additional mitigations based on your scenario.

Additional practices

1. **Use Microsoft's Inclusive Design Guidelines** to build inclusive solutions.
2. **Conduct research to test the product and solicit feedback.** Include a diverse group of stakeholders (for example, direct users, consumers of generated results, admins, and so on) to your research structure to seek their feedback at different stages of deployment. Depending on the research goal, you can use various methodologies, such as Community Jury, online experiments, beta testing, and

testing with real users after deployment. Consider including stakeholders from different demographic groups to gather a wider range of feedback.

3. **Conduct a legal review.** Obtain appropriate legal advice to review your solution, particularly if you plan to use it in sensitive or high-risk applications.
4. **Learn more about responsible AI** here .

Learn more about responsible AI

- Microsoft AI principles
- Microsoft responsible AI resources
- Microsoft Azure Learning courses on responsible AI

Learn more about Azure OpenAI

- Limited access to Azure OpenAI Service - Azure Cognitive Services | Microsoft Learn
- Code of Conduct for the Azure OpenAI Service | Microsoft Learn
- Data, privacy, and security for Azure OpenAI Service - Azure Cognitive Services | Microsoft Learn

See also

- Limited access to Azure OpenAI Service
- Code of conduct for Azure OpenAI Service integrations
- Data, privacy, and security for Azure OpenAI Service

Limited access to Azure OpenAI Service

Article • 02/09/2023 • 2 minutes to read

As part of Microsoft's commitment to responsible AI, we are designing and releasing Azure OpenAI Service with the intention of protecting the rights of individuals and society and fostering transparent human-computer interaction. For this reason, we currently limit the access and use of Azure OpenAI, including limiting access to the ability to modify content filters and modify abuse monitoring.

Registration process

Azure OpenAI requires registration and is currently only available to managed customers and partners working with Microsoft account teams. Customers who wish to use Azure OpenAI are required to submit a registration form.

Customers must attest to any and all use cases for which they will use the service. Customers who wish to add additional use cases after initial onboarding must submit the additional use cases using this form. The use of Azure OpenAI is limited to use cases that have been selected in a registration form. Microsoft may require customers to re-verify this information. Read more about example use cases and use cases to avoid here.

Customers who wish to modify content filters and modify abuse monitoring after they have onboarded to the service are subject to additional scenario restrictions and are required to register here.

Access to the Azure OpenAI Service is subject to Microsoft's sole discretion based on eligibility criteria and a vetting process, and customers must acknowledge that they have read and understand the Azure terms of service for Azure OpenAI Service.

Azure OpenAI Service is made available to customers under the terms governing their subscription to Microsoft Azure Services, including the Azure OpenAI section of the Microsoft Product Terms. Please review these terms carefully as they contain important conditions and obligations governing your use of Azure OpenAI Service.

Important links

- Register to use Azure OpenAI
- Add additional use cases (if needed)
- Register to modify content filters and abuse monitoring (if needed)

Help and support

FAQ about Limited Access can be found here. If you need help with Azure OpenAI, find support here. Report abuse of Azure OpenAI here.

Report problematic content to cscraireport@microsoft.com.

See also

- Code of conduct for Azure OpenAI Service integrations
- Transparency note for Azure OpenAI Service
- Characteristics and limitations for Azure OpenAI Service
- Data, privacy, and security for Azure OpenAI Service

Code of conduct for Azure OpenAI Service

Article • 01/30/2023 • 6 minutes to read

The following Code of Conduct defines the requirements that all Azure OpenAI Service implementations must adhere to in good faith. This code of conduct is in addition to the Acceptable Use Policy in the Microsoft Online Services Terms.

Access requirements

Azure OpenAI Service is a Limited Access service that requires registration and is only available to managed customers and partners with Microsoft account teams. Customers who wish to use this feature are required to register through this form both for initial access for experimentation and for approval to move from experimentation to production. To learn more, see Limited Access to Azure OpenAI Service.

Responsible AI mitigation requirements

Integrations with AzureOpen AI Service must:

- Implement meaningful human oversight
- Implement strong technical limits on inputs and outputs to reduce the likelihood of misuse beyond the application's intended purpose
- Test applications thoroughly to find and mitigate undesirable behaviors
- Establish feedback channels
- Implement additional scenario-specific mitigations

To learn more, see the Azure OpenAI transparency note.

Integrations with Azure OpenAI Service must not:

- be used in any way that violates Microsoft's Acceptable Use Policy, including but not limited to any use prohibited by law, regulation, government order, or decree, or any use that violates the rights of others;
- be used in any way that is inconsistent with this code of conduct, including the Limited Access requirements, the Responsible AI mitigation requirements, and the Content requirements;

- exceed the documented use case you provided to Microsoft in connection with your request to use the service;
- interact with individuals under the age of consent in any way that could result in exploitation or manipulation or otherwise prohibited by law or regulation;
- generate or interact with content prohibited in this Code of Conduct;
- be presented alongside or monetize content prohibited in this Code of Conduct;
- make decisions without appropriate human oversight if your application may have a consequential impact on any individual's legal position, financial position, life opportunities, employment opportunities, human rights, or result in physical or psychological injury to an individual;
- infer sensitive information about people without their explicit consent unless if used in a lawful manner by a law enforcement entity, court, or government official subject to judicial oversight in a jurisdiction that maintains a fair and independent judiciary; or
- be used for chatbots that **(i)** are erotic, romantic, or used for companionship purposes, or which are otherwise prohibited by this Code of Conduct; **(ii)** are personas of specific people without their explicit consent; **(iii)** claim to have special wisdom/insight/knowledge, unless very clearly labeled as being for entertainment purposes only; or **(iv)** enable end users to create their own chatbots without oversight.

Content requirements

We prohibit the use of our service for generating content that can inflict harm on individuals or society. Our content policies are intended to improve the safety of our platform.

These content requirements apply to the output of all models developed by OpenAI and hosted in Azure OpenAI, such as GPT-3 and Codex models, and includes content provided as input to the service and content generated as output from the service.

Exploitation and Abuse

Child sexual exploitation and abuse

Azure OpenAI Service prohibits content that describes, features, or promotes child sexual exploitation or abuse, whether or not prohibited by law. This includes sexual content involving a child or that sexualizes a child.

Grooming

Azure OpenAI Service prohibits content that describes or is used for purposes of grooming of children. Grooming is the act of an adult building a relationship with a child for the purposes of exploitation, especially sexual exploitation. This includes communicating with a child for the purpose of sexual exploitation, trafficking, or other forms of exploitation.

Non-consensual intimate content

Azure OpenAI Service prohibits content that describes, features or promotes non-consensual intimate activity.

Sexual solicitation

Azure OpenAI Service prohibits content that describes, features, or promotes, or used for, purposes of solicitation of commercial sexual activity and sexual services. This includes encouragement and coordination of real sexual activity.

Trafficking

Azure OpenAI Service prohibits content describing or used for purposes of human trafficking. This includes the recruitment of individuals, facilitation of transport and payment for, and the promotion of, exploitation of people such as forced labor, domestic servitude, sexual slavery, forced marriages, and forced medical procedures.

Suicide and Self-Injury

Azure OpenAI Service prohibits content that describes, praises, supports, promotes, glorifies, encourages and/or instructs individual(s) on self-injury or to take their life.

Violent Content and Conduct

Graphic violence and gore

Azure OpenAI Service prohibits content that describes, features, or promotes graphic violence or gore.

Terrorism and Violent Extremism

Azure OpenAI Service prohibits content that depicts an act of terrorism, praises, or supports a terrorist organization, terrorist actor, or violent terrorist ideology, encourages

terrorist activities, offers aid to terrorist organizations or terrorist causes, or aids in recruitment to a terrorist organization.

Violent Threats, Incitement, and Glorification of Violence

Azure OpenAI Service prohibits content advocating or promoting violence toward others through violent threats or incitement.

Harmful Content

Hate speech and discrimination

Azure OpenAI Service prohibits content that attacks, denigrates, intimidates, degrades, targets, or excludes individuals or groups on the basis of traits such as actual or perceived race, ethnicity, national origin, gender, gender identity, sexual orientation, religious affiliation, age, disability status, caste, or any other characteristic that is associated with systemic prejudice or marginalization.

Bullying and harassment

Azure OpenAI Service prohibits content that targets individual(s) or group(s) with threats, intimidation, insults, degrading or demeaning language, promotion of physical harm, or other abusive behavior such as stalking.

Deception, disinformation, and inauthentic activity

Azure OpenAI Service prohibits content that is intentionally deceptive and likely to adversely affect the public interest, including deceptive or untrue content relating to health, safety, election integrity, or civic participation). Azure OpenAI Service also prohibits inauthentic interactions, such as fake accounts, automated inauthentic activity, impersonation to gain unauthorized information or privileges, and claims to be from any person, company, government body, or entity without explicit permission to make that representation.

Active malware or exploits

Content that directly supports unlawful active attacks or malware campaigns that cause technical harms, such as delivering malicious executables, organizing denial of service attacks, or managing command and control servers.

Additional content policies

We prohibit the use of our Azure OpenAI Service for scenarios in which the system is likely to generate undesired content due to limitations in the models or scenarios in which the system cannot be applied in a way that properly manages potential negative consequences to people and society. Without limiting the foregoing restriction, Microsoft reserves the right to revise and expand the above Content requirements to address specific harms to people and society.

This includes content that is sexually graphic, including consensual pornographic content and intimate descriptions of sexual acts, as well as content that may influence the political process, such as an election, passage of legislation, and content for campaigning purposes;

We may at times limit our service's ability to respond to particular topics, such as probing for personal information or seeking opinions on sensitive topics or current events.

We prohibit the use of Azure OpenAI Service for activities that significantly harm other individuals, organizations, or society, including but not limited to use of the service for purposes in conflict with the applicable Azure Legal Terms and the Microsoft Product Terms.

Report abuse

If you suspect that Azure OpenAI Service is being used in a manner that is abusive or illegal, infringes on your rights or the rights of other people, or violates these policies you can report it at the Report Abuse Portal.

Report problematic content

If Azure OpenAI Service outputs problematic content that you believe should have been filtered, report it at cscraireport@microsoft.com.

See also

- Limited access to Azure OpenAI Service
- Transparency note for Azure OpenAI Service
- Data, privacy, and security for Azure OpenAI Service

Data, privacy, and security for Azure OpenAI Service

Article • 01/30/2023 • 5 minutes to read

This article provides details regarding how data provided by you to the Azure OpenAI service is processed, used, and stored. Azure OpenAI stores and processes data to provide the service, monitor for abusive use, and to develop and improve the quality of Azure's Responsible AI systems. Please also see the Microsoft Products and Services Data Protection Addendum, which governs data processing by the Azure OpenAI Service except as otherwise provided in the applicable Product Terms.

Azure OpenAI was designed with compliance, privacy, and security in mind; however, the customer is responsible for its use and the implementation of this technology.

What data does the Azure OpenAI Service process?

Azure OpenAI processes the following types of data:

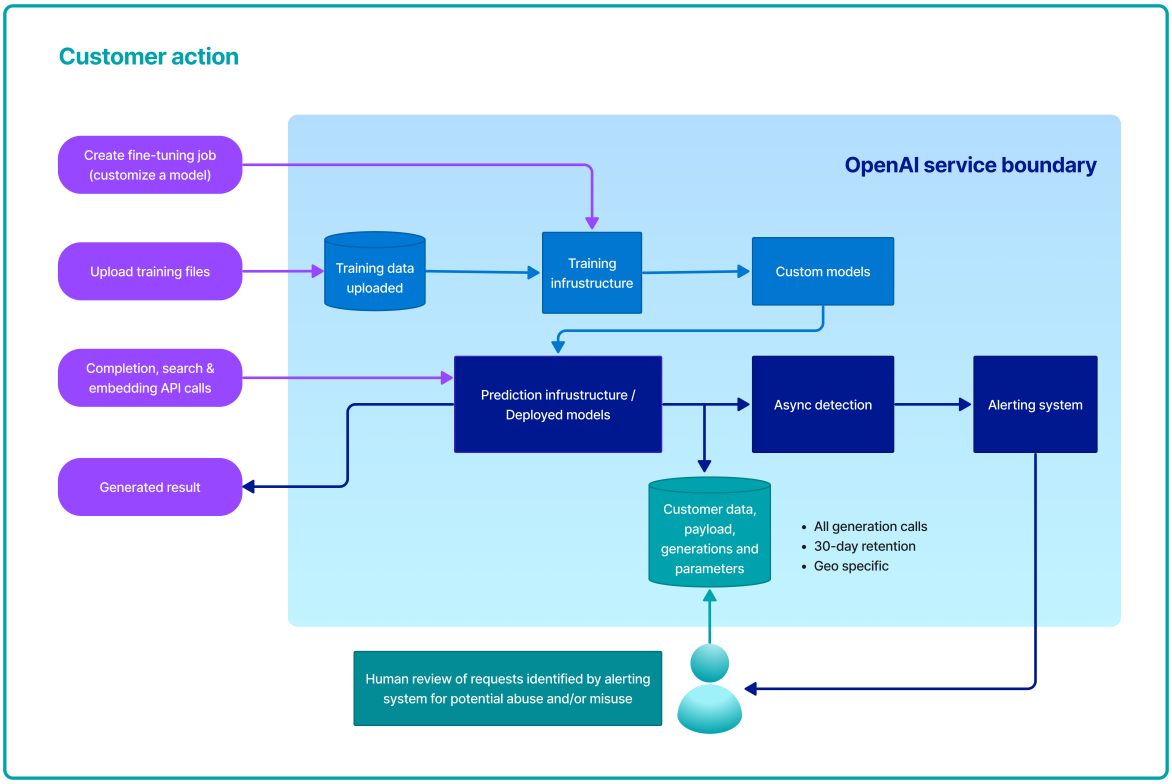
- **Text prompts, queries and responses** submitted by the user via the completions, search, and embeddings operations.
- **Training & validation data**. You can provide your own training data consisting of prompt-completion pairs for the purposes of fine-tuning an OpenAI model.
- **Results data from training process**. After training a fine-tuned model, the service will output meta-data on the job which includes tokens processed and validation scores at each step.

How does the Azure OpenAI Service process data?

The diagram below illustrates how your data is processed. This diagram covers three different types of processing:

1. How the Azure OpenAI Service creates a fine-tuned (custom) model with your training data
2. How the Azure OpenAI Service processes your text prompts to generate completions, embeddings, and search results; and

3. How the Azure OpenAI Service and Microsoft personnel analyze prompts & completions for abuse, misuse or harmful content generation.



Training data for purposes of fine-tuning an OpenAI model

The training data (prompt-completion pairs) submitted to the Fine-tunes API through the Azure OpenAI Studio is pre-processed using automated tools for quality checking including data format check. The training data is then imported to the model training component on the Azure OpenAI platform. During the training process, the training data are decomposed into batches and used to modify the weights of the OpenAI models.

Training data provided by the customer is only used to fine-tune the customer's model and is not used by Microsoft to train or improve any Microsoft models.

Text prompts to generate completions, embeddings and search results

Once a model is deployed, you can generate text using this model using our Completions operation through the REST API, client libraries or Azure OpenAI Studio.

Abuse and harmful content generation

The Azure OpenAI Service stores prompts & completions from the service to monitor for abusive use and to develop and improve the quality of Azure OpenAI's content management systems. Learn more about our content management and filtering. Authorized Microsoft employees can access your prompt & completion data that has triggered our automated systems for the purposes of investigating and verifying potential abuse; for customers who have deployed Azure OpenAI Service in the European Union, the authorized Microsoft employees will be located in the European Union. This data may be used to improve our content management systems.

In the event of a confirmed policy violation, we may ask you to take immediate action to remediate the issue to and to prevent further abuse. Failure to address the issue may result in suspension or termination of Azure OpenAI resource access.

How is data retained and what Customer controls are available?

- **Training, validation, and training results data**. The Files API allows customers to upload their training data for the purpose of fine-tuning a model. This data is stored in Azure Storage, encrypted at rest by Microsoft Managed keys, within the same region as the resource and logically isolated with their Azure subscription and API Credentials. Uploaded files can be deleted by the user via the DELETE API operation.

- **Fine-tuned OpenAI models**. The Fine-tunes API allows customers to create their own fine-tuned version of the OpenAI models based on the training data that you have uploaded to the service via the Files APIs. The trained fine-tuned models are stored in Azure Storage in the same region, encrypted at rest and logically isolated with their Azure subscription and API credentials. Fine-tuned models can be deleted by the user by calling the DELETE API operation.

- **Text prompts, queries and responses**. The requests & response data may be temporarily stored by the Azure OpenAI Service for up to 30 days. This data is encrypted and is only accessible to authorized engineers for (1) debugging purposes in the event of a failure, (2) investigating patterns of abuse and misuse or (3) improving the content filtering system through using the prompts and completions flagged for abuse or misuse.

To learn more about Microsoft's privacy and security commitments visit the Microsoft Trust Center.

Frequently asked questions

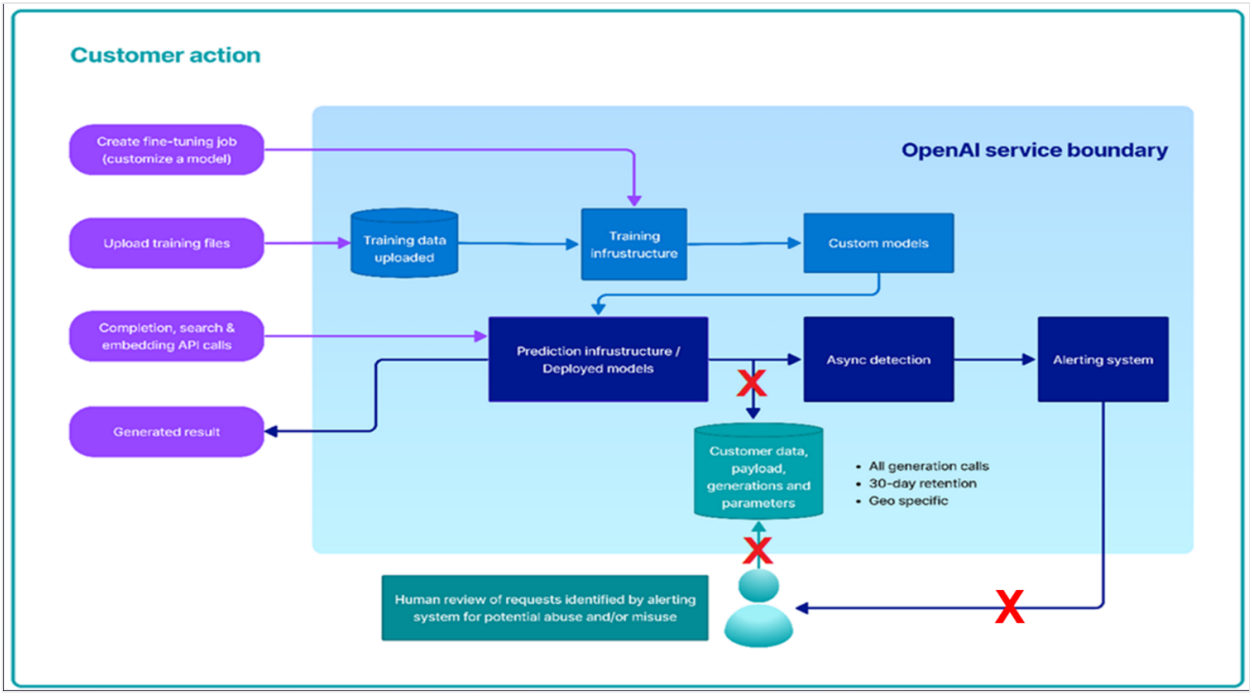
Can a customer opt out of the logging and human review process?

Some customers in highly regulated industries with low risk use cases process sensitive data with less likelihood of misuse. Because of the nature of the data or use case, these customers do not want or do not have the right to permit Microsoft to process such data for abuse detection due to their internal policies or applicable legal regulations.

To empower its enterprise customers and to strike a balance between regulatory / privacy needs and abuse prevention, the Azure Open AI Service will include a set of Limited Access features to provide potential customers with the option to modify following:

1. abuse monitoring
2. content filtering

These Limited Access features will enable potential customers to opt out of the human review and data logging processes subject to eligibility criteria governed by Microsoft's Limited Access framework. Customers who meet Microsoft's Limited Access eligibility criteria and have a low-risk use case can apply for the ability to opt-out of both data logging and human review process. This allows trusted customers with low-risk scenarios the data and privacy controls they require while also allowing us to offer AOAI models to all other customers in a way that minimizes the risk of harm and abuse.



If Microsoft approves a customer's request to access Limited Access features with the capability to (i) modify abuse monitoring and (ii) modify content filtering, then Microsoft will not store the associated request or response. Since no request or response data will be stored at rest in the Service Results Store in this case, the human review process will no longer be feasible. Therefore, both CMK and Lockbox will be deemed out-of-scope for harm and abuse detection.

See also

- Limited access to Azure OpenAI Service
- Code of conduct for Azure OpenAI Service integrations
- Transparency note and use cases for Azure OpenAI Service
- Characteristics and limitations for Azure OpenAI Service
- Report abuse of Azure OpenAI Service through the Report Abuse Portal
- Report problematic content to cscraireport@microsoft.com

Azure OpenAI REST API reference

Article • 01/26/2023 • 6 minutes to read

This article provides details on the REST API endpoints for the Azure OpenAI Service, a service in the Azure Cognitive Services suite. The REST APIs are broken up into two categories:

- **Management APIs**: The Azure Resource Manager (ARM) provides the management layer in Azure that allows you to create, update and delete resource in Azure. All services use a common structure for these operations. Learn More
- **Service APIs**: The Azure OpenAI service provides you with a set of REST APIs for interacting with the resources & models you deploy via the Management APIs.

Management APIs

The Azure OpenAI Service is deployed as a part of the Azure Cognitive Services. All Cognitive Services rely on the same set of management APIs for creation, update and delete operations. The management APIs are also used for deploying models within an OpenAI resource.

Management APIs reference documentation

Authentication

The Azure OpenAI service provides two methods for authentication. you can use either API Keys or Azure Active Directory.

- **API Key authentication**: For this type of authentication, all API requests must include the API Key in the `api-key` HTTP header. The Quickstart provides a tutorial for how to make calls with this type of authentication
- **Azure Active Directory authentication**: You can authenticate an API call using an Azure Active Directory token. Authentication tokens are included in a request as the `Authorization` header. The token provided must be preceded by `Bearer`, for example `Bearer YOUR_AUTH_TOKEN`. You can read our how-to guide on authenticating with Azure Active Directory.

REST API versioning

The service APIs are versioned using the `api-version` query parameter. All versions follow the YYYY-MM-DD date structure. For example:

```
HTTP

POST
https://YOUR_RESOURCE_NAME.openai.azure.com/openai/deployments/YOUR_DEPLOYME
NT_NAME/completions?api-version=2022-12-01
```

We currently have the following versions available: `2022-12-01`

Completions

With the Completions operation, the model will generate one or more predicted completions based on a provided prompt. The service can also return the probabilities of alternative tokens at each position.

Create a completion

```
HTTP

POST https://{your-resource-
name}.openai.azure.com/openai/deployments/{deployment-id}/completions?api-
version={api-version}
```

Path parameters

Parameter	Type	Required?	Description
`your-resource-name`	string	Required	The name of your Azure OpenAI Resource.
`deployment-id`	string	Required	The name of your model deployment. You're required to first deploy a model before you can make calls
`api-version`	string	Required	The API version to use for this operation. This follows the YYYY-MM-DD format.

Supported versions

- `2022-12-01`

Request body

Parameter	Type	Required?	Default	Description
`prompt`	string or array	Optional	`<|endoftext|>`	The prompt(s) to generate completions for, encoded as a string, a list of strings, or a list of token lists. Note that `<|endoftext|>` is the document separator that the model sees during training, so if a prompt isn't specified the model will generate as if from the beginning of a new document.
`max_tokens`	integer	Optional	16	The maximum number of tokens to generate in the completion. The token count of your prompt plus max_tokens can't exceed the model's context length. Most models have a context length of 2048 tokens (except davinci-codex, which supports 4096).
`temperature`	number	Optional	1	What sampling temperature to use. Higher values means the model will take more risks. Try 0.9 for more creative applications, and 0 (`argmax sampling`) for ones with a well-defined answer. We generally recommend altering this or top_p but not both.
`top_p`	number	Optional	1	An alternative to sampling with temperature, called nucleus sampling, where the model considers the results of the tokens with top_p probability mass. So 0.1 means only the tokens comprising the top 10% probability mass are considered. We generally recommend altering this or temperature but not both.

Parameter	Type	Required?	Default	Description
`n`	integer	Optional	1	How many completions to generate for each prompt. Note: Because this parameter generates many completions, it can quickly consume your token quota. Use carefully and ensure that you have reasonable settings for max_tokens and stop.
`stream`	boolean	Optional	False	Whether to stream back partial progress. If set, tokens will be sent as data-only server-sent events as they become available, with the stream terminated by a data: [DONE] message.
`logprobs`	integer	Optional	null	Include the log probabilities on the logprobs most likely tokens, as well the chosen tokens. For example, if logprobs is 10, the API will return a list of the 10 most likely tokens. the API will always return the logprob of the sampled token, so there may be up to logprobs+1 elements in the response.
`echo`	boolean	Optional	False	Echo back the prompt in addition to the completion
`stop`	string or array	Optional	null	Up to four sequences where the API will stop generating further tokens. The returned text won't contain the stop sequence.
`presence_penalty`	number	Optional	0	Number between -2.0 and 2.0. Positive values penalize new tokens based on whether they appear in the text so far, increasing the model's likelihood to talk about new topics.

Parameter	Type	Required?	Default	Description
`frequency_penalty`	number	Optional	0	Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency in the text so far, decreasing the model's likelihood to repeat the same line verbatim.
`best_of`	integer	Optional	1	Generates best_of completions server-side and returns the "best" (the one with the lowest log probability per token). Results can't be streamed. When used with n, best_of controls the number of candidate completions and n specifies how many to return – best_of must be greater than n. Note: Because this parameter generates many completions, it can quickly consume your token quota. Use carefully and ensure that you have reasonable settings for max_tokens and stop.

Parameter	Type	Required?	Default	Description
`logit_bias`	map	Optional	null	Modify the likelihood of specified tokens appearing in the completion. Accepts a json object that maps tokens (specified by their token ID in the GPT tokenizer) to an associated bias value from -100 to 100. You can use this tokenizer tool (which works for both GPT-2 and GPT-3) to convert text to token IDs. Mathematically, the bias is added to the logits generated by the model prior to sampling. The exact effect will vary per model, but values between -1 and 1 should decrease or increase likelihood of selection; values like -100 or 100 should result in a ban or exclusive selection of the relevant token. As an example, you can pass {"50256": -100} to prevent the <|endoftext|> token from being generated.

Example request

Console

```
curl
https://YOUR_RESOURCE_NAME.openai.azure.com/openai/deployments/YOUR_DEPLOYME
NT_NAME/completions?api-version=2022-12-01\
  -H "Content-Type: application/json" \
  -H "api-key: YOUR_API_KEY" \
  -d "{
  \"prompt\": \"Once upon a time\",
  \"max_tokens\": 5
}"
```

Example response

JSON

```
{
    "id": "cmpl-4kGh7iXtjW4lc9eGhff6Hp8C7btdQ",
    "object": "text_completion",
```

```
    "created": 1646932609,
    "model": "ada",
    "choices": [
        {
            "text": ", a dark line crossed",
            "index": 0,
            "logprobs": null,
            "finish_reason": "length"
        }
    ]
}
```

Embeddings

Get a vector representation of a given input that can be easily consumed by machine learning models and other algorithms.

Create an embedding

HTTP

```
POST https://{your-resource-name}.openai.azure.com/openai/deployments/{deployment-id}/embeddings?api-version={api-version}
```

Path parameters

Parameter	Type	Required?	Description
`your-resource-name`	string	Required	The name of your Azure OpenAI Resource.
`deployment-id`	string	Required	The name of your model deployment. You're required to first deploy a model before you can make calls
`api-version`	string	Required	The API version to use for this operation. This follows the YYYY-MM-DD format.

Supported versions

- `2022-12-01`

Request body

Parameter	Type	Required?	Default	Description

Parameter	Type	Required?	Default	Description
`input`	string or array	Yes	N/A	Input text to get embeddings for, encoded as a string or array of tokens. To get embeddings for multiple inputs in a single request, pass an array of strings or array of token arrays. Each input must not exceed 2048 tokens in length. Currently we accept a max array of 1. Unless you're embedding code, we suggest replacing newlines (\n) in your input with a single space, as we have observed inferior results when newlines are present.
`user`	string	No	Null	A unique identifier representing for your end-user. This will help Azure OpenAI monitor and detect abuse. **Do not pass PII identifiers instead use pseudoanonymized values such as GUIDs**

Example request

Console

```
curl
https://YOUR_RESOURCE_NAME.openai.azure.com/openai/deployments/YOUR_DEPLOYME
NT_NAME/embeddings?api-version=2022-12-01\
  -H "Content-Type: application/json" \
  -H "api-key: YOUR_API_KEY" \
  -d "{\"input\": \"The food was delicious and the waiter...\"}"
```

Example response

JSON

```
{
  "object": "list",
  "data": [
    {
      "object": "embedding",
      "embedding": [
        0.018990106880664825,
        -0.0073809814639389515,
        .... (1024 floats total for ada)
        0.021276434883475304,
      ],
      "index": 0
    }
  ],
```

```
  "model": "text-similarity-babbage:001"
}
```

Next steps

Learn about managing deployments, models, and finetuning with the REST API. Learn more about the underlying models that power Azure OpenAI.

Additional resources

Deployments - Create

Reference

Service: Cognitive Services

API Version: 2022-12-01

Creates a new deployment for the Azure OpenAI resource according to the given specification.

In this article

HTTP

```
POST {endpoint}/openai/deployments?api-version=2022-12-01
```

URI Parameters

Name	In	Required	Type	Description
`endpoint`	path	True	string url	Supported Cognitive Services endpoints (protocol and hostname, for example: https://aoairesource.openai.azure.com. Replace "aoairesource" with your Azure OpenAI account name).
`api-version`	query	True	string	The requested API version.

Request Header

Name	Required	Type	Description

api-key	True	string	Provide your Cognitive Services Azure OpenAI account key here.

Request Body

Name	Required	Type	Description
model	True	string	The OpenAI model identifier (model-id) to deploy. Can be a base model or a fine tune.
scale_settings	True	ScaleSettings: ManualScale Settings Standard ScaleSettings	ScaleSettings The scale settings of a deployment. It defines the modes for scaling and the reserved capacity.
error		Error	Error Error content as defined in the Microsoft REST guidelines (https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md#7102-error-condition-responses).

Responses

Name	Type	Description
201 Created	Deployment	The deployment has been successfully created. Headers Location: string
Other Status Codes	Error Response	An error occurred.

Security

api-key

Provide your Cognitive Services Azure OpenAI account key here.

Type: apiKey
In: header

Examples

Creating a deployment.

Sample Request

HTTP

HTTP

```
POST https://aoairesource.openai.azure.com/openai/deployments?api-version=2022-12-01


{
  "scale_settings": {
    "capacity": 2,
    "scale_type": "manual"
  },
  "model": "curie"
}
```

Sample Response

Status code: 201

HTTP

```
location:
https://aoairesource.openai.azure.com/openai/deployments/deployment-afa0669ca01e4693ae3a93baf40f26d6
```

Response Body

JSON

```
{
  "scale_settings": {
    "capacity": 2,
    "scale_type": "manual"
  },
  "model": "curie",
  "owner": "organization-owner",
  "id": "deployment-afa0669ca01e4693ae3a93baf40f26d6",
  "status": "notRunning",
  "created_at": 1646126127,
  "updated_at": 1646127311,
  "object": "deployment"
}
```


Definitions

Deployment	Deployment
Error	Error
ErrorCode	ErrorCode
ErrorResponse	ErrorResponse
InnerError	InnerError
InnerErrorCode	InnerErrorCode
ManualScaleSettings	ManualScaleSettings
ScaleType	ScaleType
StandardScaleSettings	StandardScaleSettings
State	State
TypeDiscriminator	TypeDiscriminator

Deployment

Deployment

Name	Type	Description
created_at	integer	A timestamp when this job or item was created (in unix epochs).
error	Error	Error Error content as defined in the Microsoft REST guidelines (https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md#7102-error-condition-responses).
id	string	The identity of this item.
model	string	The OpenAI model identifier (model-id) to deploy. Can be a base model or a fine tune.
object	Type Discriminator	TypeDiscriminator Defines the type of an object.
owner	string	The owner of this deployment. For Azure OpenAI only "organization-owner" is supported.
scale_settings	ScaleSettings: ManualScale Settings Standard ScaleSettings	ScaleSettings The scale settings of a deployment. It defines the modes for scaling and the reserved capacity.
status	State	State The state of a job or item.
updated_at	integer	A timestamp when this job or item was modified last (in unix epochs).

Error

Error

Name	Type	Description

code	Error Code	ErrorCode Error codes as defined in the Microsoft REST guidelines (https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md#7102-error-condition-responses).
details	Error[]	The error details if available.
innererror	Inner Error	InnerError Inner error as defined in the Microsoft REST guidelines (https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md#7102-error-condition-responses).
message	string	The message of this error.
target	string	The location where the error happened if available.

ErrorCode

ErrorCode

Name	Type	Description
conflict	string	The requested operation conflicts with the current resource state.
fileImportFailed	string	Import of file failed.
forbidden	string	The operation is forbidden for the current user/api key.
internalFailure	string	Internal error. Please retry.
invalidPayload	string	The request data is invalid for this operation.
itemDoesAlreadyExist	string	The item does already exist.
jsonlValidationFailed	string	Validation of jsonl data failed.

notFound	string	The resource is not found.
quotaExceeded	string	Quota exceeded.
serviceUnavailable	string	The service is currently not available.
unexpectedEntityState	string	The operation cannot be executed in the current resource's state.

ErrorResponse

ErrorResponse

Name	Type	Description
error	Error	Error Error content as defined in the Microsoft REST guidelines (https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md#7102-error-condition-responses).

InnerError

InnerError

Name	Type	Description
code	Inner Error Code	InnerErrorCode Inner error codes as defined in the Microsoft REST guidelines (https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md#7102-error-condition-responses).
innererror	Inner Error	InnerError Inner error as defined in the Microsoft REST guidelines (https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md#7102-error-condition-responses).

InnerErrorCode

InnerErrorCode

Name	Type	Description
invalidPayload	string	The request data is invalid for this operation.

ManualScaleSettings

ManualScaleSettings

Name	Type	Description
capacity	integer	The constant reserved capacity of the inference endpoint for this deployment.
scale_type	string: manual	ScaleType Defines how scaling operations will be executed.

ScaleType

ScaleType

Name	Type	Description
manual	string	Scaling of a deployment will happen by manually specifying the capacity of a model.
standard	string	Scaling of a deployment will happen automatically based on usage.

StandardScaleSettings

StandardScaleSettings

Name	Type	Description

scale_type	string: standard	ScaleType Defines how scaling operations will be executed.

State

State

Name	Type	Description
canceled	string	The operation has been canceled and is incomplete.
deleted	string	The entity has been deleted but may still be referenced by other entities predating the deletion.
failed	string	The operation has completed processing with a failure and cannot be further consumed.
notRunning	string	The operation was created and is not queued to be processed in the future.
running	string	The operation has started to be processed.
succeeded	string	The operation has successfully be processed and is ready for consumption.

TypeDiscriminator

TypeDiscriminator

Name	Type	Description
deployment	string	This object represents a deployment.
file	string	This object represents a file.
fine-tune	string	This object represents a fine tune job.
fine-tune-event	string	This object represents an event of a fine tune job.

list	string	This object represents a list of other objects.
model	string	This object represents a model (can be a base models or fine tune job result).

Azure Cognitive Services support and help options

Article • 07/22/2022 • 2 minutes to read

Are you just starting to explore the functionality of Azure Cognitive Services? Perhaps you are implementing a new feature in your application. Or after using the service, do you have suggestions on how to improve it? Here are options for where you can get support, stay up-to-date, give feedback, and report bugs for Cognitive Services.

Create an Azure support request

Explore the range of Azure support options and choose the plan that best fits, whether you're a developer just starting your cloud journey or a large organization deploying business-critical, strategic applications. Azure customers can create and manage support requests in the Azure portal.

- Azure portal
- Azure portal for the United States government

Post a question on Microsoft Q&A

For quick and reliable answers on your technical product questions from Microsoft Engineers, Azure Most Valuable Professionals (MVPs), or our expert community, engage with us on Microsoft Q&A, Azure's preferred destination for community support.

If you can't find an answer to your problem using search, submit a new question to Microsoft Q&A. Use one of the following tags when you ask your question:

- Cognitive Services

Vision

- Computer Vision
- Custom Vision
- Face
- Form Recognizer
- Video Indexer

Language

- Immersive Reader
- Language Understanding (LUIS)
- QnA Maker
- Language service
- Translator

Speech

- Speech service

Decision

- Anomaly Detector
- Content Moderator
- Metrics Advisor
- Personalizer

Azure OpenAI

- Azure OpenAI

Post a question to Stack Overflow

For answers on your developer questions from the largest community developer ecosystem, ask your question on Stack Overflow.

If you do submit a new question to Stack Overflow, please use one or more of the following tags when you create the question:

- Cognitive Services

Vision

- Computer Vision
- Custom Vision
- Face
- Form Recognizer
- Video Indexer

Language

- Immersive Reader
- Language Understanding (LUIS)

- QnA Maker
- Language service
- Translator

Speech

- Speech service

Decision

- Anomaly Detector
- Content Moderator
- Metrics Advisor
- Personalizer

Azure OpenAI

- Azure OpenAI

Submit feedback

To request new features, post them on https://feedback.azure.com. Share your ideas for making Cognitive Services and its APIs work better for the applications you develop.

- Cognitive Services

Vision

- Computer Vision
- Custom Vision
- Face
- Form Recognizer
- Video Indexer

Language

- Immersive Reader
- Language Understanding (LUIS)
- QnA Maker
- Language service
- Translator

Speech

- Speech service

Decision

- Anomaly Detector
- Content Moderator
- Metrics Advisor
- Personalizer

Stay informed

Staying informed about features in a new release or news on the Azure blog can help you find the difference between a programming error, a service bug, or a feature not yet available in Cognitive Services.

- Learn more about product updates, roadmap, and announcements in Azure Updates.
- News about Cognitive Services is shared in the Azure blog.
- Join the conversation on Reddit about Cognitive Services.

Next steps

What are Azure Cognitive Services?