# Requirements

- Deploy an Azure OpenAI resource and an Azure OpenAI model
- Generate natural language responses by using Azure OpenAI
- Apply prompt engineering techniques by using Azure OpenAI
- Generate and improve code by using Azure OpenAI
- Generate images with DALL-E in Azure OpenAI
- Use Azure OpenAI on your data
- Be familiar with Azure services.
- Have experience developing applications by using C# or Python

# Get Started

- The models we will be using in this course are:
    - GPT-3.5-Turbo (GPT = Generative Pre-trained Transformer).
        - This provides advanced language AI with Azure security.
            - It uses a Large Language Model (LLM).
        - They use conversation-in and message-out, as opposed to earlier models, which were text-in and text-out.
        - It can understand and generate natural language and code.
        - The GPT-3.5-Turbo model is the most capable and cost-effective model in the GPT-3.5 family. Microsoft recommends using this instead of legacy GPT-3.5 and GPT-3 models.
        - At the time of recording, this is about 10 times cheaper than GPT-4 (which can solve difficult problems with improved accuracy).
    - DALL-E
        - This creates original image using natural language.
        - At the time of requirement, you need to use a
            - US East location to use DALL-E-2, or
            - Sweden Central location to use DALL-E-3.
- They are based on:
    - Prompt (input), and
    - Completion (output).
- You can use both the Completions API and Chat Completion API.
    - The Completions API is there for legacy reasons – it is compatible with GPT-3.
    - The Chat Completion API is the preferred version, and is compatible with GPT-4.
- The text models are priced in tokens, which are roughly syllables. On average, they are 4 characters long.
    - Pricing information: https://azure.microsoft.com/en-gb/pricing/details/cognitive-services/openai-service/
    - The pricing for GPT-3.5-Turbo is 0.15-0.2 US cents for 1,000 tokens.
    - The pricing for GPT-4 is 3-12 US cents for 1,000 tokens.
- The image models are priced by image:

 

    - For Dall-E-2, the price is 2 cents per image.
        - You can use up to 1,000 characters in your Request.
    - For Dall-E-3, the price is 8 cents per image.
        - You can use up to 4,000 characters in your Request.

## Natural Language Solutions

- The prompt is how users interact with GPT models.
    - What is the first thing that comes to your mind when I say …?
    - Simple instruction: *Write an introduction for a monthly newsletter*.
    - Complex instructions: *Write an introduction for a monthly newsletter named Power Services. It should mention the new 6-monthly release schedule, and the success of the previous schedule*.
    - Complex instructions: *Write an introduction for a monthly newsletter, including the following:*
        - *The newsletter is named Power Services*.
        - *A nice greeting*
        - *Thanking everyone for the improvements in the previous schedule*.
        - *Mentioning the new 6-monthly release schedule*
        - *Signed by Phillip Burton*.
- Primary Content refers to text which is to be processed or transformed.
    - *Can you please tell me how to get to the library?* (Primary content)
      *Translate to French* (instruction).
- Primary Content can be fairly long, and can be structured:

  *Go to the navigation pane on the left-hand side, and select "Publish".*
  *The computer will check for errors in the content.*
  *You can then use the "demo website" link.*
  *This is for your team or others who wish to try out the bot.*
  *It is not intended to use with customers.*
  *Then click on Channels to publish onto:*
  *Custom website (your own website),*
  *Mobile App,*
  *Facebook,*
  *Microsoft Teams,*
  *Skype,*
  *Cortana and*
  *Slack.*
  *The presentation may be different in different channels:*
  *Welcome messages are not supported in Facebook.*
  *Customer satisfaction surveys will be shown as an adaptive card on a website, but text-only in Teams and Facebook.*
  *Summarize the previous text in a few sentences*
- You can add cues – prime the output:

  Zero cues: *Summarize the previous text in a few sentences* OR *TL;DR* (Too long, didn't read).
  One cue: *Summarize the previous text in a few sentences.*
  *The key takeaway of the message is:*

 

Two cues: *Summarize the previous text in a few sentences.*
*Key Points.*
*1.*

- Supporting content is additional content which can be used with the primary content, but is not the main target of task.
  *The important topics are the different channels and the demo website link.*
- Best practices are:
  - Be as specific as possible.
  - Be descriptive,
  - Repeat yourself if necessary. Give instructions before and after your primary content, and use an instruction and a cue.
  - In GPT-3 and prior, where to put instructions – before or after the primary content - can make a difference.
    - Microsoft research shows that telling the model the task at the beginning can produce higher-quality results.
    - For later versions, it doesn't make any material response.
    - However, repeating the instructions at the end can make a difference.
  - Give it alternative options. For example: "Respond with 'not known' if the answer is not present".
    - *Tell me how to put it on Twitter.*
    - *Tell me how to put it on Twitter. Respond with 'not known' if the answer is not present.*
  - Before starting a new conversation, clear the chat history.
  - Avoid long questions – break them into multiple questions if possible.

## Apply Prompt Engineering

- System message
  - This is included at the beginning of a prompt.
  - It is used to give the model with content, instructions, or other information.
  - It can also describe personality, what the model should and shouldn't answer, and the format of model responses.
    - Prompt: *Describe a daisy.*
    - System: *Start with the word "Yo".*
    - System: *Speak like a pirate.*
    - System: *You are William Shakespeare.*
  - Define:
    - the specific tasks – who the users will be, what input they will provide, and what you expect to model to do with the information.
    - How it should complete the tasks – including additional tools if needed.
      - *To complete this task, you can [insert tools that the model can use and instructions to use]*

 

        - The scope and limitations. Say what should happen if the prompt is off-target.
            - *Do not perform actions that are not related to [task or topic name].*
        - The posture and tone.
        - The language and syntax of the output format
            - You use the [insert desired syntax] in your output
            - You will bold important words in your response.
- You can apply system messages for non-chat environments.
    - Analyse the sentiment from the speech, on a scale of 1 to 5, 5 being the highest. Explain why there is that rating.
        - Sentiment is a computer term, saying whether something is positive, negative, or neutral.

      *The GPT chat system is one of the best things I have ever seen.*
      *January has been very long. I wish that it was over. Thankfully, spring is around the corner.*
- Other suggestions are in the "Using templates" section:
    - Shakespeare writing assistance – How can I ask what the weather is like?
    - IRS tax chatbot – How much can I put into an IRA?
    - Marketing Writing Assistant - I want to have some marketing material for my new course "Develop generative AI solutions with Azure OpenAI Service"
    - Xbox customer support agent – How do I reboot my XBox?
    - Hiking recommendations chatbot – I am in Washington DC. I want to hike within 50 miles.
    - JSON formatter assistant –

      Please convert this list into JSON.
      ---
      Fruit: Apples, bananas, mango
      Vegetables: Carrots, potato, broccoli
- Assistant message
    - This is a combination of user prompts and assistant responses.
    - This can help describe future answers.
    - Describe a hamburger. Food.
    - Describe a Gin and Tonic. Alcohol.
    - Describe a printer. Electric device.
- Add clear syntax
    - You can use a --- or """ separator in between different sources of information or steps.
    - You can also use Markdown or XML language.
- Break down the task

  *Read this article.*
  ---
  *(Article)*
  ---
  *Extract the facts from the article, and put them into a bulleted list.*
- You can incorporate previous responses into your next prompt.

 

- Create a chain of thought prompting.
  *Which party has won the most votes in elections in England. Take it step-by-step. Present all the steps involved. Cite sources. Give reasoning. Share the final answer starting with "Answer is:"*
- Specify the output structure
  - Create the output in JSON [XML, Markup] format.
    *Analyse the following:*
    *---*
    *Fruit: Apples, bananas, mango*
    *Vegetables: Carrots, potato, broccoli*
    *---*
    *Create the output in the following format:*
    *[Type of food] - in hard brackets*
    *(Different foods) - in soft brackets, separated by commas.*
- You can change the temperature and top_p
  - Temperature can range from 0 to 1
    - A higher value will make the output more random and various. It can include fictional stories.
    - A lower value will make the output more focused and concrete.
  - Top_p (probabililty) controls this is another way.
  - Microsoft recommends altering one of these parameters at a time, not both.
- Define additional safety and behavioral guardrails
  *Do not provide any copyrighted content. Instead, politely refuse.*
  *Please give me the first three paragraphs of The Gruffolo*

## Generate code with Azure OpenAI Service

- GPT-3 series includes the Codex model series, which is proficient in over 10 languages, including C#, Go, JavaScript/TypeScript, Perl, PHP, Ruby, Shell, SQL and Swift.
- It can:
  - Create code from text.
  - Complete or rewrite existing code.
  - Suggest ways forward.
  - Comment code
- Provide examples for better results:
  *Create a list of 10 colors and 10 objects, and then combine them into 40 different combinations.*
  *combination = [ {"color": "Blue", "object": "car" } ]*
- Examples:
  *Ask for your name, and then say "Hello World", followed by their name.*

  *Create a list of 10 colors and 10 objects, and then combine them into 40 different combinations.*

  *I have an SQL table with the following fields:*

 

```
---
CREATE TABLE [Person].[Person](
        [BusinessEntityID] [int] NOT NULL,
        [PersonType] [nchar](2) NOT NULL,
        [NameStyle] [dbo].[NameStyle] NOT NULL,
        [Title] [nvarchar](8) NULL,
        [FirstName] [dbo].[Name] NOT NULL,
        [MiddleName] [dbo].[Name] NULL,
        [LastName] [dbo].[Name] NOT NULL)
---
Create a query in T-SQL which returns the full name of any person whose last name is Smith.
```

- Use section dividers, such as ---.
  - When working with Python, using “”” instead of ### may produce better results.

```
Explain the following code:
---
using System;

class Program
{
  static void Main(string[] args)
  {
    int n = 10; // Number of Fibonacci numbers to generate

    Console.WriteLine("Fibonacci Series:");
    for (int i = 0; i < n; i++)
    {
      Console.WriteLine(Fibonacci(i));
    }
  }

  static int Fibonacci(int n)
  {
    if (n <= 1)
    {
      return n;
    }
    else
    {
      return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
  }
}

Explain the following code:
```

 

```
“””
def fibonacci(n):
  if n <= 1:
    return n
  else:
    return fibonacci(n - 1) + fibonacci(n - 2)

n = 10  # Number of Fibonacci numbers to generate

print("Fibonacci Series:")
for i in range(n):
  print(fibonacci(i))
```

- Start with a comment, data or code:

```
# Write some code that returns the Fibonacci numbers.

# Table Person = PersonID, Title, FirstName, MiddleName, LastName
# Table Transactions = PersonID, TransactionDate, TransactionAmount
# Create a query which returns all people who have made a transaction
over $100.
```

- Explain an SQL query.
- Specify the programming language

```
# Python
# Write some code that returns the Fibonacci numbers.
```

- Write a prompt

```
# Python
# Write some code that returns the Fibonacci numbers.
def fibnumbers(n):
```

- However, writing your comments using section dividers inside functions may produce better results.

```
# Python
def fibnumbers(n):
"""
# Write some code that returns the Fibonacci numbers.
"""
```

- Ask for a unit test

```
# JavaScript
# Write some code that returns the Fibonacci numbers.
"""
def fibnumbers(n):
"""
# Unit test
fibResult =
```

- Be as precise as possible.

```
# Python
def fibnumbers(n):
"""
```

 

*# Write some code that returns the first 10 Fibonacci numbers in descending order.*
*"""*
*# Unit test*
*fibResult =*

- Check the code for errors.
- Convert between languages

  *Convert this from Python to C#*
  *# Python version*

  *# End*
  *# C# version*
- Specify a library, or suggest a library or API

  *<!-- Use Bootstrap to write a page with three buttons -->*
- Reduce the temperature
  - Setting it as 0, or 0.1 or 0.2, tends to give better results.
  - Higher temperatures can give you really random results.
- Limit the size of the query if necessary by reducing max_tokens.
- You can ask OpenAI to document code

  *Write code in Python which takes 2 strings "Hello" and "There", combines them with a space in the middle, and print "The answer is Hello There".*
  *Can you document this code.*
  *Can you give me an explanation for this code.*
  *Can you give me this code, using the string literals "Good" and "morning".*
  *Can you give me this code, using the string literals "Thank" and "you".*
  *Can you combine all of these code examples into one.*
- You can refactor the code.
  - This restructures the code without changing its function.
  - This can include simplifying complex code, removing redundant or unnecessary code, improving naming conventions, and breaking large functions into smaller, more modular functions.

    *Can you refactor this code.*

## Responsible Generative AI

- Azure OpenAI Service monitors content and behavior against its Code of Conduct, using:
  - Content Classification, for both images and language, in both prompts and completions (inputs and outputs).
  - Abuse Pattern Capture,
  - Human Review and Decision,
  - Notification and Action.
- It filters against the following categories:
  - Hate and fairness (non-discrimination),
  - Sexual references,
  - Self-harm and

 

    - Violence.
- It categories these against four severity levels:
    - Safe,
    - Low,
    - Medium and
    - High.
- You can configure in the Azure OpenAI Studio – Content filters to allow:
    - Only safe,
    - Safe and Low,
    - Safe, Low and Medium
    - No filters, if you have been approved for modified content filtering.
- It also protects against:
    - Jailbreak attempts (trying to bypass policies),
    - Protected material text (such as song lyrics, articles, recipes and select web content), and
    - Protected material code (source code).
- If your call is successful, then:
    - The HTTP response code will be 200 (“OK”), and
    - finish_reason will be either “stop” or “length”.
- If at least one of the responses was successful, and another was filtered, then:
    - The HTTP response code will be 200, and
    - finish_reason will be “content_filter”.
- If the responses were filtered, then:
    - The HTTP response code will be 400 (“Bad request”).
- If the call is still continuing, then
    - finish_reason will be null.

## ChatCompletions

- You create a ChatCompletion path using:
    - An OpenAI Endpoint (string),
    - An OpenAI key (string), and
    - An API version (string).
        - This follows the “YYYY-MM-DD” or “YYYY-MM-DD-preview” format.
        - It is taken from https://learn.microsoft.com/en-us/azure/ai-services/openai/reference#chat-completions
        - At the time of writing, the latest stable version was “2023-05-15” and the latest preview version was “2023-12-01-preview”.
- You then create a request body. For GPT-3.5-Turbo, this includes:
    - model (C#)/deployment (Python). The name of the model.
    - messages. This should include any previous messages. Each message needs:
        - role – either:
            - system: provides the behavior for the model.

 

                - user: the input for chat completions.
                - assistant: responses to system-instructed, user-prompted input
            - content – the text.
        - n – the number of choices to generate. The default is 1.
        - temperature – a number between 0 and 2. The default is 1.
            - Higher values will make the output more random.
            - Lower values will make it more focused and repeated.
        - top_p – an alternative to sampling with "temperature", called "nucleus sampling".
            - Microsoft recommends altering temperature or top_p, but not both.
        - Max_tokens – the maximum number of tokens to be used.
            - It is recommended that you use 300 or 500 for GPT 3.5.
            - The limit is 4,096 tokens.
- The response will include:
    - usage – how many tokens were used in the Prompt, the Completion, and the Total
    - choices – the response, which includes:
        - finish_reason
            - "stop" indicates that success.
            - "length" indicates that it has ended due to the maximum number of tokens having been used.
            - "content_filter" indicates that it has been stopped due to hate, sexual, violence or self-harm.
                - You can filter these for "low, medium, high", "medium, high" or "high".
                - "No filters" requires approval.
        - message, which includes content

## Versions of code

- C#
    - The current version at the time of writing is 1.0.0-beta.13
    - You should be specific as to which version you use, not just "the latest". Otherwise, you may find that future code breaks.
    - The transition from 1.0.0-beta.9 to 1.0.0-beta.10 was a "breaking change" – code running in .9 did not work in .10.
    - The code for DALL-E currently works in 1.0.0-beta.9.
- Python
    - The current version at the time of writing is 1.11.1. Version 1.0.0 was introduced in November 2023.
    - The transition from 0.28.1 to 1.x was a "breaking change".
    - The code for DALL-E currently works in 0.28.1

## C# Code (using version 1.0.0-beta.13)

- In the appropriate folder, run the command
  `dotnet new console -n Program`

 

- Go to the appropriate “Program” folder.
- Install the library
  `dotnet add package Azure.AI.OpenAI --`, followed by the version number
- To install a specific version, you can use:
  - `dotnet add package Azure.AI.OpenAI --prerelease`
    - This will install the latest version.
  - `dotnet add package Azure.AI.OpenAI --version=1.0.0-beta.9`
- Open the Program.cs file and fill in the following:

```
using Azure;
using Azure.AI.OpenAI;
```

- These allow you to use Azure and OpenAI.

```
string Key = "11a7d741fe494791824b61c8cc20bc19";
string Endpoint = "https://test240129.openai.azure.com/";
string ModelName = "Test240129";
```

- Fill in the details from the Azure Open AI Service.
  - The “key” is from the “Keys and Endpoint” section – Key 1.
  - The “endpoint” is from the “Keys and Endpoint” section – Endpoint.
  - The ModelName is the name of the model itself.

```
OpenAIClient client = new(new Uri(Endpoint), new AzureKeyCredential(Key));
```

- This creates the client, using the key and endpoint.

```
var chatCompletionsRequest = new ChatCompletionsOptions()
{
    DeploymentName = ModelName,
    MaxTokens = 200,
    Messages =
    {
        new ChatRequestSystemMessage("You are helpful."),
        new ChatRequestUserMessage("Write a slogan for a computer
programmer.")
    },
    Temperature=0.8f,
    ChoiceCount=2,
};
```

- This creates a ChatCompletionsRequest, and uses the client previously defined to create a ChatCompletion.
- It passes in:
  - The model name,
  - The maximum number of tokens to be used,
  - The temperature (from 0 to 1),
  - The messages, which include:
    - A system message and
    - A user message.

```
ChatCompletions ChatCompletionsResponse =
client.GetChatCompletions(chatCompletionsRequest);

Console.WriteLine(ChatCompletionsResponse.Choices[0].Message.Content);
Console.WriteLine(ChatCompletionsResponse.Choices[1].Message.Content);
```

- It saves it in the variable “ChatCompletionsResponse”.

 

- To run the program, use:
  `dotnet run Program.cs`

## Python Code

- Create a .env file with:

```
AZURE_OPENAI_KEY=
AZURE_OPENAI_ENDPOINT=
AZURE_API_VERSION=
AZURE_OPENAI_MODEL=
```

- Fill in the settings, and save the file.
- Create a program.py file.
- Install the library:
  `pip install openai`
  - If you want a specific version of openai, then you would add that version to the end of the command, after two equal signs:

  `pip install openai==0.28.1`
- To check which version of OpenAI you have, you would use:
  `pip show openai`
- Fill in the details from the Azure Open AI Service.
  - The AZURE_OPENAI_KEY is from the “Keys and Endpoint” section – Key 1.
  - The AZURE_OPENAI_ENDPOINT is from the “Keys and Endpoint” section – Endpoint.
  - The AZURE_API_VERSION is taken from https://learn.microsoft.com/en-us/azure/ai-services/openai/reference#chat-completions
  - The AZURE_OPENAI_MODEL is the name of the model itself.

```
import os
from dotenv import load_dotenv
from openai import AzureOpenAI

load_dotenv()
```

- The first, second and last lines are used to get the environmental variables.
- The third line imports the Azure Open AI class.

```
client = AzureOpenAI(api_key = os.getenv("AZURE_OPENAI_KEY"),
                     api_version = os.getenv("AZURE_API_VERSION"),
                     azure_endpoint = os.getenv("AZURE_OPENAI_ENDPOINT")
                     )

model_name=os.getenv("AZURE_OPENAI_MODEL")
```

- This creates the client, using the key, endpoint and version.
- It also retrieves the model name.

```
response = client.chat.completions.create(
              model=model_name,
              max_tokens=200,
              temperature=0.8,
              n=1,
              messages=[
                        {"role": "system", "content": "You are being
helpful."},
```

 

```
                        {"role": "user", "content": "Write a slogan for a
computer programmer."}
                    ])
```

- This uses the client previously defined to create a ChatCompletion.
- It passes in:
    - The model name,
    - The maximum number of tokens to be used,
    - The temperature (from 0 to 1),
    - The messages, which include:
        - A system message and
        - A user message.
- It saves it in the variable “response”.

```
print(response.choices[0].message.content)
```

- It uses the “response” variable to retrieve the content of the response.
- To run it, enter in the terminal:
    - python program.py
        - Or whatever the name of your program file is.

# Generate images with Azure OpenAI Service

- In the settings section of the DALL-E Playground for DALL-E 2, you can select:
    - The number of images (1 to 3), and
    - The resolution of the images: 256x256, 512x512 and 1024x1024 (default).
- In the settings section of the DALL-E Playground for DALL-E 3, you can select:
    - Image Quality (standard or hd), and
    - The resolution of the images: 256x256, 512x512 and 1024x1024 (default) and 1024x1792.
    - Image style: vivid and natural

## C# code (using version 1.0.0-beta.9)

- In the appropriate folder, run the command
  `dotnet new console -n Program`
- Open the Program.cs file.
- Install the library. The code seems to have broken in 1.0.0-beta.10, so use 1.0.0-beta.9
  `dotnet add package Azure.AI.OpenAI --version=1.0.0-beta.9`

## Python code (using version 0.28.1)

- You can create code in Python by clicking on “show code”.
    - However, the latest release of the OpenAI Python library does not, at the time of writing, support DALL-E 2 when used with Azure OpenAI. Therefore, you need to install a version of Azure OpenAI less than 1.
- Create a program.py file.
- Install the library:
  `pip install openai==0.28.1`
- Fill in the details from the Azure Open AI Service.

 

- Copy the code from DALL-E into program.py
  `Add print(image_url) to the end`
- Run the program using
  `python program.py`

## Use your own data with Azure OpenAI Service

- Use your own data allows or requires the chat model to use your data in answering prompts.
  - If your data is up-to-date or contains specialised data, then this can improve the responses.
- It uses Azure Blob Storage to store your data, and Azure AI Search to index it.
  - The regions you can use are limited. At the time of writing, there were 16 regions where your Azure OpenAI resource could be:
    - Canada East, East US, East US 2, North Central US, South Central US, West US
    - France Central, Norway East, Sweden Central, Switzerland North, UK South, West Europe
    - South India
    - Brazil South
    - Japan East, Australia East
- You can use the following chat models:
  - gpt-35-turbo (0301),
  - gpt-35-turbo-16k,
  - gpt-4, and
  - gpt-4-32k.
- To add your own data in the Azure OpenAI Studio, click on “Add your data (preview) – “+ Add a data source”.
- You can use the data sources:
  - Azure AI Search
  - Azure Blob Storage
  - Azure Cosmos DB for MongoDB vCore
  - URL/web address
  - Upload files.
    - You can use Text files, Markdown files, HTML files, Microsoft Word files, Microsoft PowerPoint files, and pdfs.
      - The best citation titles are from Markdown “.md” files.
      - Text contents are extracts from PDFs as a pre-processing step.
    - There is a limit of 16 megabytes per upload (but you can do multiple uploads).
- In this example, we will upload PDF files, including a “Microsoft Copilot Studio” document, which states “Microsoft Copilot Studio is the new name for Power Virtual Agents.”
  - This is the only reference to Power Virtual Agents in this document.
- You will need to:
  - Create an azure Blob storage resource,

 

    - Create an Azure AI Search resource, and
    - Turn on Cross-origin resource sharing (CORS), to allow Azure OpenAI to access the storage account and Azure AI Search.
    - Check "I acknowledge that connecting to an Azure AI Search account will incur usage to my account".
        - Pricing the Basic version is about $80 per month.
        - The Free tier is not supported for "Use your own data".
- If the data source has not been attached to the Chat playground, you can:
    - Click on "Add your data (preview) – "+ Add a data source".
    - Select the data source as "Azure AI Search".
    - Select the relevant Subscription, Azure AI Search service and Azure AI Search Index.
    - Check "I acknowledge that connecting to an Azure AI Search account will incur usage to my account".
- You should then use Keyword search.
    - It performs fast and flexible querying.
    - Queries need to be the language of the uploaded documents.
    - Semantic and Vector searches are more specialised. They need a Basic or higher model pricing, and will cost more.
- If you are using your own index, then you can map the fields to Content Data fields.
- It then ingests and indexes your documents.
- In the Advanced settings, you can select:
    - "Limit responses to your data content" – this is checked by default.
    - "Strictness (1-5)" – a higher figure means that more of your documents can be filtered as being irrelevant for your query. The default is 3.
    - "Retrieved documents (3-20)" – this is the number of top-scoring documents from your data to be used to generate responses. The default is 5.
- For the code, you will need:
    - The SearchEndpoint – this can be found by going to Azure AI services – AI Search – [click on the search index] – Url.
    - The SearchKey – this can be found by going to Keys in that search index.
    - The SearchIndex – this is the name of the index. In the below, I will use "powerapps" as the name of the index.
- In this example, we will ask it:
  `"Talk to me about Power Virtual Agents."`
- It will state that: "Power Virtual Agents is now known as Microsoft Copilot Studio[doc1]."

## C# code (using version 1.0.0-beta.13)

```
string SearchEndpoint = "";
string SearchKey = "";
string SearchIndex = "";

AzureCognitiveSearchChatExtensionConfiguration ownData = new()
{
```

 

```
    SearchEndpoint = new Uri(SearchEndpoint),
    Authentication = new OnYourDataApiKeyAuthenticationOptions(SearchKey),
    IndexName = SearchIndex
};
```

- This adds the configuration of the Azure Search Endpoint into a variable called “ownData”.
- In the chatCompletionsRequest, you will use the “ownData”:

```
    AzureExtensionsOptions = new AzureChatExtensionsOptions()
    {
        Extensions = {ownData}
    }
```

## Python code

- Add into the .env file:

```
SEARCH_ENDPOINT =
SEARCH_KEY =
SEARCH_INDEX =
```

- Reference them in the main code:

```
azure_search_endpoint = os.getenv("SEARCH_ENDPOINT")
azure_search_key = os.getenv("SEARCH_KEY")
azure_search_index = os.getenv("SEARCH_INDEX")
```

- Add a changed base URL:

```
client = AzureOpenAI(
            base_url=f"{azure_oai_endpoint}/openai/deployments/{azure_oai_model}/extensions
",
            api_key=azure_oai_key,
            api_version="2023-09-01-preview")
```

- Create the dataSources, using the Search references, into a dictionary item, use key-value pairs:

```
extra_config = dict(dataSources = [{
                                    "type": "AzureCognitiveSearch",
                                    "parameters": {
                                           "endpoint": os.getenv("SEARCH_ENDPOINT"),
                                           "key": os.getenv("SEARCH_KEY"),
                                           "indexName": os.getenv("SEARCH_INDEX"),
                                       }
                                   }]
                    )
```

- Incorporate this new dictionary in the ChatCompletions request:

```
                        ,
                extra_body= extra_config
```

 