6.5 - Implement Retrieval Augmented Generation (RAG) with Azure OpenAI Service

Overview

Azure OpenAI on your data allows developers to implement RAG with supported AI chat models to reference specific sources of data to ground the response.

By the end of this module, you'll be able to:

- Describe the capabilities of Azure OpenAI on your data
- Configure Azure OpenAI to use your own data
- Use Azure OpenAI API to generate responses based on your own data

---

Introduction

Azure OpenAI enables developers to implement Retrieval Augmented Generation (RAG) by connecting supporting AI chat models to your own data. Those models can reference specific sources of data to ground the response, augmenting the capabilities of the AI model when it creates a response.

In this module, you'll learn how to implement RAG by adding your own data with Azure OpenAI and generate responses based on both your data and the Azure OpenAI pretrained knowledge.

Understand Retrieval Augmented Generation (RAG) with Azure OpenAI Service

RAG with Azure OpenAI allows developers to use supported AI chat models that can reference specific sources of information to ground the response. Adding this information allows the model to reference both the specific data provided and its pretrained knowledge to provide more effective responses.

Azure OpenAI enables RAG by connecting pretrained models to your own data sources.

Azure OpenAI on your data utilizes the search ability of Azure AI Search to add the relevant data chunks to the prompt. Once your data is in an AI Search index, Azure OpenAI on your data goes through the following steps:

1. Receive user prompt.
2. Determine relevant content and intent of the prompt.
3. **Query the search index with that content and intent.**
4. Insert search result chunk into the Azure OpenAI prompt, along with system message and user prompt.
5. Send entire prompt to Azure OpenAI.
6. Return response and data reference (if any) to the user.

By default, Azure OpenAI on your data **encourages, but doesn't require, the model to respond only using your data.** This setting can be unselected when connecting your data, which may result in the model choosing to use its pretrained knowledge over your data.

Fine-tuning vs. RAG

Fine-tuning is a technique used to create a custom model by training an existing foundational model such as `gpt-35-turbo` with a dataset of additional training data.

Fine-tuning can result in higher quality requests than prompt engineering alone, customize the model on examples larger than can fit in a prompt, and allow the user to provide fewer examples to get the same high quality response. However, the process for fine-tuning is both costly and time intensive, and should only be used for use cases where it's necessary.

RAG with Azure OpenAI on your data still uses the stateless API to connect to the model, which removes the requirement of training a custom model with your data and simplifies the interaction with the AI model.

AI Search first finds the useful information to answer the prompt, it then adds that information to the prompt as grounding data, and Azure OpenAI then forms the response based on that information.

Add your own data source

Adding your data can be done through the Azure AI Foundry portal, in the **Chat** playground, or by **specifying your data source in an API call.**

The data source you add is then used to augment the prompt sent to the model. When setting up your data in the AI Foundry, you can choose to:

- Upload your data files
- Use data in a blob storage account
- Connect to an existing AI Search index.

If you're uploading or using files already in a storage account, Azure OpenAI on your data supports `.md`, `.txt`, `.html`, `.pdf`, and Microsoft Word or PowerPoint files. If any of these files contain graphics or images, the response quality depends on how well text can be extracted from the visual content.

When uploading data or connecting to files in a storage account, it's recommended to use the Azure AI Foundry to create the search resource and index.

Adding data this way allows the appropriate chunking to happen when inserting into the index, yielding better responses.

If you're using large text files or forms, you should use the available data preparation script to improve the AI model's accuracy.

Enabling semantic search for your AI Search service can improve the result of searching your data index and you're likely to receive higher quality responses and citations. However, enabling semantic search may increase the cost of the search service.

You can also use the wizard in your AI Search resource to vectorize your data appropriately, which you'll walk through in this module's exercise.

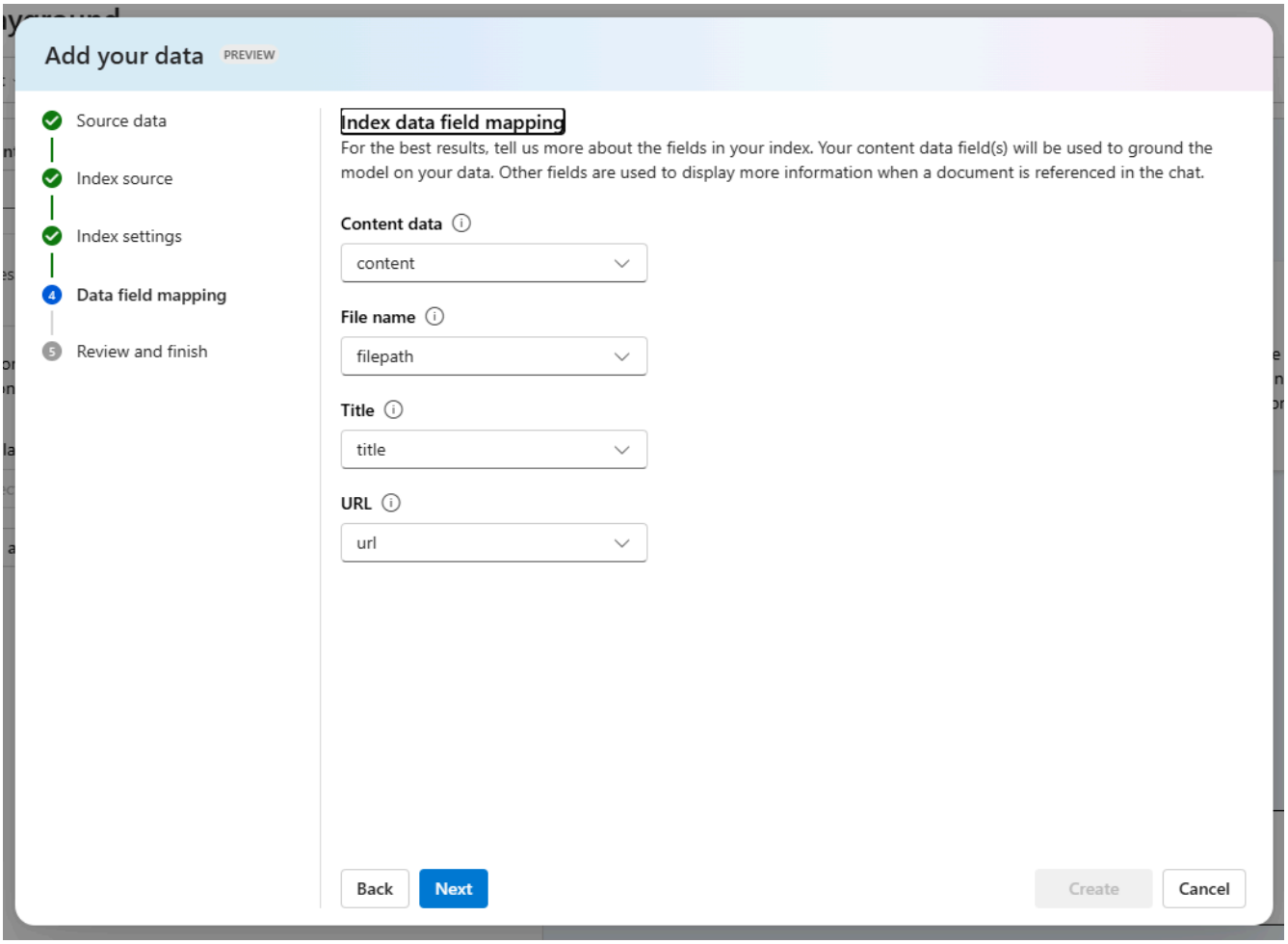
It takes a few extra steps compared to doing so in AI Studio, however serves as a good example of using the RAG pattern with an existing dataset.

Connect your data in Azure AI Studio

To connect your data, navigate to the **Chat** playground in Azure AI Studio and select the **Add your data** tab. Select the **Add a data source** button to get your data connected. The prompts guide you through setting up the connection to each data source, and getting that data into a search index.

> Tip: If you are using the wizard in AI Studio to create and connect your data source, you will need to create a hub and a project, which aren't covered here. AI Studio will walk you through doing that, or refer to the AI Studio documentation.

If you're using your own index that wasn't created through Azure AI Studio, one of the pages allows you to specify your column mapping. It's important to provide accurate fields, to enable the model to provide a better response, especially for **Content data**.



Chat with your model using your own data

RAG with Azure OpenAI on your own data can be used in Azure AI Studio with the **Chat** playground, or by using the API.

Token considerations and recommended settings

Since RAG with Azure OpenAI on your data includes search results on your index in the prompt, it's important to understand how that impacts your token allotment.

Each call to the model includes tokens for the system message, the user prompt, conversation history, retrieved search documents, internal prompts, and the model's response.

The system message, for example, is a useful reference for instructions for the model and is included with every call. While there's no token limit for the system message, **when using your own data the system message gets truncated if it exceeds the model's token limit** (which varies per model, from 400 to 4000 tokens). The response from the model is also limited when using your own data to 1500 tokens.

Due to these token limitations, it's recommended that you limit both the question length and the conversation history length in your call. Prompt engineering techniques such as breaking down the task and chain of thought prompting can help the model respond more effectively.

Using the API

Using the API with your own data, you need to **specify the data source where your data is stored.** With each call you need to include the `endpoint`, `key`, and `indexName` for your AI Search resource.

Your request body will be similar to the following JSON.

```
{
    "dataSources": [
        {
            "type": "AzureCognitiveSearch",
            "parameters": {
                "endpoint": "<your_search_endpoint>",
                "key": "<your_search_endpoint>",
                "indexName": "<your_search_index>"
            }
        }
    ],
    "messages":[
        {
            "role": "system",
            "content": "You are a helpful assistant assisting users with travel
recommendations."
        },
        {
            "role": "user",
            "content": "I want to go to New York. Where should I stay?"
        }
    ]
}
```

The call when using your own data needs to be sent to a different endpoint than is used when calling a base model, which includes `extensions`. Your call will be sent to a URL similar to the following.

```
<your_azure_openai_resource>/openai/deployments/<deployment_name>/chat/completions?api-
version=<version>
```

The request will also need to include the `Content-Type` and `api-key`.

Exercise - Add your data for RAG with Azure OpenAI Service

The Azure OpenAI Service enables you to use your own data with the intelligence of the underlying LLM. You can limit the model to only use your data for pertinent topics, or blend it with results from the pre-trained model.

In the scenario for this exercise, you will perform the role of a software developer working for Margie's Travel Agency. You will explore how use Azure AI Search to index your own data and use it with Azure OpenAI to augment prompts.

This exercise will take approximately **30** minutes.

Provision Azure resources

To complete this exercise, you'll need:

- An Azure OpenAI resource.
- An Azure AI Search resource.
- An Azure Storage Account resource.

1. Sign into the **Azure portal** at `https://portal.azure.com`.
2. Create an **Azure OpenAI** resource with the following settings:
   - **Subscription**: *Select an Azure subscription that has been approved for access to the Azure OpenAI service*
   - **Resource group**: *Choose or create a resource group*
   - **Region**: *Make a **random** choice from any of the following regions**
     - Canada East
     - East US
     - East US 2
     - France Central
     - Japan East
     - North Central US
     - Sweden Central
     - Switzerland North
     - UK South
   - **Name**: *A unique name of your choice*
   - **Pricing tier**: Standard S0

   > - Azure OpenAI resources are constrained by regional quotas. The listed regions include default quota for the model type(s) used in this exercise. Randomly choosing a region reduces the risk of a single region reaching its quota limit in scenarios where you are sharing a subscription with other users. In the event of a quota limit being reached later in the exercise, there's a possibility you may need to create another resource in a different region.

3. While the Azure OpenAI resource is being provisioned, create an **Azure AI Search** resource with the following settings:

- **Subscription**: *The subscription in which you provisioned your Azure OpenAI resource*
- **Resource group**: *The resource group in which you provisioned your Azure OpenAI resource*
- **Service name**: *A unique name of your choice*
- **Location**: *The region in which you provisioned your Azure OpenAI resource*
- **Pricing tier**: Basic

4. While the Azure AI Search resource is being provisioned, create a **Storage account** resource with the following settings:
   - **Subscription**: *The subscription in which you provisioned your Azure OpenAI resource*
   - **Resource group**: *The resource group in which you provisioned your Azure OpenAI resource*
   - **Storage account name**: *A unique name of your choice*
   - **Region**: *The region in which you provisioned your Azure OpenAI resource*
   - **Primary service**: Azure Blob Storage or Azure Data Lake Storage Gen 2
   - **Performance**: Standard
   - **Redundancy**: Locally redundant storage (LRS)
5. After all three of the resources have been successfully deployed in your Azure subscription, review them in the Azure portal and gather the following information (which you'll need later in the exercise):
   - The **endpoint** and a **key** from the Azure OpenAI resource you created (available on the **Keys and Endpoint** page for your Azure OpenAI resource in the Azure portal)
   - The endpoint for your Azure AI Search service (the **Url** value on the overview page for your Azure AI Search resource in the Azure portal).
   - A **primary admin key** for your Azure AI Search resource (available in the **Keys** page for your Azure AI Search resource in the Azure portal).

Upload your data

You're going to ground the prompts you use with a generative AI model by using your own data. In this exercise, the data consists of a collection of travel brochures from the fictional *Margies Travel* company.

1. In a new browser tab, download an archive of brochure data from `https://aka.ms/own-data-brochures`. Extract the brochures to a folder on your PC.
2. In the Azure portal, navigate to your storage account and view the **Storage browser** page.
3. Select **Blob containers** and then add a new container named `margies-travel`.
4. Select the **margies-travel** container, and then upload the .pdf brochures you extracted previously to the root folder of the blob container.

Deploy AI models

You're going to use two AI models in this exercise:

- **A text embedding model to *vectorize* the text in the brochures so it can be indexed efficiently for use in grounding prompts.**
- A GPT model that you application can use to generate responses to prompts that are grounded in your data.

Deploy a model

Next, you will deploy an Azure OpenAI model resource from the CLI. In the Azure portal; select **Cloud Shell** icon from the top menu bar and ensure that your terminal is set to **Bash**. Refer to this example and replace the following variables with your own values:

```
az cognitiveservices account deployment create \
    -g *your resource group* \
    -n *your Open AI resource* \
    --deployment-name text-embedding-ada-002 \
    --model-name text-embedding-ada-002 \
    --model-version "2" \
    --model-format OpenAI \
    --sku-name "Standard" \
    --sku-capacity 5
```

```
> \* Sku-capacity is measured in thousands of tokens per minute. A rate limit of 5,000 tokens per minute is more than adequate to complete this exercise while leaving capacity for other people using the same subscription.
```

After the text embedding model has been deployed, create a new deployment of the **gpt-35-turbo-16k** model with the following settings:

```
az cognitiveservices account deployment create \
    -g *your resource group* \
    -n *your Open AI resource* \
    --deployment-name gpt-35-turbo-16k \
    --model-name gpt-35-turbo-16k \
    --model-version "0125" \
    --model-format OpenAI \
    --sku-name "Standard" \
    --sku-capacity 5
```

```
> \* Sku-capacity is measured in thousands of tokens per minute. A rate limit of 5,000 tokens per minute is more than adequate to complete this exercise while leaving capacity for other people using the same subscription.
```

Create an index

To make it easy to use your own data in a prompt, you'll **index it using Azure AI Search**.

You'll use the text embedding model to *vectorize* the text data (which results in each text token in the index being represented by numeric vectors - making it compatible with the way a generative AI model represents text)

1. In the Azure portal, navigate to your Azure AI Search resource.
2. On the **Overview** page, select **Import and vectorize data**.
3. In the **Setup your data connection** page, select **Azure Blob Storage** and configure the data source with the following settings:
    - **Subscription**: The Azure subscription in which you provisioned your storage account.
    - **Blob storage account**: The storage account you created previously.

- **Blob container**: margies-travel
- **Blob folder**: *Leave blank*
- **Enable deletion tracking**: Unselected
- **Authenticate using managed identity**: Unselected

4. On the **Vectorize your text** page, select the following settings:
   - **Kind**: Azure OpenAI
   - **Subscription**: The Azure subscription in which you provisioned your Azure OpenAI service.
   - **Azure OpenAI Service**: Your Azure OpenAI Service resource
   - **Model deployment**: *text-embedding-ada-002*
   - **Authentication type**: API key
   - **I acknowledge that connecting to an Azure OpenAI service will incur additional costs to my account**: Selected
5. On the next page, do not select the option to vectorize images or extract data with AI skills.
6. On the next page, enable semantic ranking and schedule the indexer to run once.
7. On the final page, set the **Objects name prefix** to `margies-index` and then create the index.

Prepare to develop an app in Visual Studio Code

Now let's explore the use of your own data in an app that uses the Azure OpenAI service SDK. You'll develop your app using Visual Studio Code. The code files for your app have been provided in a GitHub repo.

> **Tip**: If you have already cloned the **mslearn-openai** repo, open it in Visual Studio code. Otherwise, follow these steps to clone it to your development environment.

1. Start Visual Studio Code.
2. Open the palette (SHIFT+CTRL+P) and run a **Git: Clone** command to clone the `https://github.com/MicrosoftLearning/mslearn-openai` repository to a local folder (it doesn't matter which folder).
3. When the repository has been cloned, open the folder in Visual Studio Code.

   > **Note**: If Visual Studio Code shows you a pop-up message to prompt you to trust the code you are opening, click on **Yes, I trust the authors** option in the pop-up.

4. Wait while additional files are installed to support the C# code projects in the repo.

   > **Note**: If you are prompted to add required assets to build and debug, select **Not Now**.

Configure your application

Applications for both C# and Python have been provided, and both apps feature the same functionality. First, you'll complete some key parts of the application to enable using your Azure OpenAI resource.

1. In Visual Studio Code, in the **Explorer** pane, browse to the **Labfiles/02-use-own-data** folder and expand the **CSharp** or **Python** folder depending on your language preference. Each folder contains the language-specific files for an app into which you're going to integrate Azure OpenAI functionality.
2. Right-click the **CSharp** or **Python** folder containing your code files and open an integrated terminal. Then install the Azure OpenAI SDK package by running the appropriate command for your language preference:

C#:

```
dotnet add package Azure.AI.OpenAI --version 1.0.0-beta.17
```

Python:

```
pip install openai==1.54.3
```

3. In the **Explorer** pane, in the **CSharp** or **Python** folder, open the configuration file for your preferred language
   - **C#**: appsettings.json
   - **Python**: .env
4. Update the configuration values to include:
   - The **endpoint** and a **key** from the Azure OpenAI resource you created (available on the **Keys and Endpoint** page for your Azure OpenAI resource in the Azure portal)
   - The **deployment name** you specified for your gpt-35-turbo model deployment (should be `gpt-35-turbo-16k`).
   - The endpoint for your search service (the **Url** value on the overview page for your search resource in the Azure portal).
   - A **key** for your search resource (available in the **Keys** page for your search resource in the Azure portal - you can use either of the admin keys)
   - The name of the search index (which should be `margies-index`).
5. Save the configuration file.

Add code to use the Azure OpenAI service

Now you're ready to use the Azure OpenAI SDK to consume your deployed model.

1. In the **Explorer** pane, in the **CSharp** or **Python** folder, open the code file for your preferred language, and replace the comment ***Configure your data source*** with code to add the Azure OpenAI SDK library:

   C#: ownData.cs

   `// Configure your data source AzureSearchChatExtensionConfiguration ownDataConfig = new() { SearchEndpoint = new Uri(azureSearchEndpoint), Authentication = new OnYourDataApiKeyAuthenticationOptions(azureSearchKey), IndexName = azureSearchIndex };`

   Python: ownData.py

```
# Configure your data source

text = input('\nEnter a question:\n')

completion = client.chat.completions.create(
    model=deployment,
    messages=[
        {"role": "user", "content": text},
    ],
    extra_body={
        "data_sources": [
```

```
            {
                "type": "azure_search",
                "parameters": {
                    "endpoint": os.environ["AZURE_SEARCH_ENDPOINT"],
                    "index_name": os.environ["AZURE_SEARCH_INDEX"],
                    "authentication": {
                        "type": "api_key",
                        "key": os.environ["AZURE_SEARCH_KEY"],
                    }
                }
            }
        ]
    }
)
```

2. Review the rest of the code, noting the use of the _extensions_ in the request body that is used to provide information about the data source settings.
3. Save the changes to the code file.

Run your application

Now that your app has been configured, run it to send your request to your model and observe the response. You'll notice the only difference between the different options is the content of the prompt, all other parameters (such as token count and temperature) remain the same for each request.

1. In the interactive terminal pane, ensure the folder context is the folder for your preferred language. Then enter the following command to run the application.
    - **C#**: `dotnet run`
    - **Python**: `python ownData.py`

    > **Tip**: You can use the **Maximize panel size** (**^**) icon in the terminal toolbar to see more of the console text.

2. Review the response to the prompt `Tell me about London`, which should include an answer as well as some details of the data used to ground the prompt, which was obtained from your search service.
    > **Tip**: If you want to see the **citations from your search index**, set the variable **_show citations_** near the top of the code file to **true**.

Knowledge Check

Summary

In this module, you learned how RAG with Azure OpenAI allows developers to use supported AI chat models to reference specific sources of data.

Connecting your own data allows the model to reference the specific information provided and its pretrained knowledge to provide more effective responses.

Additional Reading:

- [Azure OpenAI on your data](https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/use-your-data)
- [Azure AI Search](https://learn.microsoft.com/en-us/azure/search/)

✍️ Compiled by [Kenneth Leung](https://github.com/kennethleungty) (2025)