

How to develop Intelligent Applications with .NET 9

Author: Luis Coco Enríquez



Personal profile

Senior .Net Software Engineer

AI services, Aspire .NET 9, Blazor, MAUI-Blazor

Azure, AWS, Google Cloud Services



<https://www.linkedin.com/in/luis-coco-enriquez-44a28a29/>



CERTIFICATION

Microsoft Certified: Azure Solutions Architect Expert

Expires on November 29, 2023 at 12:59 AM (UTC -0-1:00) • Earned on November 28, 2022



CERTIFICATION

Microsoft Certified: Azure Administrator Associate

Expires on September 19, 2023 at 1:59 AM (UTC -0-2:00) • Earned on September 18, 2021



CERTIFICATION

Microsoft Certified: Azure Developer Associate

Expires on November 2, 2023 at 12:59 AM (UTC -0-1:00) • Earned on November 1, 2021



CERTIFICATION

Microsoft Certified: Azure Fundamentals



CERTIFICATION

Microsoft Certified: Azure AI Fundamentals



CERTIFICATION

Microsoft Certified: Azure Data Fundamentals

Agenda

Topic



ChatGPT and ChatGPT APIs



OpenAI for .NET



Azure OpenAI and Azure.AI.OpenAI library



Semantic Kernel

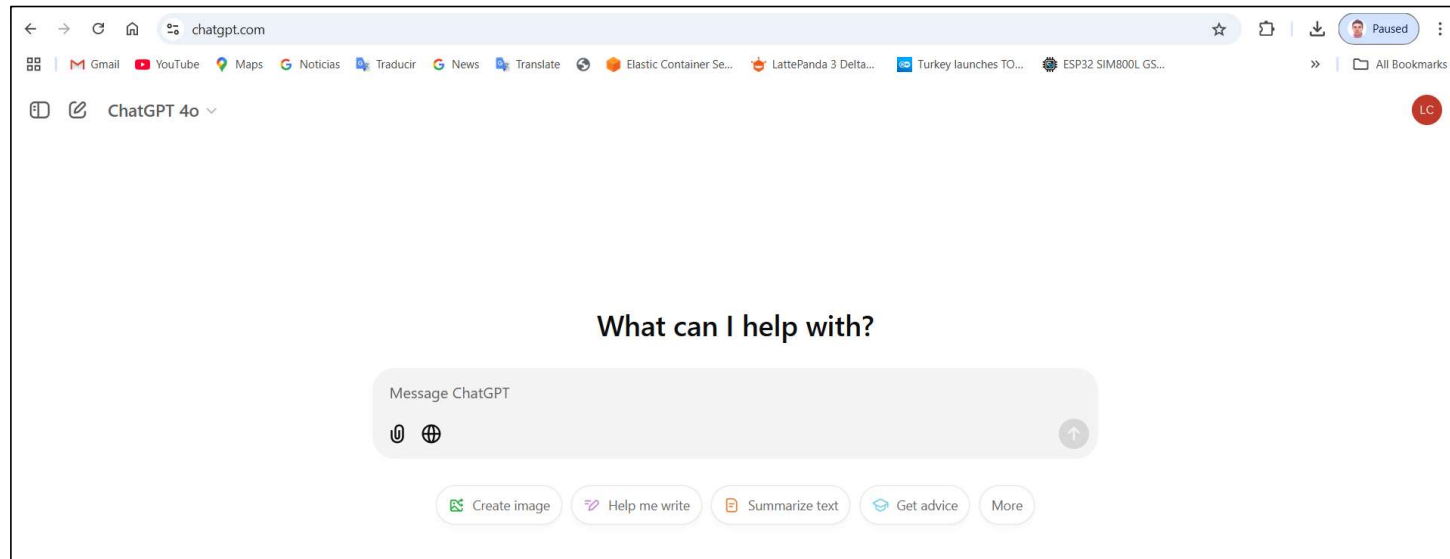
Microsoft.Extensions.AI

What is ChatGPT? <https://en.wikipedia.org/wiki/ChatGPT>

ChatGPT is a **Generative** Artificial Intelligence (AI) **chatbot** developed by **OpenAI** and launched in **November 2022**

ChatGPT can generate human-like conversational responses

ChatGPT is built on **OpenAI**'s proprietary series of **Generative Pre-trained Transformer (GPT) Large Language Models (LLMs)**, and is fine-tuned for conversational applications using a combination of supervised learning and reinforcement learning from human feedback. Successive user prompts and replies are considered at each conversation stage as context



<https://chatgpt.com/>

What is GPT (Generative Pre-Trained Transformer)?

A generative pre-trained transformer (**GPT**) is a type of large language model (**LLM**) and a prominent framework for generative artificial intelligence.

It is an **Artificial Neural Network** that is used in **natural language processing by machines**.

It is based on the **Transformer Deep Learning Architecture**, pre-trained on large data sets of unlabeled text, and able to generate novel human-like content.

As of 2023, most **LLMs** had these characteristics and are sometimes referred to broadly as **GPTs**.

What is aTransformer (Deep Learning Architecture)?

A **Transformer** is a deep learning architecture developed by researchers at **Google** and based on the multi-head attention mechanism, proposed in the **2017** paper "**Attention Is All You Need**".

Text is **converted** to **numerical** representations called **tokens**.

Each **token** is **converted** into a **vector** via lookup from a word embedding table.

At each **layer**, each **token** is then **contextualized** within the scope of the context window **with other** (unmasked) **tokens** via a parallel multi-head attention mechanism, allowing the **signal** for **key tokens** to be **amplified** and less important tokens to be diminished.

[https://en.wikipedia.org/wiki/Transformer_\(deep_learning_architecture\)](https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture))

What is ChatGPT OpenAI API?

The screenshot shows the OpenAI Platform API reference page for the 'Create chat completion' endpoint. The page is titled 'Completions' and features a sidebar with navigation links. The main content area is titled 'Create chat completion' and includes a POST request to `https://api.openai.com/v1/chat/completions`. It describes the endpoint's purpose: 'Creates a model response for the given chat conversation. Learn more in the [text generation](#), [vision](#), and [audio](#) guides.' The 'Request body' section details the required parameters: **messages** (array, required), **model** (string, required), and **store** (boolean or null, optional, defaults to false). The 'Example request' section shows a curl command and a JSON payload. The 'Response' section shows a JSON object representing the chat completion.

OpenAI Platform

Search CTRL K

ENDPOINTS

- Audio
- Chat
 - Create chat completion
 - The chat completion object
 - The chat completion chunk object
- Embeddings
- Fine-tuning
- Batch
- Files
- Uploads
- Images
- Models
- Cookbook
- Forum
- Help

Completions

Create chat completion

POST `https://api.openai.com/v1/chat/completions`

Creates a model response for the given chat conversation. Learn more in the [text generation](#), [vision](#), and [audio](#) guides.

Request body

messages array **Required**

A list of messages comprising the conversation so far. Depending on the [model](#) you use, different message types (modalities) are supported, like [text](#), [images](#), and [audio](#).

▼ Show possible types

model string **Required**

ID of the model to use. See the [model endpoint compatibility](#) table for details on which models work with the Chat API.

store boolean or null **Optional** Defaults to false

Whether or not to store the output of this chat completion request for use in our [model distillation](#) or [evals](#) products.

Default Image input Streaming Functions Logprobs

Example request `gpt-4o` `curl`

```
1 curl https://api.openai.com/v1/chat/completions \
2   -H "Content-Type: application/json" \
3   -H "Authorization: Bearer $OPENAI_API_KEY" \
4   -d '{
5     "model": "gpt-4o",
6     "messages": [
7       {
8         "role": "system",
9         "content": "You are a helpful assistant."
10      },
11      {
12        "role": "user"
```

Response

```
1 {
2   "id": "chatcmpl-123",
3   "object": "chat.completion",
4   "created": 1677652288,
5   "model": "gpt-4o-mini",
```

What is ChatGPT OpenAI API?

The screenshot displays a REST client interface with a POST request to `https://api.openai.com/v1/chat/completions`. The request body is a JSON object with a system message and a user message. The response is a JSON object containing a completion for the user's message.

```
1 {
2   "model": "gpt-4",
3   "messages": [
4     {
5       "role": "system",
6       "content": "You are a helpful assistant."
7     },
8     {
9       "role": "user",
10      "content": "Hello!"
11    }
12  ]
13 }
```

Status: 200 OK Time: 1178 ms Size: 1.66 KB

```
1 {
2   "id": "chatcmpl-8kvR0umdU1KxUTBzVSDdysjciMyDe",
3   "object": "chat.completion",
4   "created": 1706194558,
5   "model": "gpt-4-0613",
6   "choices": [
7     {
8       "index": 0,
9       "message": {
10        "role": "assistant",
11        "content": "Hello! How can I assist you today?"
12      }
13    }
14  ]
15 }
```

```
//Set the authorization header with the API
key
_httpClient.DefaultRequestHeaders
.Add("Authorization", $"Bearer {apiKey}");
```

```
//Make the POST request using the
configured HttpClient var response =
await _httpClient.PostAsync(apiUrl,
requestContent);
```

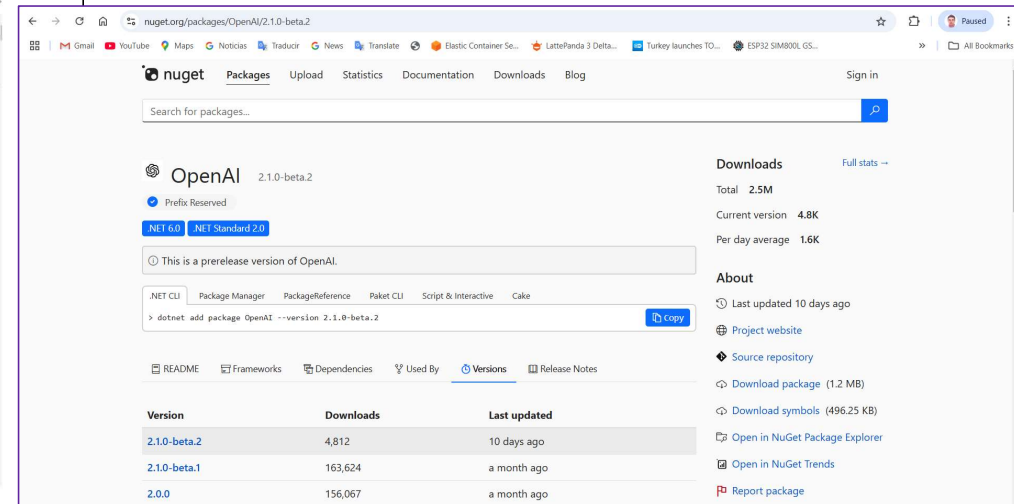
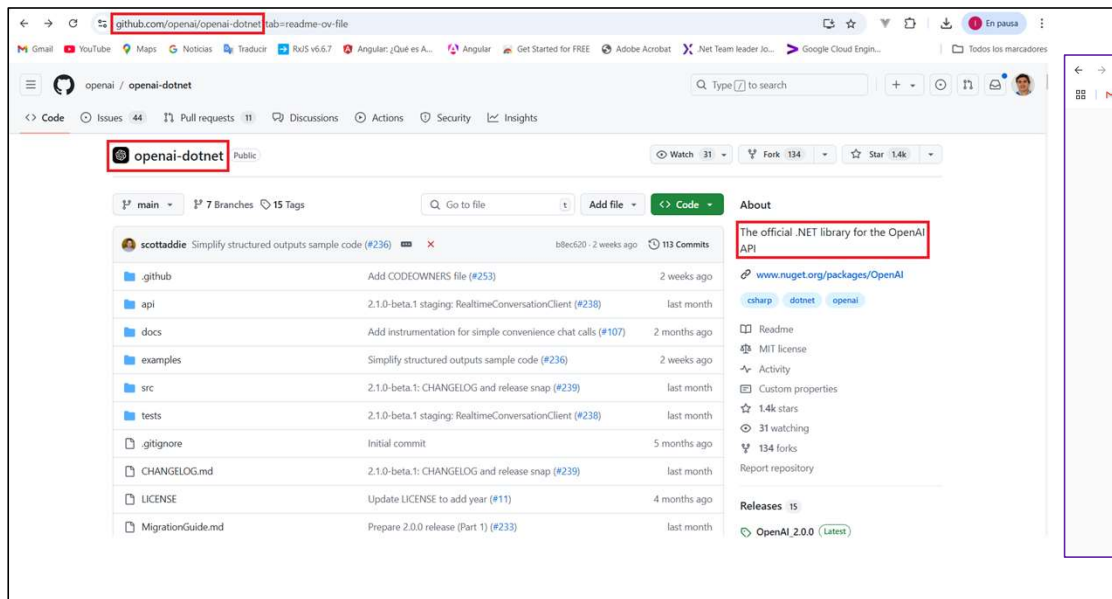
https://github.com/luiscoco/ChatGPT_OpenAI-Sample1-QuickStart

https://github.com/luiscoco/ChatGPT_OpenAI-Sample2-ChatGPT-WebAPI



OpenAI for .NET

The official .NET library for the OpenAI API



<https://github.com/openai/openai-dotnet>

<https://www.nuget.org/packages/OpenAI>

<https://github.com/openai/openai-openapi/blob/master/openapi.yaml>



OpenAI for .NET

The official .NET library for the OpenAI API

```
using NUnit.Framework;
using OpenAI.Chat;
using System;

namespace OpenAI.Examples;

public partial class ChatExamples
{
    [Test]
    public void Example01_SimpleChat()
    {
        ChatClient client = new(model: "gpt-4o", apiKey: Environment.GetEnvironmentVariable("OPENAI_API_KEY"));

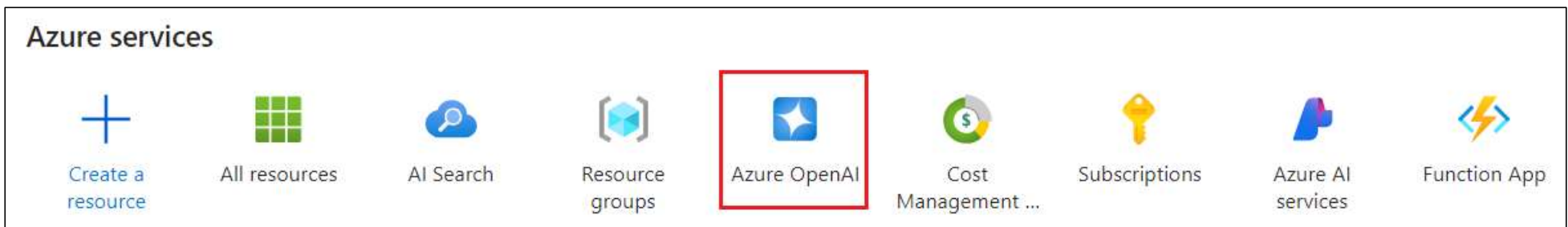
        ChatCompletion completion = client.CompleteChat("Say 'this is a test.'");

        Console.WriteLine($"[ASSISTANT]: {completion.Content[0].Text}");
    }
}
```



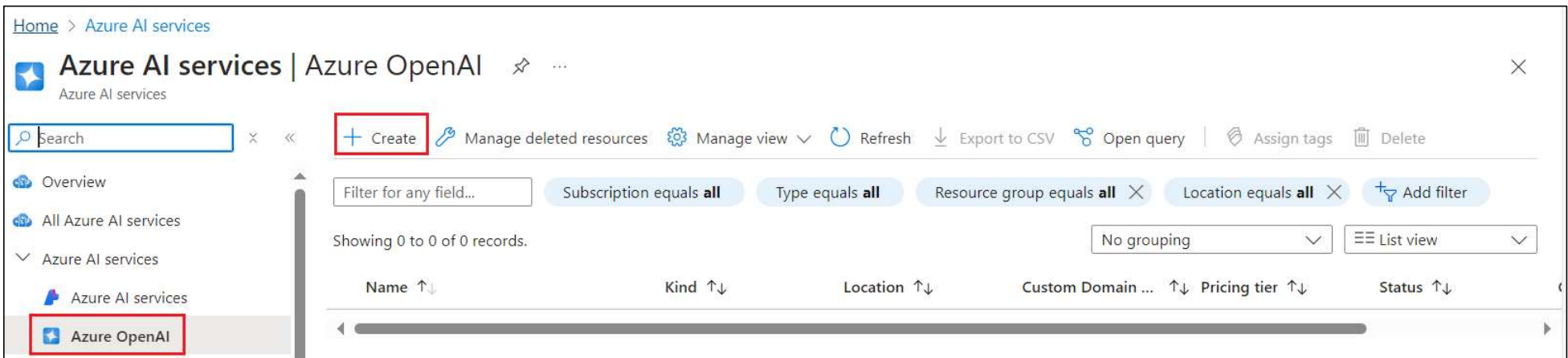
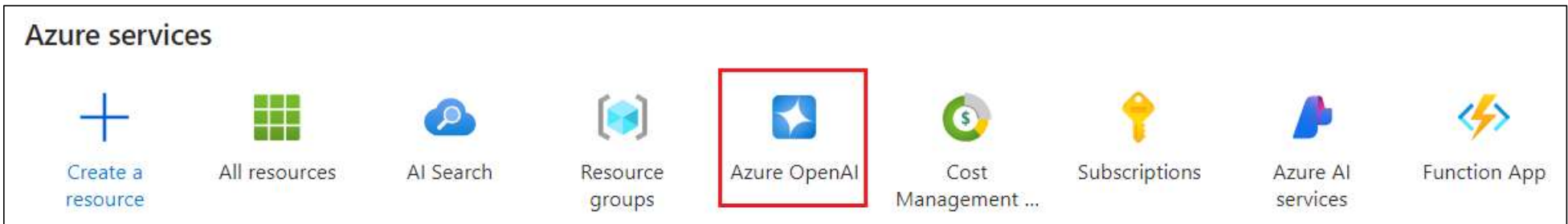
What is Azure OpenAI?

Azure OpenAI Service provides **REST API** access to OpenAI's powerful **LLMs** Large Language Models including o1-preview, o1-mini, GPT-4o, GPT-4o mini, GPT-4 Turbo with Vision, GPT-4, GPT-3.5-Turbo, and Embeddings model series. These models can be easily adapted to your specific task including but not limited to **content generation**, **summarization**, **image understanding**, **semantic search**, and **natural language to code translation**. Users can access the service through **REST APIs**, **Python** or **C# SDK**, or in the **Azure AI Studio**.





Azure OpenAI: How to provision in Azure Portal





Azure OpenAI: How to provision in Azure Portal

Home > Azure AI services | Azure OpenAI >

Create Azure OpenAI

1 Basics 2 Network 3 Tags 4 Review + submit

Azure OpenAI Service provides access to OpenAI's powerful language models, including all the latest OpenAI models. These models can be easily adapted to your specific tasks, including but not limited to content generation, summarization, image understanding, semantic search, and natural language to code translation. Top use cases include Call Centers, Virtual Assistants, Accessibility, Content Generation, and Code Development. The service also features the Assistants API, Fine Tuning capabilities and many ways to connect your data to the service for conversational experiences. The service can be scaled through Standard (tokens) and Provisioned (PTUs) deployment types.

[Learn more](#)

Project Details

Subscription * ⓘ

Resource group * ⓘ [Create new](#)

Instance Details

Region ⓘ

Name * ⓘ ✓

Pricing tier * ⓘ

[View full pricing details](#)

Home > Microsoft.CognitiveServicesOpenAI-20241029171327 | Overview > luxoftaisample

luxoftaisample | Model deployments

Azure OpenAI

Search

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Resource Management
- Keys and Endpoint
- Model deployments**

Model deployments features have moved to Azure OpenAI Studio.

Manage Deployments

Azure OpenAI: How to provision in Azure Portal



Azure OpenAI Studio / Model deployments

Model deployments

Deploy a model with your private API key and an endpoint URI (Uniform Resource Identifier).

Model deployments App deployments

+ Deploy model Refresh Edit Delete Open in playground Reset view

Deploy base model

Deploy fine-tuned model

Deploy model from Azure ML

Model name	Model version	State
------------	---------------	-------

No deployments to display

Select a model

Choose a model to create a new deployment. For flows and other resources, create a deployment from their respective list. [Go to model catalog.](#)

Models: 20 Inference tasks Show description

Search

- o1-mini Chat completion
- gpt-4o-mini Chat completion
- gpt-4o Chat completion
- gpt-4-32k Chat completion
- gpt-35-turbo-instruct Chat completion
- gpt-35-turbo-16k Chat completion
- dall-e-3

3. **Content innovation:** Use GPT-4o's generative capabilities to create engaging and diverse content formats, catering to a broad range of consumer preferences.

Note: updated version 2024-08-06

GPT-4o has been released under a new version `2024-08-06` which brings new functionalities and a larger output size (from 4,096 to 16,384).

In addition to features such as:

- Text, image processing
- JSON Mode
- parallel function calling
- Enhanced accuracy and responsiveness
- Parity with English text and coding tasks compared to GPT-4 Turbo with Vision
- Superior performance in non-English languages and in vision tasks
- Support for enhancements

This new version has been trained to support complex structured outputs.

Resources

- "Hello GPT-4o" (OpenAI announcement)
- Introducing GPT-4o: OpenAI's new flagship multimodal model now in preview

Confirm Cancel



Azure OpenAI Service models

Models	Description
o1-preview and o1-mini	Limited access models, specifically designed to tackle reasoning and problem-solving tasks with increased focus and capability.
GPT-4o & GPT-4o mini & GPT-4 Turbo	The latest most capable Azure OpenAI models with multimodal versions, which can accept both text and images as input.
GPT-4o audio	A GPT-4o model that supports low-latency, "speech in, speech out" conversational interactions.
GPT-4	A set of models that improve on GPT-3.5 and can understand and generate natural language and code.
GPT-3.5	A set of models that improve on GPT-3 and can understand and generate natural language and code.
Embeddings	A set of models that can convert text into numerical vector form to facilitate text similarity.
DALL-E	A series of models that can generate original images from natural language.
Whisper	A series of models in preview that can transcribe and translate speech to text.
Text to speech (Preview)	A series of models in preview that can synthesize text to speech.

<https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/models>



Azure OpenAI: How to provision in Azure Portal

Deploy model gpt-4o

Deployment name *

gpt-4o

Deployment type

Standard

Standard: Pay Per API call, with lower rate limits. Adheres to Azure data residency promises. Best for intermittent workloads with low to medium volume. [Learn more about Standard deployment type](#)

Deployment details

Model version

2024-05-13 (Default)

Resource location

West US

8K tokens per minute quota available for your deployment

Tokens per Minute Rate Limit

8K

Corresponding requests per minute (RPM) = 48

Enable dynamic quota

Enabled

A new Azure OpenAI resource will be created for your deployment

gpt-4o

Details

Metrics

Risks & Safety

Open in playground

Edit

Delete

Deployment info

Name

gpt-4o

Provisioning state

Succeeded

Deployment type

Standard

Created on

2024-10-29T16:28:58.2777317Z

Created by

luiscocoenriquez@hotmail.com

Modified on

Oct 29, 2024 5:28 PM

Modified by

luiscocoenriquez@hotmail.com

Version update policy

Once a new default version is available

Rate limit (Tokens per minute)

8,000

Rate limit (Requests per minute)

48

Model name

gpt-4o

Model version

2024-05-13

Life cycle status

GenerallyAvailable

Date created

May 13, 2024 2:00 AM

Date updated

Aug 21, 2024 2:00 AM

Model retirement date

May 20, 2025 2:00 AM

Endpoint

Target URI

https://luisc-m2unuymc-westus.openai.azure.com/openai/deployments/gpt-4o/chat/c...

Key

.....

Monitoring & safety

Content filter

DefaultV2

Useful links for application development

[Code sample repository](#)

[Tutorial](#)

Luxoft
A DXC Technology Company

https://github.com/luiscoco/Azure_OpenAI_How-to-deploy-GPT-4o

© 2024 Luxoft, A DXC Technology Company. All rights reserved.

November 15, 2024 16



Azure OpenAI Nuget Packages

<https://github.com/Azure/azure-sdk-for-net/tree/main/sdk/openai>

Name	Last commit message	Last commit date
..		
Azure.AI.OpenAI.Assistants	Update AutoRest C# version to 3.0.0-beta.20240926.1 (#46331)	last month
Azure.AI.OpenAI	fix for 46109 (#46401)	last month
tools/TestFramework	Update System.Memory.Data (#46134)	28 days ago
.gitignore	Azure OpenAI: 2.0.0-beta.3 updates (#45479)	2 months ago
ci.yml	Skip project reference tests for openai (#44896)	3 months ago
tests.yml	Skip project reference tests for openai (#44896)	3 months ago

<https://www.nuget.org/packages/Azure.AI.OpenAI/2.1.0-beta.1>

<https://www.nuget.org/packages/Azure.AI.OpenAI.Assistants/1.0.0-beta.4>

<https://www.nuget.org/packages/Aspire.Azure.AI.OpenAI/9.0.0-preview.4.24511.1>

<https://www.nuget.org/packages/Aspire.Hosting.Azure.CognitiveServices/9.0.0-rc.1.24511.1>

Azure OpenAI: How to provision the service with C# SDK



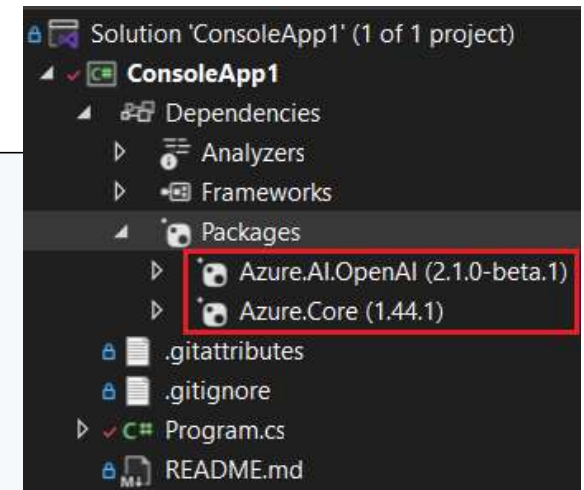
AzureOpenAIClient (Endpoint, Key)

GetChatClient (DeploymentName)

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

class Program
{
    static async Task Main(string[] args)
    {
        var endpoint = new Uri("https://cocoe-m2ae1t7j-swedencentral.openai.azure.com/");
        var credentials = new AzureKeyCredential("1c1ad980b9ae425eb7ae14581fea4fe4");
        var deploymentName = "gpt-4o"; // Ensure this matches your Azure OpenAI deployment name

        var openAIClient = new AzureOpenAIClient(endpoint, credentials);
        var chatClient = openAIClient.GetChatClient(deploymentName);
    }
}
```





Azure OpenAI: ChatCompletion with gpt-4o model

AI **Completion** Service: You input some text as a prompt. The model generates the completion and attempts to match your context or pattern. Suppose you provide the prompt "As Descartes said, I think, therefore" to the API. For this prompt, Azure OpenAI returns the completion endpoint "I am" with high probability.

```
using Azure;
using Azure.AI.OpenAI;
using static System.Environment;

string endpoint = GetEnvironmentVariable("AZURE_OPENAI_ENDPOINT");
string key = GetEnvironmentVariable("AZURE_OPENAI_API_KEY");

AzureOpenAIClient azureClient = new(
    new Uri(endpoint),
    new AzureKeyCredential(key));

// This must match the custom deployment name you chose for your model
ChatClient chatClient = azureClient.GetChatClient("gpt-35-turbo");

ChatCompletion completion = chatClient.CompleteChat(
[
    new SystemChatMessage("You are a helpful assistant that talks like a pirate."),
    new UserChatMessage("Does Azure OpenAI support customer managed keys?"),
    new AssistantChatMessage("Yes, customer managed keys are supported by Azure OpenAI"),
    new UserChatMessage("Do other Azure AI services support this too?")
]);

Console.WriteLine($"{completion.Role}: {completion.Content[0].Text}");
```

```
AzureOpenAIClient azureClient = new(
    new Uri("https://your-azure-openai-resource.com"),
    new DefaultAzureCredential());
ChatClient chatClient = azureClient.GetChatClient("my-gpt-35-turbo-deployment");

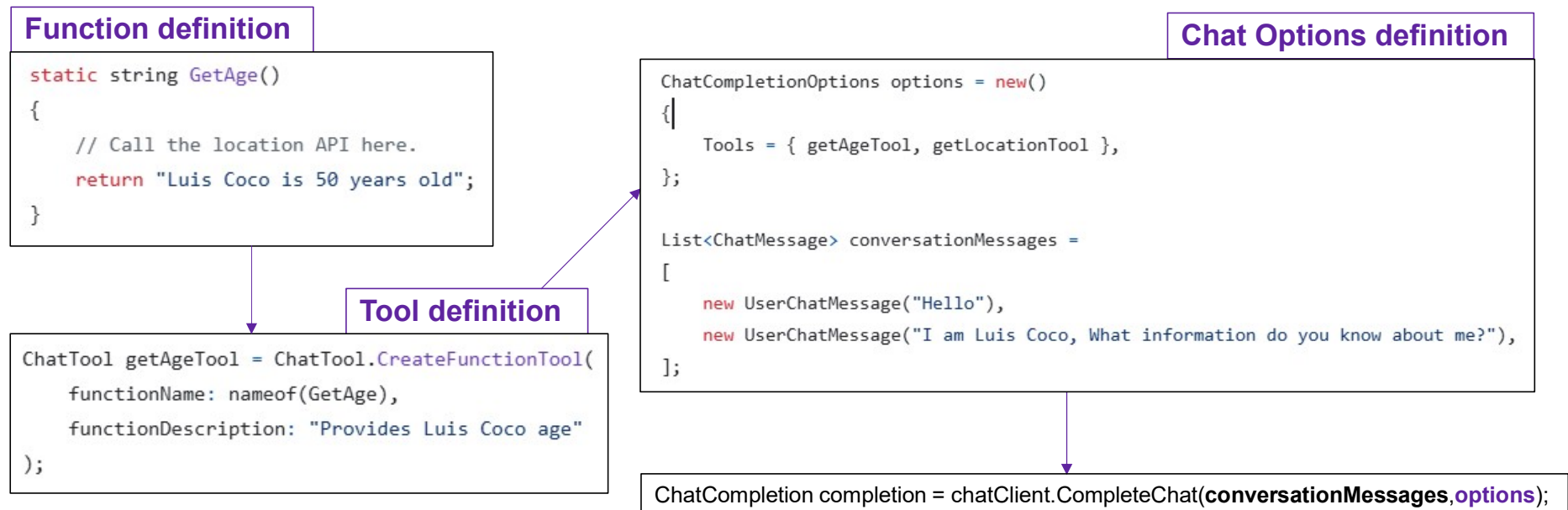
CollectionResult<StreamingChatCompletionUpdate> completionUpdates = chatClient.CompleteChatStreaming(
[
    new SystemChatMessage("You are a helpful assistant that talks like a pirate."),
    new UserChatMessage("Hi, can you help me?"),
    new AssistantChatMessage("Arrr! Of course, me hearty! What can I do for ye?"),
    new UserChatMessage("What's the best way to train a parrot?"),
]);

foreach (StreamingChatCompletionUpdate completionUpdate in completionUpdates)
{
    foreach (ChatMessageContentPart contentPart in completionUpdate.ContentUpdate)
    {
        Console.WriteLine(contentPart.Text);
    }
}
```



Azure OpenAI: Use Chat Tools (call Functions)

Tools extend chat completions by allowing an assistant to invoke defined functions and other capabilities in the process of fulfilling a chat completions request. To use chat tools, start by defining a function tool.

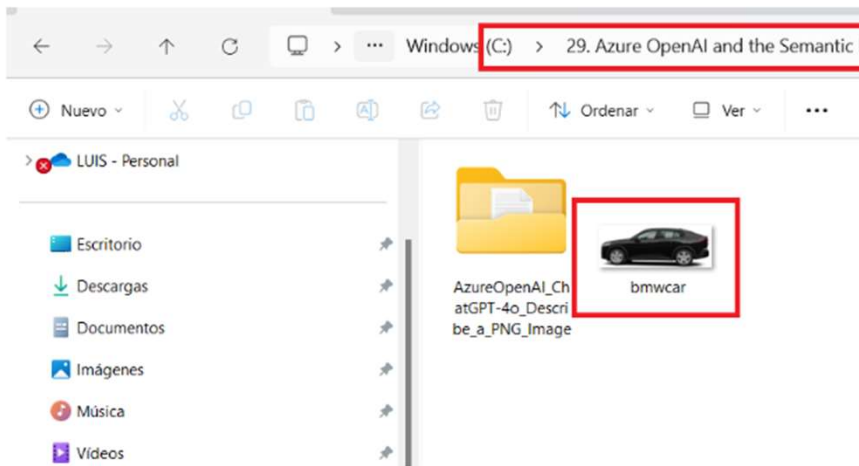


https://github.com/luisccoco/Azure_OpenAI-Use_Chats_Tools-Function_Call
https://github.com/luisccoco/Azure_OpenAI-Use_Chats_Tools-Function_Call-version2



Azure OpenAI: Describe and Image with gpt-4o model

We can provide an image in the **prompt** when invoking the **gpt-4o** model with the **CompleteChat** function



```
List<ChatMessage> messages = new List<ChatMessage>
{
    new UserChatMessage(
        ChatMessageContentPart.CreateTextPart("Please describe the following image:"),
        ChatMessageContentPart.CreateImagePart(imageData, "image/png"))
};

// Send the chat completion request
ChatCompletion chatCompletion;

try
{
    chatCompletion = await chatClient.CompleteChatAsync(messages);
}
```

We run the application and see the results

```
[ASSISTANT]:
The image shows a side view of a black BMW vehicle. The vehicle appears to be a compact SUV or crossover. It features a sleek and modern design with a slightly sloped roofline towards the rear, giving it an aerodynamic look. The windows are tinted, and the vehicle has five doors, including the rear hatch. The wheels have a contemporary five-spoke design, and the side mirrors are integrated smoothly into the overall design. The vehicle also has a noticeable character line running along the side, which adds to its sporty appearance.
```




Azure OpenAI: Create an Image with DALL-E model

OpenAI's **DALL-E** models generate images based on user-provided text prompts

```
ImageClient chatClient = openAIClient.GetImageClient("dall-e-3");

var imageGeneration = await chatClient.GenerateImageAsync(
    "a BMW iX2 black",
    new ImageGenerationOptions()
    {
        Size = GeneratedImageSize.W1024xH1024
    }
);
```

Consola de depuración de Mi: x + -

```
https://dalleproduse.blob.core.windows.net/private/images/a5a9aced-a50f-46b0-8984-6cfda214fdac/generated_00.png?se=2024-10-16T15%3A12%3A02Z&sig=D0eUwQYp4WVjwCtgzUjY%2BjFscrsqyd2Vzls1%2BuZJd%2FM%3D&ske=2024-10-21T15%3A11%3A13Z&skoid=09ba021e-c417-441c-b203-c81e5dcd7b7f&sks=b&skt=2024-10-14T15%3A11%3A13Z&sktid=33e01921-4d64-4f8c-a055-5bdaffd5e33d&skv=2020-10-02&sp=r&spr=https&sr=b&sv=2020-10-02
```



Azure OpenAI: Speech to Text with Whisper model

The **Whisper** model can **transcribe** human speech in numerous languages, and it can also **translate** other languages into English.

```
using Azure;
using Azure.AI.OpenAI;
using Azure.Identity; // Required for Passwordless auth

var endpoint = new Uri("https://whisperluismodel.openai.azure.com/");
var credentials = new AzureKeyCredential("58b40a57c13d475184e472aa58dd392d");

var deploymentName = "whisper"; // Default deployment name, update with your own if necessary
var audioFilePath = "C:\\14. Blazor LCE Youtube channel\\file1.mp3";

var openAIClient = new AzureOpenAIClient(endpoint, credentials);

var audioClient = openAIClient.GetAudioClient(deploymentName);
var result = await audioClient.TranscribeAudioAsync(audioFilePath);

Console.WriteLine("Transcribed text:");
foreach (var item in result.Value.Text)
{
    Console.Write(item);
}
```

https://github.com/luiscoco/Azure_OpenAI-WhisperModel-Read_a_Text



Azure OpenAI: Read a Text with tts model

```
using Azure;
using Azure.AI.OpenAI;
using Azure.Identity;
using OpenAI.Audio;

var endpoint = new Uri("https://voiceaimodel.openai.azure.com");
var credentials = new AzureKeyCredential("71df94595ffa4b8ba3f3b39de2d80ae9");

var deploymentName = "tts";
string speechFilePath = "C:\\Autónomo\\speech_output.wav";

var openAIClient = new AzureOpenAIClient(endpoint, credentials);

var audioClient = openAIClient.GetAudioClient(deploymentName);

var result = await audioClient.GenerateSpeechAsync("Hello World", GeneratedSpeechVoice.Echo);

Console.WriteLine($"Streaming response to {speechFilePath}");

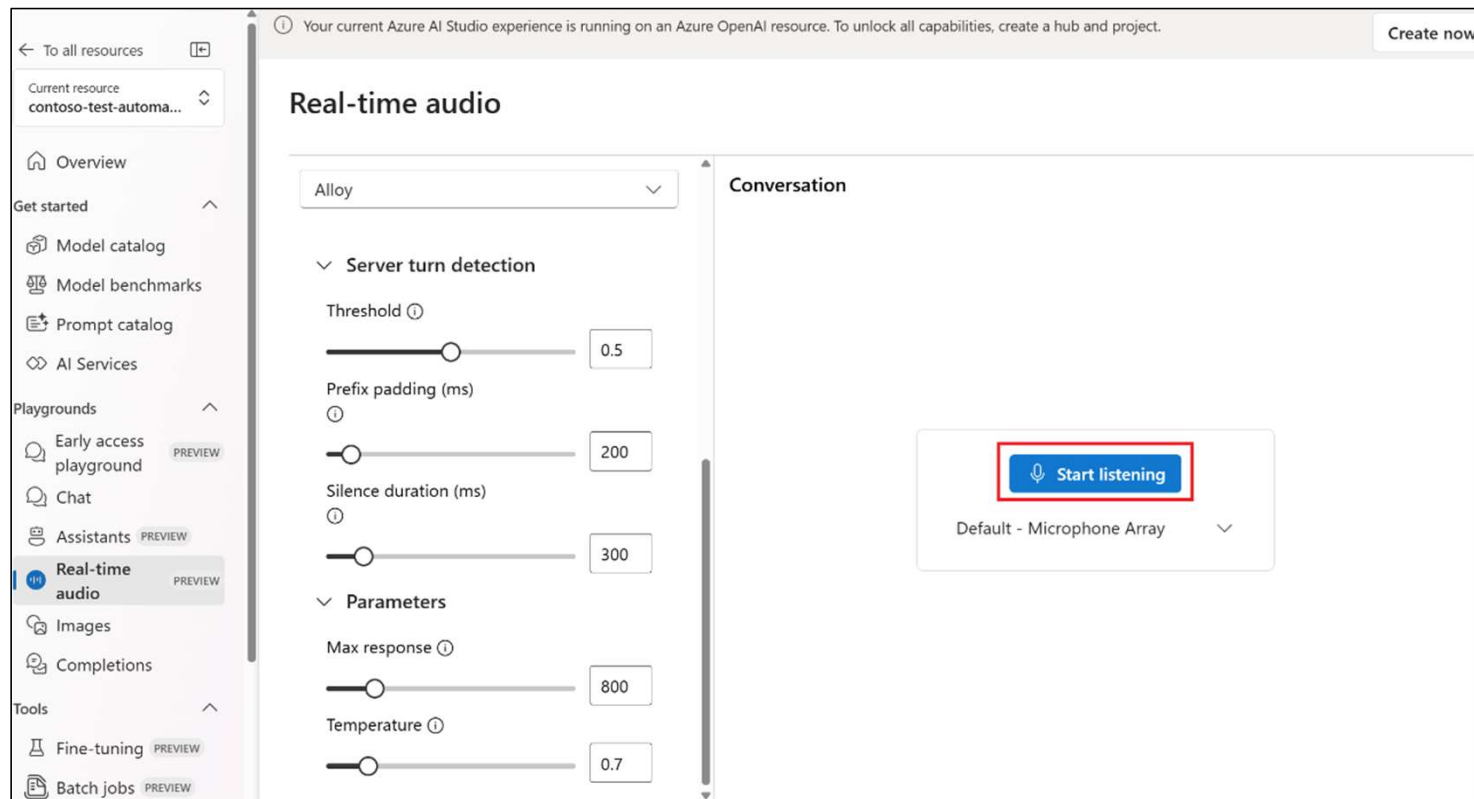
Directory.CreateDirectory(Path.GetDirectoryName(speechFilePath));

await File.WriteAllBytesAsync(speechFilePath, result.Value.ToArray());

Console.WriteLine("Finished streaming");
```




Azure OpenAI: Realtime Speech with gpt-4o-realtime-preview model



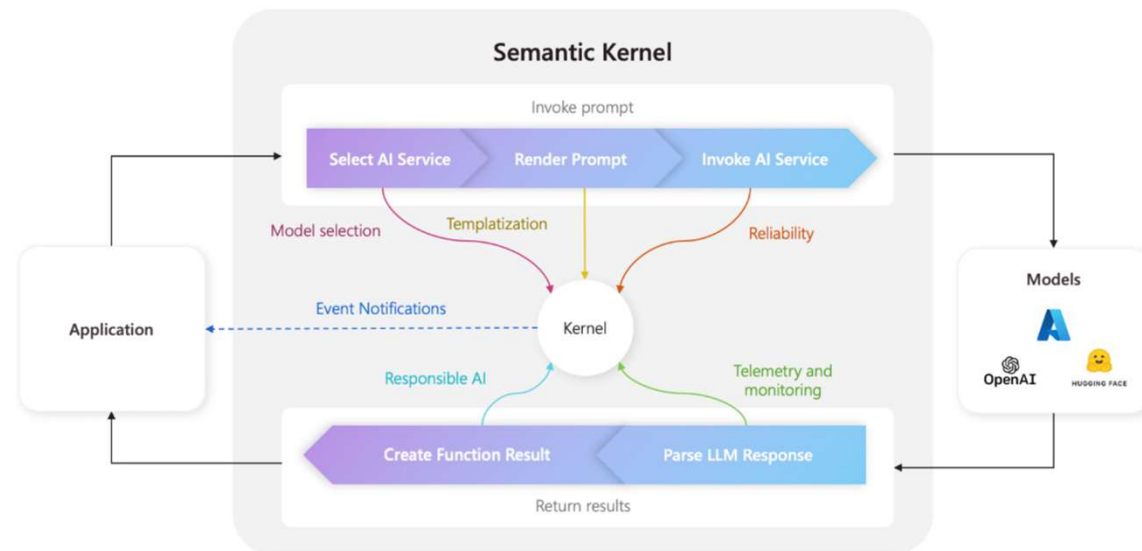
<https://learn.microsoft.com/en-us/azure/ai-services/openai/realtime-audio-quickstart>



What is Microsoft Semantic Kernel?

Semantic Kernel (SK) is a lightweight **SDK** enabling integration of **AI Large Language Models (LLMs)** with conventional programming languages.

The SK extensible programming model combines natural language **semantic functions**, traditional code **native functions**, and **embeddings-based memory** unlocking new potential and adding value to applications with AI



<https://github.com/microsoft/semantic-kernel>

© 2024 Luxoft, A DXC Technology Company. All rights reserved.



Semantic Kernel: build Kernel and invoke ChatCompletion

OpenAI service

```
using Microsoft.SemanticKernel;

var builder = Kernel.CreateBuilder();

builder.AddOpenAIChatCompletion(
    modelId: "gpt-4o",
    apiKey: "API-KEY");

Kernel kernel = builder.Build();
```

<https://platform.openai.com/api-keys>

Azure OpenAI service

```
using Microsoft.SemanticKernel;

var builder = Kernel.CreateBuilder();

builder.AddAzureOpenAIChatCompletion("gpt-4o",
    "https://luiscocoaiservice.openai.azure.com/",
    "4ETF2yj9Aq7YwZuqgkIe1xs0xWp3ulcu0hmZI9t6Vu4jH0JMaGuqJQQJ99AJACMsfrFXJ3w3AAABACOG0GFg",
    "gpt-4o");

var kernel = builder.Build();
```

```
// Example 1. Invoke the kernel with a prompt and display the result
Console.WriteLine(await kernel.InvokePromptAsync("What color is the sky?"));
Console.WriteLine();

// Example 2. Invoke the kernel with a templated prompt and display the result
KernelArguments arguments = new() { { "topic", "sea" } };
Console.WriteLine(await kernel.InvokePromptAsync("What color is the {{$topic}}?", arguments));
Console.WriteLine();

// Example 3. Invoke the kernel with a templated prompt and stream the results to the display
await foreach (var update in
    kernel.InvokePromptStreamingAsync("What color is the {{$topic}}? Provide a detailed explanation.", arguments))
{
    Console.Write(update);
}
```



Semantic Kernel: Chatbot with Ollama

```
using Microsoft.SemanticKernel;
using System;
using System.ClientModel.Primitives;

var endpoint = new Uri("http://localhost:11434");
var modelId = "phi3:latest";
#pragma warning disable SKEXP0010
var kernelBuilder = Kernel.CreateBuilder()
    .AddVolatileVectorStore()
    .AddOpenAIChatCompletion(modelId: modelId, apiKey: null,
        endpoint: endpoint);
#pragma warning restore SKEXP0010
var kernel = kernelBuilder.Build();

const string skPrompt = @"
JamicanFoodBot can give you recommendations on Jamaican Food
cuisine and recipes.
It can give explicit instructions on how to cook these dishes. It
will provide you ingredients, measurements, and cook time.

{{ $history }}
User: {{ $userInput }}
JamicanFoodBot:";

var chatFunction = kernel.CreateFunctionFromPrompt(skPrompt);
var history = "";
var arguments = new KernelArguments
{
    ["history"] = history
};
```

```
while (true)
{
    Console.WriteLine("Tell me what kind of food you are in the mood
for: ");
    var userInput = Console.ReadLine();
    if (userInput.Equals("quit",
        StringComparison.OrdinalIgnoreCase))
    {
        Console.WriteLine("Goodbye!");
        break;
    }
    arguments["userInput"] = userInput;
    Console.WriteLine("JamicanFoodBot: ");
    await foreach (var chunk in
        kernel.InvokeStreamingAsync(chatFunction, arguments))
    {
        Console.WriteLine(chunk);
        history += chunk;
    }
    history += $"User: {userInput}\n";
    arguments["history"] = history;
    Console.WriteLine();
    Console.WriteLine("Conversation history:");
    Console.WriteLine(history);
}
```



Semantic Kernel: Logging

Azure OpenAI service

```
using Microsoft.SemanticKernel;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.DependencyInjection;

var builder = Kernel.CreateBuilder();

builder.AddAzureOpenAIChatCompletion("gpt-4o",
    "https://luiscocoaiservice.openai.azure.com/",
    "",
    "gpt-4o");

builder.Services.AddLogging(c => c.AddConsole().SetMinimumLevel(LogLevel.Trace));

var kernel = builder.Build();

// Example 1. Invoke the kernel with a prompt and display the result
Console.WriteLine(await kernel.InvokePromptAsync("What color is the sky?"));
Console.WriteLine();
```

Search Solution Explorer (Ctrl+;)

Solution 'ConsoleApp1' (1 of 1 project)

- ConsoleApp1
 - Dependencies
 - Analyzers
 - Frameworks
 - Packages
 - Microsoft.Extensions.Logging (9.0.0-rc.2.24473.5)
 - Microsoft.Extensions.Logging.Console (9.0.0-rc.2.24473.5)
 - Microsoft.SemanticKernel (1.25.0)
 - .gitattributes
 - .gitignore
 - C# Program.cs
 - README.md



Semantic Kernel: Templated Prompt

```
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Connectors.OpenAI;

var builder = Kernel.CreateBuilder();

builder.AddOpenAIChatCompletion(
    modelId: "gpt-4o",
    apiKey: "API-KEY");

Kernel kernel = builder.Build();

KernelArguments arguments;

// Example 4. Invoke the kernel with a templated prompt and execution settings
arguments = new(new OpenAIPromptExecutionSettings { MaxTokens = 500, Temperature = 0.5 }) { { "topic", "dogs" } };
Console.WriteLine(await kernel.InvokePromptAsync("Tell me a story about {{$topic}}", arguments));

// Example 5. Invoke the kernel with a templated prompt and execution settings configured to return JSON
#pragma warning disable SKEXP0010
arguments = new(new OpenAIPromptExecutionSettings { ResponseFormat = "json_object" }) { { "topic", "chocolate" } };
Console.WriteLine(await kernel.InvokePromptAsync("Create a recipe for a {{$topic}} cake in JSON format", arguments));
```




Semantic Kernel: Plugin

```
using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.Connectors.OpenAI;
using System.ComponentModel;
using System.Reflection;
using System.Text.Json.Serialization;

var builder = Kernel.CreateBuilder();

builder.AddAzureOpenAIChatCompletion("gpt-4o",
    "https://luiscoaiservice.openai.azure.com/",
    "4ETF2yj9Aq7YwZuqgkIe1xs0xWp3ulcu0hmZI9t6Vu4jH0JMaGuqJQQJ99AJACMsfrFXJ3w3AAABACOG0GFg",
    "gpt-4o");

builder.Plugins.AddFromType<TimeInformation>();

var kernel = builder.Build();

// Example 1. Invoke the kernel with a prompt that asks the AI for information it cannot provide and may hallucinate
Console.WriteLine(await kernel.InvokePromptAsync("How many days until Christmas?"));

//// Example 2. Invoke the kernel with a templated prompt that invokes a plugin and display the result
Console.WriteLine(await kernel.InvokePromptAsync("The current time is {{TimeInformation.GetCurrentUtcTime}}. How many days until Christmas?"));

#pragma warning disable
// Example 3. Invoke the kernel with a prompt and allow the AI to automatically invoke functions
OpenAIPromptExecutionSettings settings = new() { FunctionChoiceBehavior = FunctionChoiceBehavior.Auto() };
Console.WriteLine(await kernel.InvokePromptAsync("How many days until Christmas? Explain your thinking.", new(settings)));

1 reference
public class TimeInformation
{
    [KernelFunction]
    [Description("Retrieves the current time in UTC.")]
    0 references
    public string GetCurrentUtcTime() => DateTime.UtcNow.ToString("R");
}
```

What is Microsoft.Extensions.AI?

These libraries provide a unified layer of **C# common abstractions** and **standard middleware implementation** for interacting with AI services

.NET Application

Leveraging AI

Microsoft.Extensions.AI (Middleware)

AI, Data Tools, & Services

UI Components
(Smart Components)

AI SDKS
(OllamaSharp, OpenAI, Azure AI Inference)

Libraries
(Semantic Kernel, AutoGen)

Vector Store SDKs

Developer Tools (Promptly)

Microsoft.Extensions.AI.Abstractions

Microsoft.Extensions.VectorData

For example, the `IChatClient` interface allows consumption of language models, whether hosted remotely or running locally. Any .NET package providing an AI client can implement this interface, enabling seamless integration with consuming .NET code.

Copy

```
IChatClient client =  
    environment.IsDevelopment ?  
        new OllamaChatClient(...) :  
        new AzureAIInferenceChatClient(...);
```

Then, regardless of the provider you're using, you can send requests as follows:

Copy

```
var response = await chatClient.CompleteAsync(  
    "Translate the following text into Pig Latin: I love .NET and AI");  
  
Console.WriteLine(response.Message);
```


Microsoft.Extensions.AI: Azure AI Inference (Github Models)

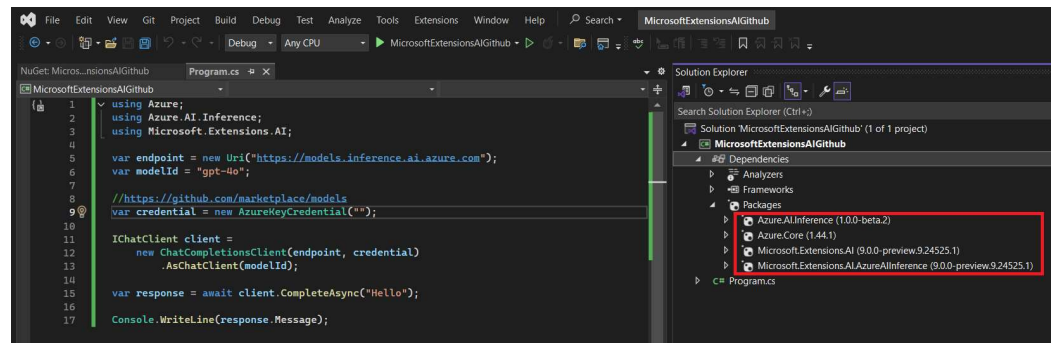
Install the [Microsoft.Extensions.AI.AzureAIInference](https://www.nuget.org/packages/Microsoft.Extensions.AI.AzureAIInference) NuGet package

```
using Azure;
using Azure.AI.Inference;
using Microsoft.Extensions.AI;

IChatClient client =
    new ChatCompletionsClient(
        endpoint: new Uri("https://models.inference.ai.azure.com"),
        new AzureKeyCredential(Environment.GetEnvironmentVariable("GH_TOKEN")))
        .AsChatClient("Phi-3.5-MoE-instruct");

var response = await client.CompleteAsync("What is AI?");

Console.WriteLine(response.Message);
```



https://github.com/luisco/microsoft-extensions-ai-github_ai_model

<https://github.com/marketplace/models>

Microsoft.Extensions.AI: OpenAI

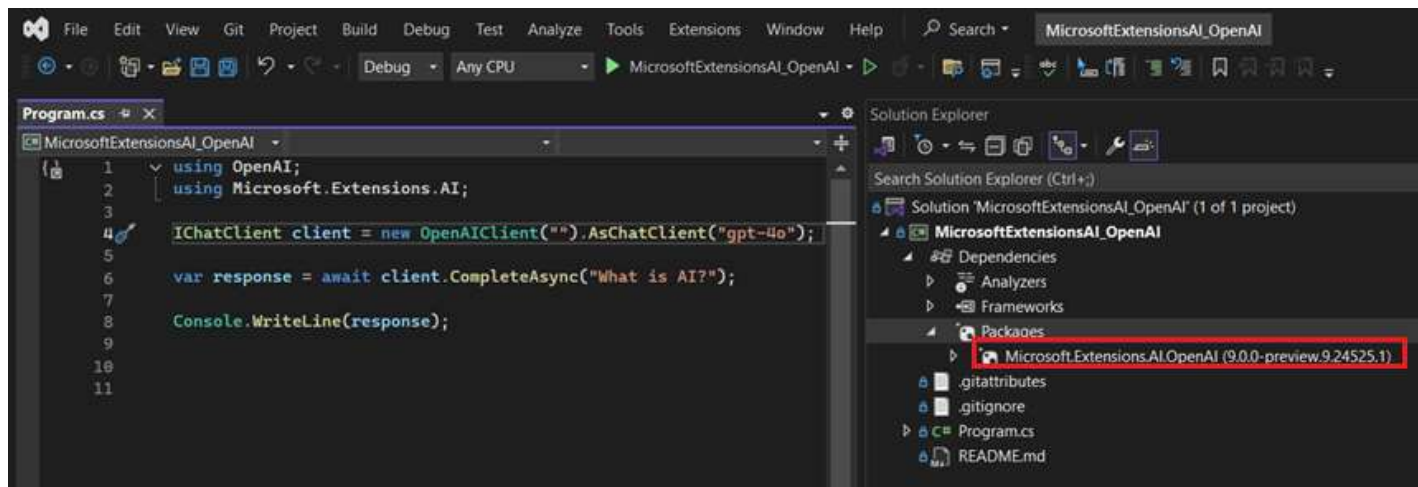
Install the [Microsoft.Extensions.AI.OpenAI](#) NuGet package

```
using OpenAI;
using Microsoft.Extensions.AI;

IChatClient client =
    new OpenAIClient(Environment.GetEnvironmentVariable("OPENAI_API_KEY"))
        .AsChatClient(modelId: "gpt-4o-mini");

var response = await client.CompleteAsync("What is AI?");

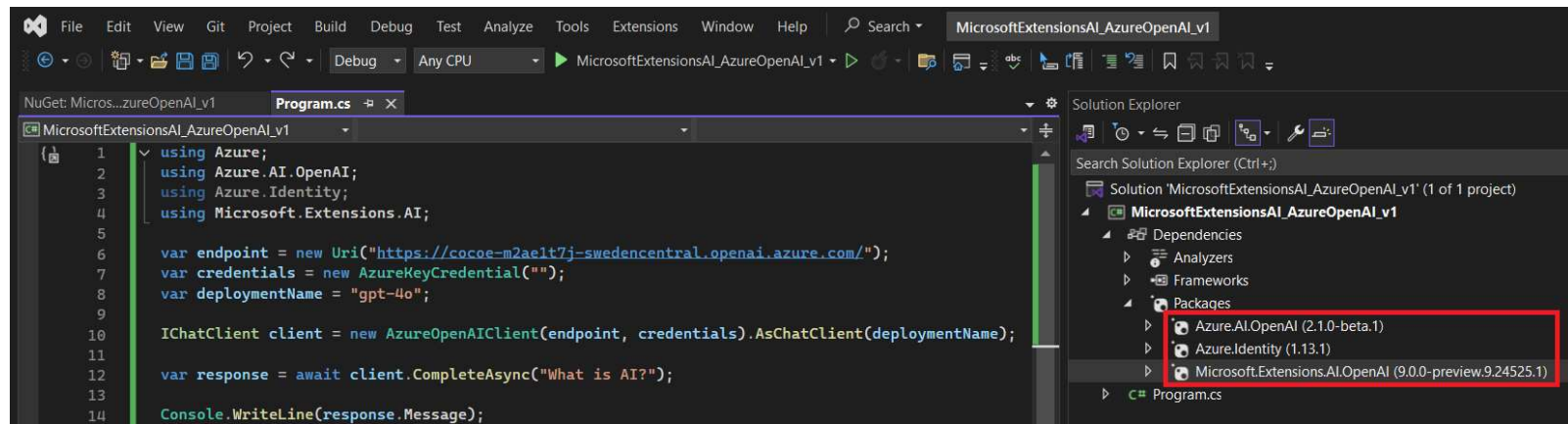
Console.WriteLine(response.Message);
```



Microsoft.Extensions.AI: Azure OpenAI

Install the [Microsoft.Extensions.AI.OpenAI](#), [Azure.AI.OpenAI](#), and [Azure.Identity](#) NuGet packages

```
using Azure.AI.OpenAI;  
using Azure.Identity;  
using Microsoft.Extensions.AI;  
  
IChatClient client =  
    new AzureOpenAIClient(  
        new Uri(Environment.GetEnvironmentVariable("AZURE_OPENAI_ENDPOINT")),  
        new DefaultAzureCredential()  
        .AsChatClient(modelId: "gpt-4o-mini");  
  
var response = await client.CompleteAsync("What is AI?");  
  
Console.WriteLine(response.Message);
```



Microsoft.Extensions.AI: Ollama

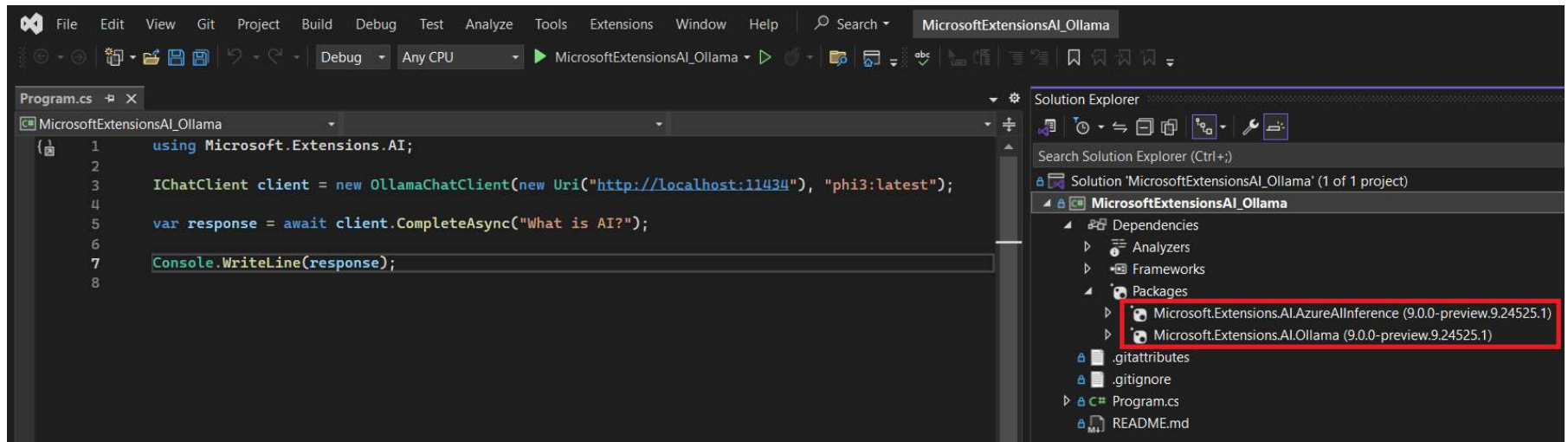
Install the [Microsoft.Extensions.AI.Ollama](#) NuGet package

```
using Microsoft.Extensions.AI;

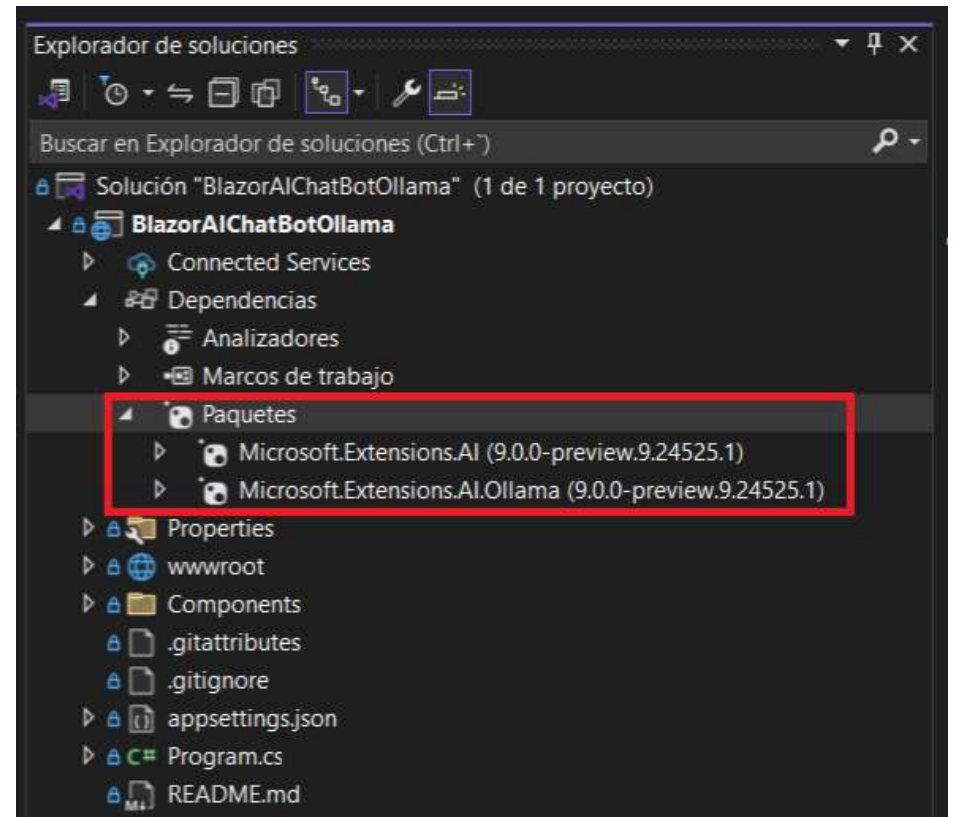
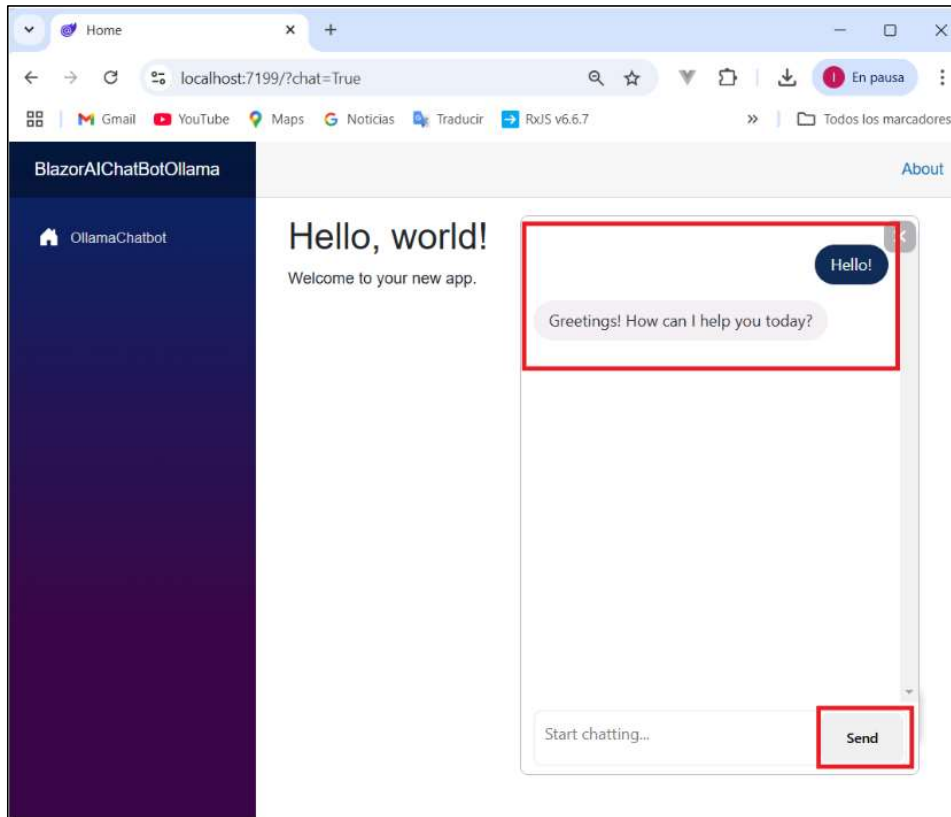
IChatClient client =
    new OllamaChatClient(new Uri("http://localhost:11434/"), "llama3.1");

var response = await client.CompleteAsync("What is AI?");

Console.WriteLine(response.Message);
```

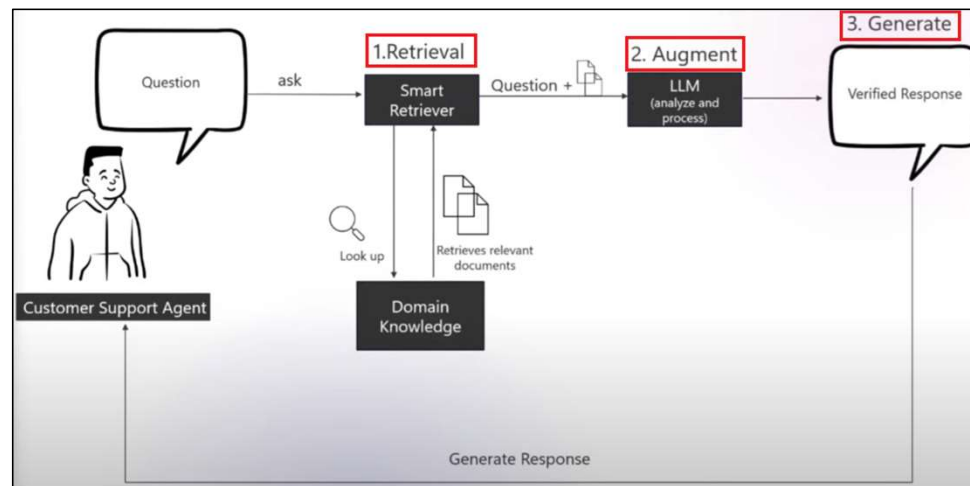


Microsoft.Extensions.AI: Blazor Web App with Ollama Chatbot



RAG (Retrieval Augmented Generation)

RAG is an AI technique that combines retrieval-based and generative models to improve information relevance and accuracy in responses. In RAG, a retrieval model **first searches** a large **database** or set of **documents** to find relevant information based on the user's query. Then, a **generative model** (like GPT) **uses the retrieved information** to produce a coherent, context-aware response. This approach helps the AI provide responses that are both **accurate** and **grounded** in relevant data sources, making it particularly useful for tasks requiring specific or factual information.



Efficiently **Searching PDF Content with AI**: Leveraging LLMs and RAG for Enhanced Retrieval

<https://github.com/microsoft/AzureDataRetrievalAugmentedGenerationSamples>

© 2024 Luxoft, A DXC Technology Company. All rights reserved.

eShopsupport [Youtube video](#)



Staff web UI

The screenshot shows a staff web UI for eShopSupport. It includes a header with a date '20419' and a user profile 'Staff'. The main area contains a form with fields for 'Case type' (set to 'Question'), 'Status' (set to 'Open'), 'Customer' (set to 'Customer McCustomer'), and 'Product' (set to 'TCX Kayak 2000'). Below the form is a 'Summary' section with the text: 'Summary: Customer wishes to perform backflips and asks how to update kayak firmware.' To the right is an 'Assistant' chat window with a message: 'What does the manual say about this?' and a response: 'Page 44 of the manual states that backflips are supported as of firmware version 3.17. Link: Manual.pdf page 44'. At the bottom is a 'Citation' section with the text: 'We have looked into this, and found that the TCX Kayak model 2000 does support backflips as of v3.17. To upgrade, please see http://...'. Callouts point to various features: 'Classification' points to the 'Case type' dropdown; 'Summarization' points to the 'Summary' section; 'Typeahead (content generation)' points to the 'Customer' field; 'Semantic search' points to the 'Product' field; 'Q&A (RAG)' points to the 'Assistant' chat window; and 'Citation' points to the 'Citation' section.

Classification

Semantic search

Q&A (RAG)

Summarization

Typeahead (content generation)

Citation

<https://github.com/dotnet/eShopSupport>

https://github.com/luiscoco/Curso_Aprende_Blazor-Nivel_Intermedio-AI_eShopSupport-Assistant_Fixed-With-Azure_OpenAI

https://github.com/luiscoco/Curso_Aprende_Blazor-Nivel_Intermedio-AI_DataGenerator

