AI-LEARN-07b Azure OpenAI API (Learn)

58 Minutes Remaining

Instructions Resources Help  100%

**Tip**: As you follow the instructions in this pane, whenever you see a icon, you can use it to copy text from the instruction pane into the virtual machine interface. This is particularly useful to copy code; but bear in mind you may need to modify the pasted code to fix indent levels or formatting before running it!

You will need an Azure subscription that has been approved for access to the Azure OpenAI service. To request subscription access to the Azure OpenAI service, visit <https://aka.ms/oaiapply>. Access is restricted to approved Microsoft enterprise customers.

1. Sign into Windows as **Student** account with the password Pa55w.rd.
2. Follow the instructions below to complete the exercise.

Integrate Azure OpenAI into your app

With the Azure OpenAI Service, developers can create chatbots, language models, and other applications that excel at understanding natural human language. The Azure OpenAI provides access to pre-trained AI models, as well as a suite of APIs and tools for customizing and fine-tuning these models to meet the specific requirements of your application. In this exercise, you'll learn how to deploy a model in Azure OpenAI and use it in your own application to summarize text.

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

Provision an Azure OpenAI resource

If you don't already have one, provision an Azure OpenAI resource in your Azure subscription.

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 available regions*\*
   * **Name**: *A unique name of your choice*
   * **Pricing tier**: Standard S0

\* Azure OpenAI resources are constrained by regional quotas. 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.

1. Wait for deployment to complete. Then go to the deployed Azure OpenAI resource in the Azure portal.

Deploy a model

Azure OpenAI provides a web-based portal named **Azure OpenAI Studio**, that you can use to deploy, manage, and explore models. You'll start your exploration of Azure OpenAI by using Azure OpenAI Studio to deploy a model.

1. On the **Overview** page for your Azure OpenAI resource, use the **Go to Azure OpenAI Studio** button to open Azure OpenAI Studio in a new browser tab.
2. In Azure OpenAI Studio, on the **Deployments** page, view your existing model deployments. If you don't already have one, create a new deployment of the **gpt-35-turbo-16k** model with the following settings:
   * **Model**: gpt-35-turbo-16k *(if the 16k model isn't available, choose gpt-35-turbo)*
   * **Model version**: Auto-update to default
   * **Deployment name**: *A unique name of your choice. You'll use this name later in the lab.*
   * **Advanced options**
     + **Content filter**: Default
     + **Tokens per minute rate limit**: 5K\*
     + **Enable dynamic quota**: Enabled

\* 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.

Prepare to develop an app in Visual Studio Code

You'll develop your Azure OpenAI 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.
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, as well as a sample text file you'll use to test the summarization. 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-nlp-azure-openai** 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 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.9

**Python**:

pip install openai==1.2.0

1. In the **Explorer** pane, in the **CSharp** or **Python** folder, open the configuration file for your preferred language
   * **C#**: appsettings.json
   * **Python**: .env
2. 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 **model name** you specified for your model deployment (available in the **Deployments** page in Azure OpenAI Studio).
3. 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 ***Add Azure OpenAI package*** with code to add the Azure OpenAI SDK library:

**C#**: Program.cs

csharp

// Add Azure OpenAI package

using Azure.AI.OpenAI;

**Python**: test-openai-model.py

python

# Add Azure OpenAI package

from openai import AzureOpenAI

1. In the application code for your language, replace the comment ***Add code to build request…*** with the necessary code for building the request; specifying the various parameters for your model such as prompt and temperature.

**C#**: Program.cs

csharp

// Initialize the Azure OpenAI client

OpenAIClient client = new OpenAIClient(new Uri(oaiEndpoint), new AzureKeyCredential(oaiKey));

// Build completion options object

ChatCompletionsOptions chatCompletionsOptions = new ChatCompletionsOptions()

{

Messages =

{

new ChatMessage(ChatRole.System, "You are a helpful assistant."),

new ChatMessage(ChatRole.User, "Summarize the following text in 20 words or less:\n" + text),

},

MaxTokens = 120,

Temperature = 0.7f,

DeploymentName = oaiModelName

};

// Send request to Azure OpenAI model

ChatCompletions response = client.GetChatCompletions(chatCompletionsOptions);

string completion = response.Choices[0].Message.Content;

Console.WriteLine("Summary: " + completion + "\n");

**Python**: test-openai-model.py

python

# Initialize the Azure OpenAI client

client = AzureOpenAI(

azure\_endpoint = azure\_oai\_endpoint,

api\_key=azure\_oai\_key,

api\_version="2023-05-15"

)

# Send request to Azure OpenAI model

response = client.chat.completions.create(

model=azure\_oai\_model,

temperature=0.7,

max\_tokens=120,

messages=[

{"role": "system", "content": "You are a helpful assistant."},

{"role": "user", "content": "Summarize the following text in 20 words or less:\n" + text}

]

)

print("Summary: " + response.choices[0].message.content + "\n")

1. Save the changes to your code file.

Test your application

Now that your app has been configured, run it to send your request to your model and observe the response.

1. In the **Explorer** pane, expand the **Labfiles/02-nlp-azure-openai/text-files** folder and open the **sample-text.txt** file. This text file contains the text you will submit to the model to be summarized.
2. 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 test-openai-model.py

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

1. Observe the summarization of the sample text file.
2. In the code file for your preferred language, and change the *temperature* parameter value in your request to **1.0** and save the file.
3. Run the application again, and observe the output.
4. Re-run the app a few times, noting the output each time - it may vary.

Increasing the temperature often causes the summary to vary, even when provided the same text, due to the increased randomness. You can run it several times to see how the output may change. Try using different values for your temperature with the same input.

Clean up

When you're done with your Azure OpenAI resource, remember to delete the deployment or the entire resource in the **Azure portal** at https://portal.azure.com.