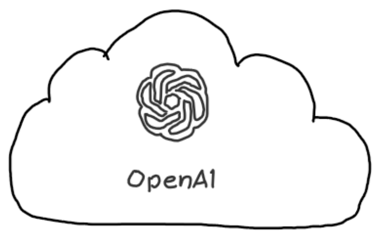
Azure OpenAI One Day Workshop

Robert Eichenseer
Senior Services Engineer

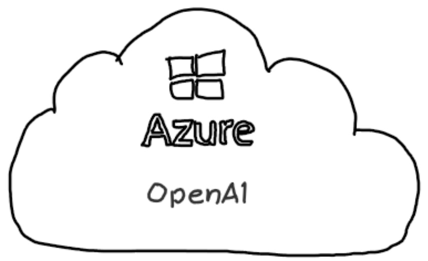
OpenAI / Azure OpenAI

MS Azure OpenAI co-develops the APIs with OpenAI, ensuring compatibility and a smooth transition from one to the other.



OpenAI

- “Startup” which runs on Azure and partners with MS
- Hosts the famous https://chat.openai.com
- Alpha Models
- Early access
- Prototyping early features



MS Azure OpenAI

- Enterprise Grade (security, availability, networking, regional availability, financially backed SLA …)
- Same models as OpenAI
- Fine-tuned models

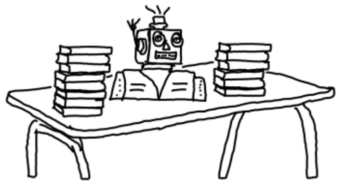
Definition

Artificial Intelligence (AI)

... the intelligence demonstrated by machines, similar to the intelligence of humans ...

Model Use

https://en.wikipedia.org/wiki/Artificial_intelligence



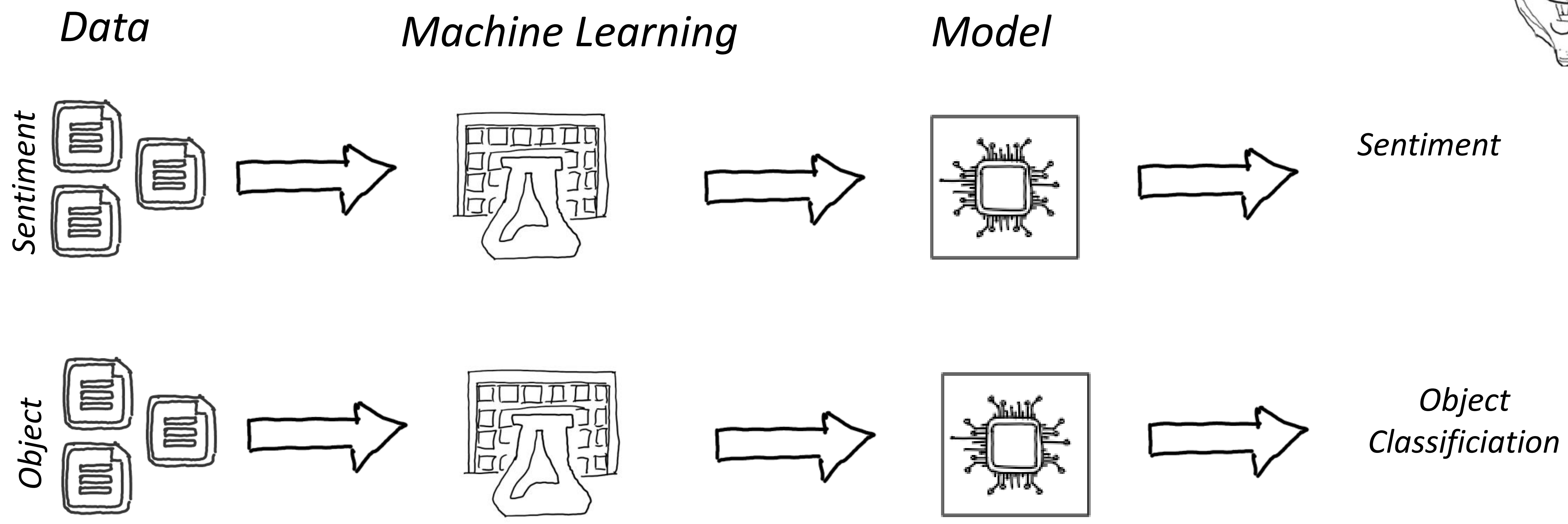
Machine Learning (ML)

... learning of an artificial system from examples and the ability to generalize them after the end of a learning phase ...

Create Model

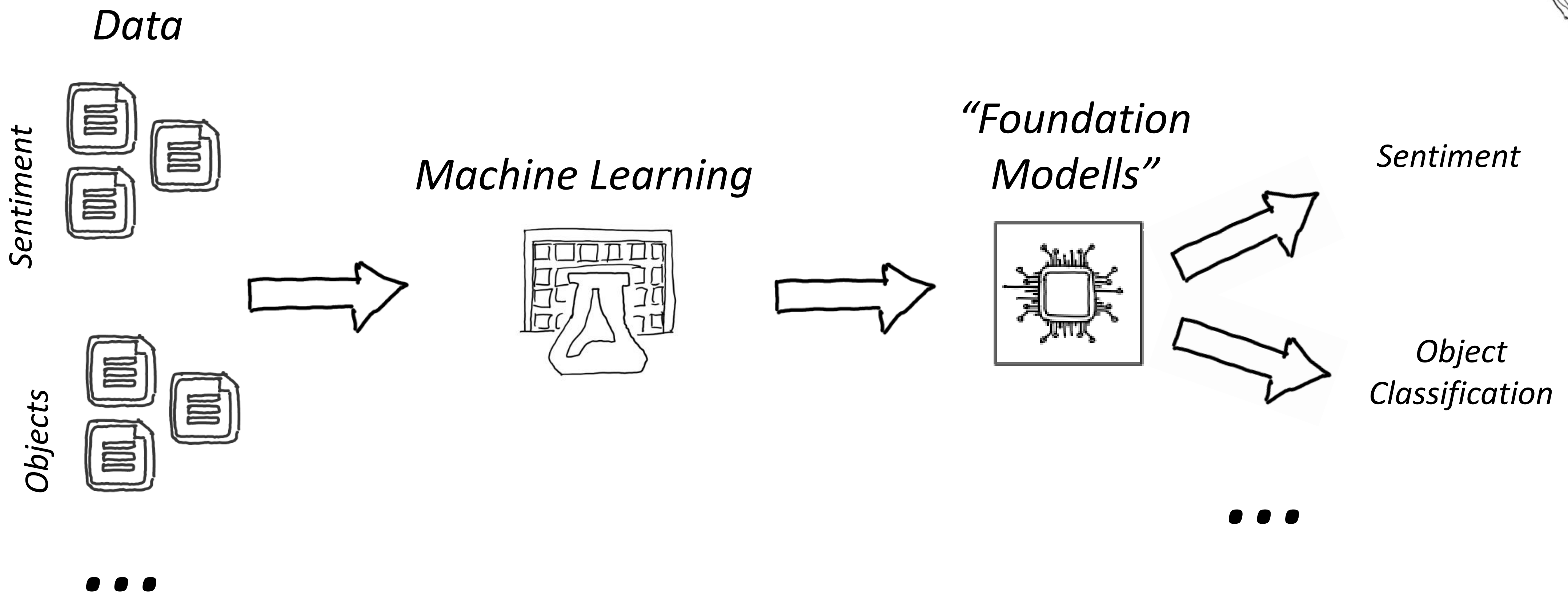
https://en.wikipedia.org/wiki/Machine_learning

Classic Approach



...

Large Language Models



OpenAI LLM Models

GPT family

Understanding of text, code, and images - generating text and code

DALL-E

Creation and editing of images using text

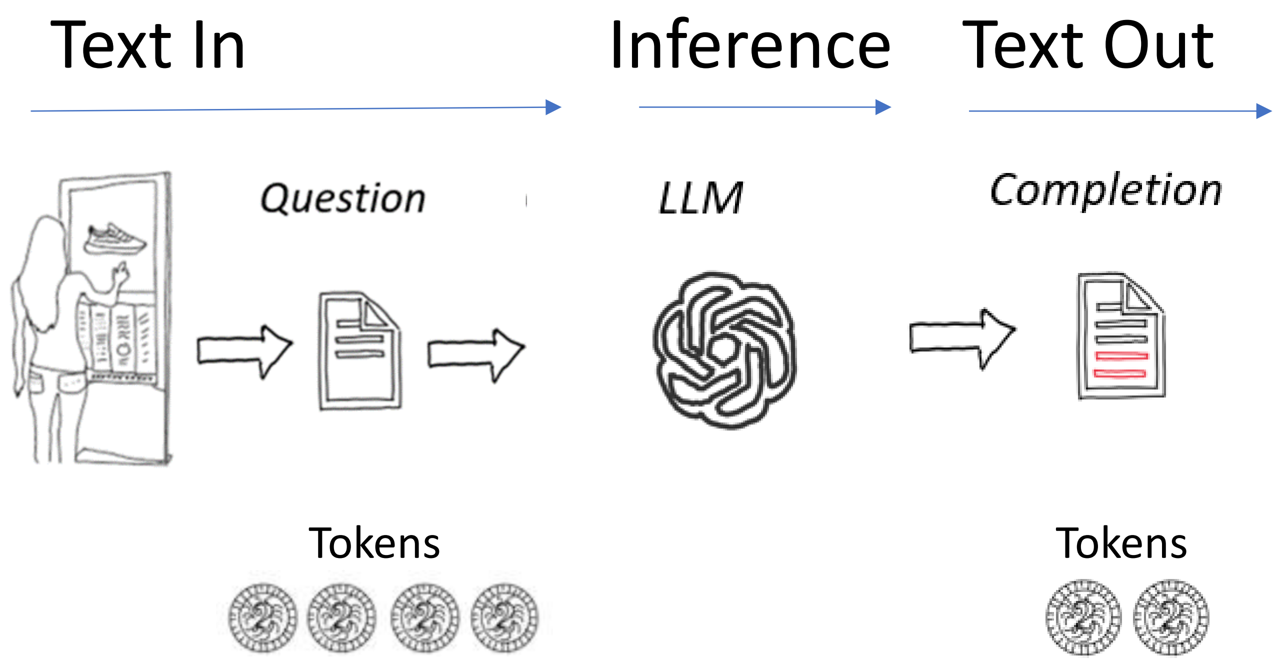
Embeddings

Conversion of text into a numerical representation (vector) while retaining semantic meaning

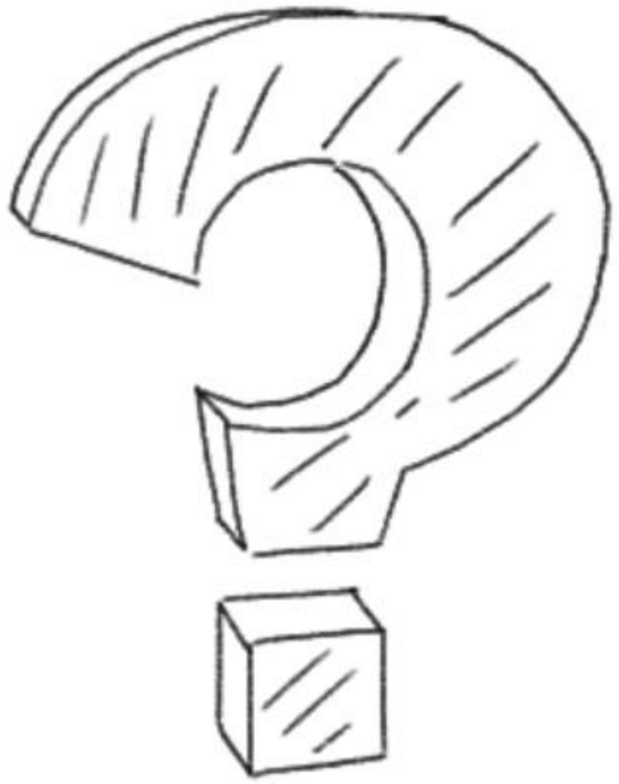
Whisper

Umwandlung von Sprache nach Text

Tokens & Pricing



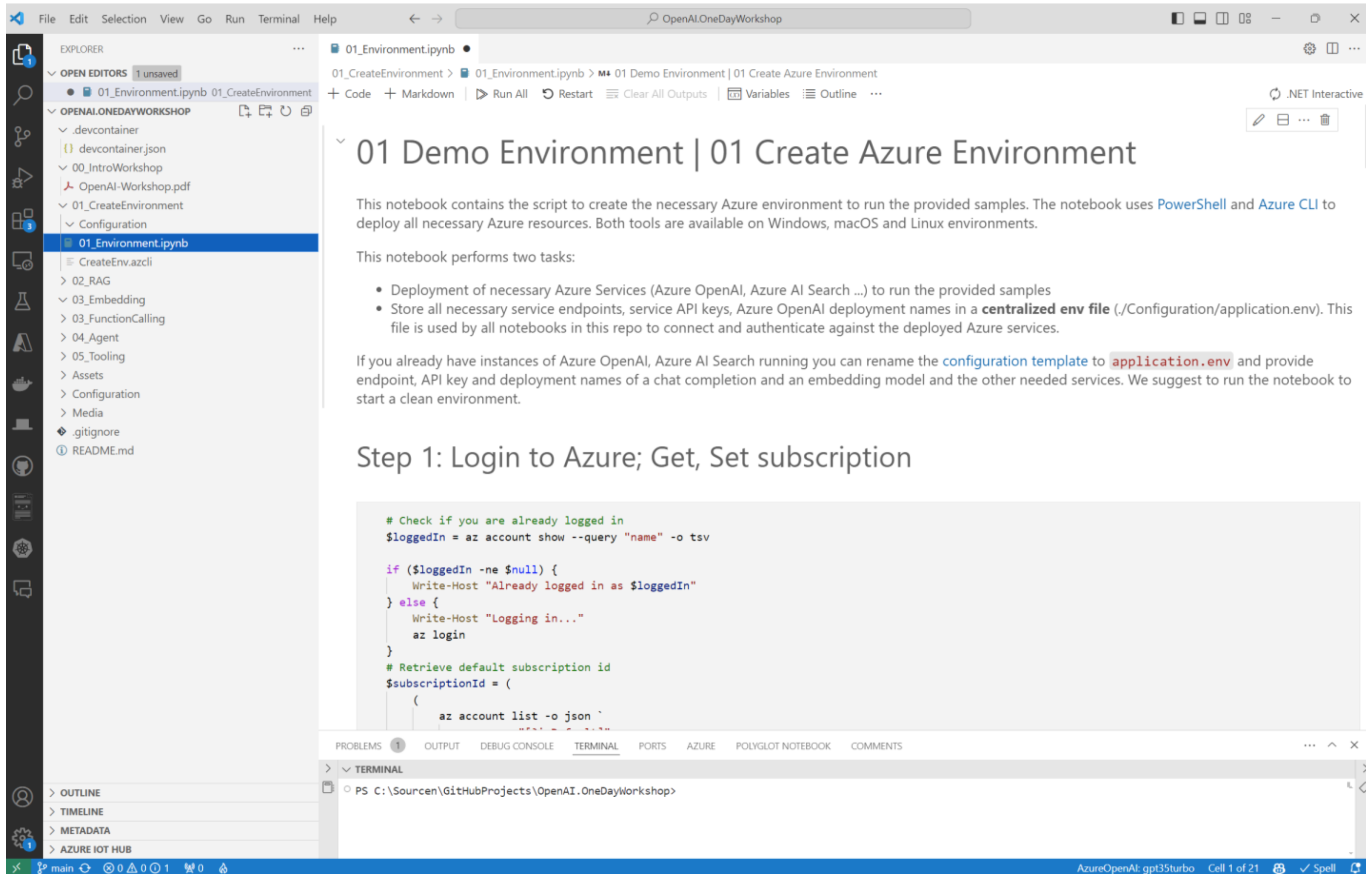
LLM in 15 Seconds



Large language models (LLMs) take a limited number of so-called tokens (words or subwords) as input and they predict the next most likely word or token and add it to the input text.

Congratulation

Setup



01 Demo Environment | 01 Create Azure Environment

This notebook contains the script to create the necessary Azure environment to run the provided samples. The notebook uses PowerShell and Azure CLI to deploy all necessary Azure resources. Both tools are available on Windows, macOS and Linux environments.

This notebook performs two tasks:

- Deployment of necessary Azure Services (Azure OpenAI, Azure AI Search ...) to run the provided samples
- Store all necessary service endpoints, service API keys, Azure OpenAI deployment names in a **centralized env file** (./Configuration/application.env). This file is used by all notebooks in this repo to connect and authenticate against the deployed Azure services.

If you already have instances of Azure OpenAI, Azure AI Search running you can rename the configuration template to `application.env` and provide endpoint, API key and deployment names of a chat completion and an embedding model and the other needed services. We suggest to run the notebook to start a clean environment.

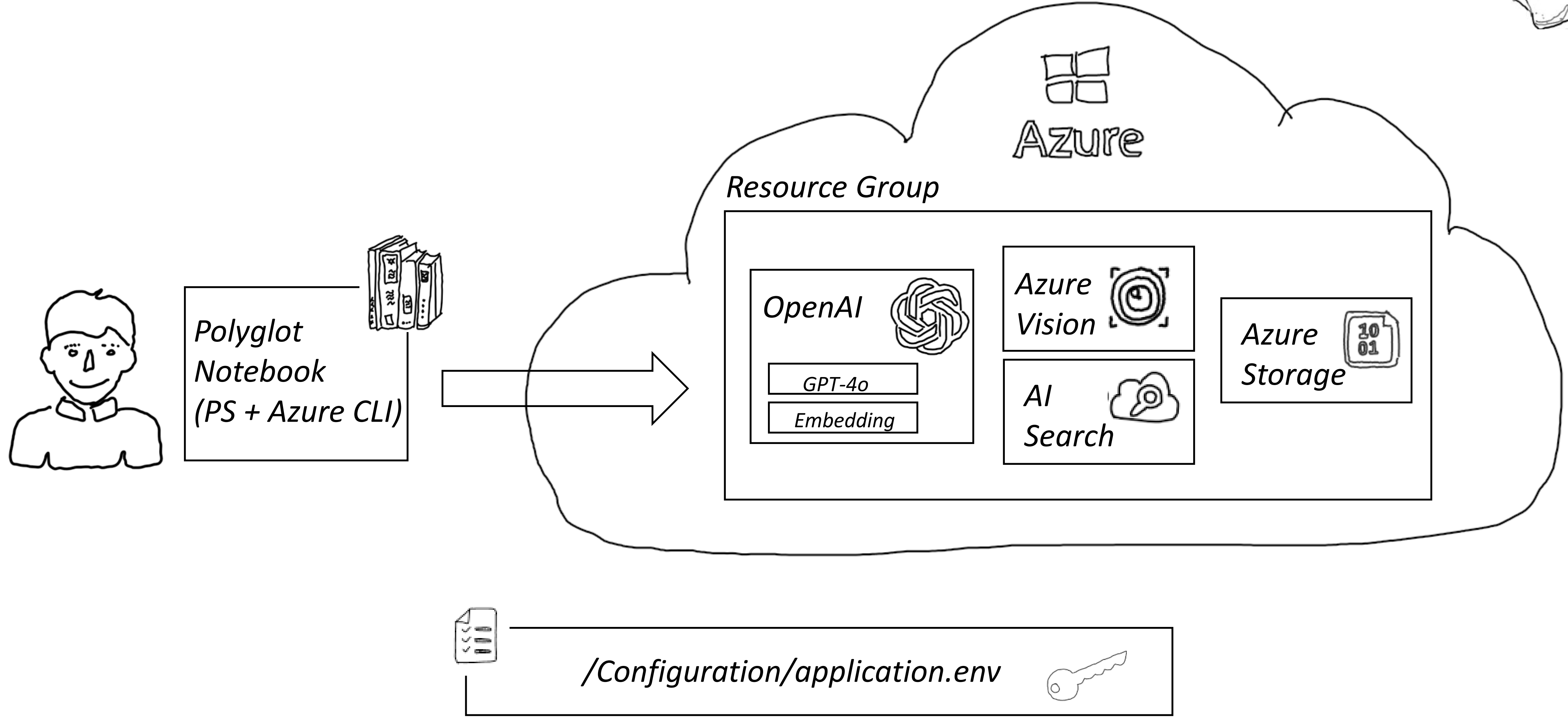
Step 1: Login to Azure; Get, Set subscription

```
# Check if you are already logged in
$loggedIn = az account show --query "name" -o tsv

if ($loggedIn -ne $null) {
    Write-Host "Already logged in as $loggedIn"
} else {
    Write-Host "Logging in..."
    az login
}
# Retrieve default subscription id
$subscriptionId = (
    (
        az account list -o json `
```

```
PS C:\Sourcen\GitHubProjects\OpenAI.OneDayWorkshop>
```

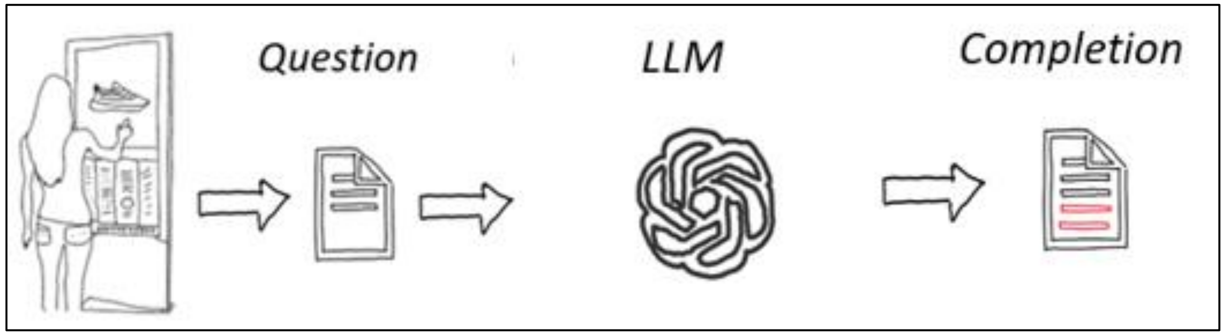
https://www.github.com/RobertEichenseer/OpenAI.OneDayWorkshop

Setup

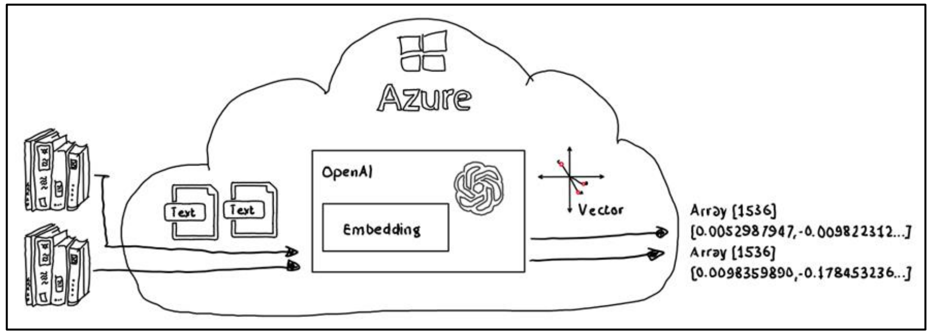

Polyglot Notebook (PS + Azure CLI)
Azure
Resource Group
OpenAI
GPT-4o
Embedding
Azure Vision
AI Search
Azure Storage
/Configuration/application.env

RAG (Retrieval-Augmented Generation)

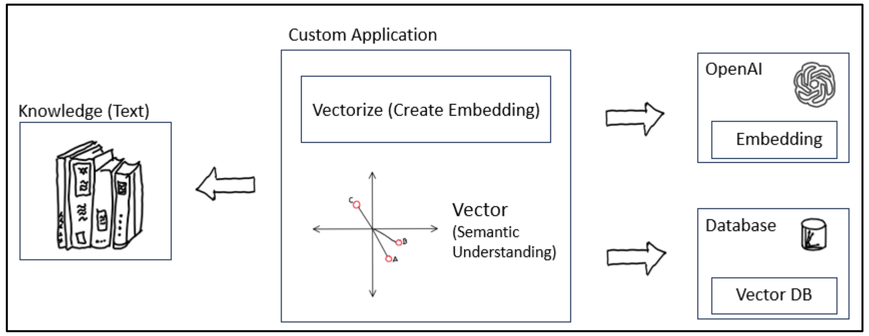
Chat Completion



Embedding



Vector DB



Chat Completion

Chat Completion API Call

/02_RAG/02_01_ChatCompletion/01_ChatCompletionText_REST.ipynb

Chat Completion SDK

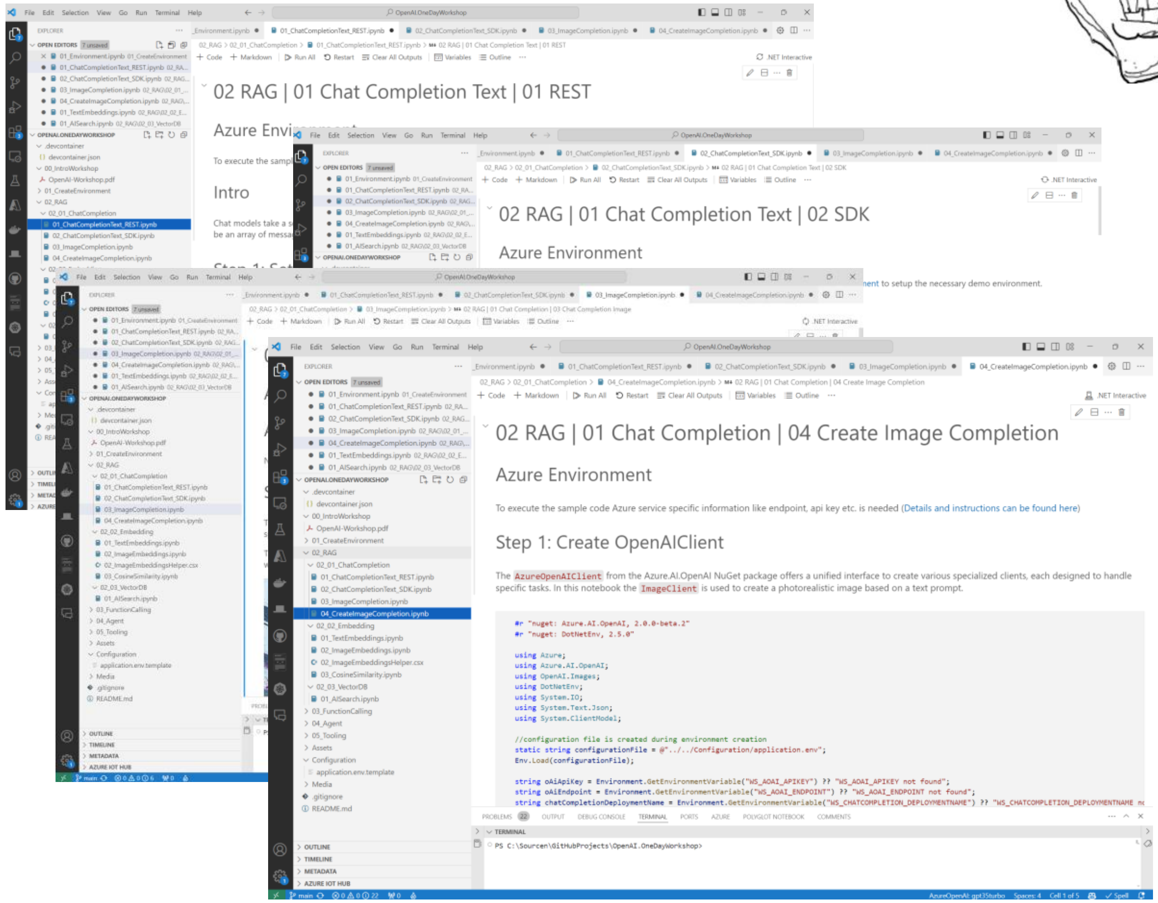
/02_RAG/02_01_ChatCompletion/02_ChatCompletionText_SDK.ipynb

Image Completion SDK

/02_RAG/02_01_ChatCompletion/03_ImageCompletion.ipynb

Create Image Completion

/02_RAG/02_01_ChatCompletion/04_CreateImageCompletion.ipynb



02 RAG | 01 Chat Completion Text | 01 REST

02 RAG | 01 Chat Completion Text | 02 SDK

02 RAG | 01 Chat Completion | 04 Create Image Completion

Azure Environment

To execute the sample code Azure service specific information like endpoint, api key etc. is needed (Details and instructions can be found here)

Step 1: Create OpenAIClient

The `AzureOpenAIClient` from the Azure.AI.OpenAI NuGet package offers a unified interface to create various specialized clients, each designed to handle specific tasks. In this notebook the `ImageClient` is used to create a photorealistic image based on a text prompt.

```
#r "nuget: Azure.AI.OpenAI, 2.0.0-beta.2"
#r "nuget: DotNetEnv, 2.5.0"

using Azure;
using Azure.AI.OpenAI;
using OpenAI.Images;
using DotNetEnv;
using System.IO;
using System.Text.Json;
using System.ClientModel;

//configuration file is created during environment creation
static string configurationFile = @"../../Configuration/application.env";
Env.Load(configurationFile);

string oAiApiKey = Environment.GetEnvironmentVariable("WS_AOAI_APIKEY") ?? "WS_AOAI_APIKEY not found";
string oAiEndpoint = Environment.GetEnvironmentVariable("WS_AOAI_ENDPOINT") ?? "WS_AOAI_ENDPOINT not found";
string chatCompletionDeploymentName = Environment.GetEnvironmentVariable("WS_CHATCOMPLETION_DEPLOYMENTNAME") ?? "WS_CHATCOMPLETION_DEPLOYMENTNAME no
```

```
PS C:\Sourcen\GitHubProjects\OpenAI.OneDayWorkshop>
```

Embedding

Text Embedding

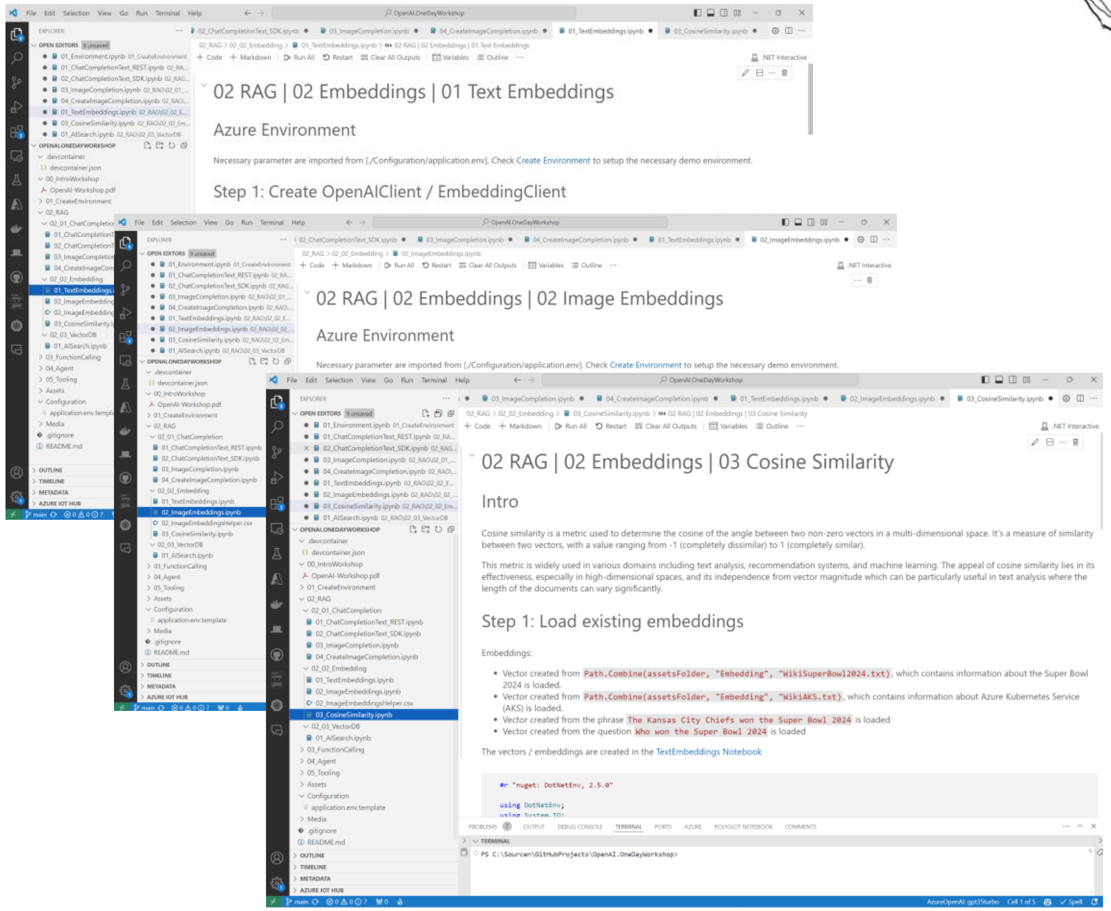
/02_RAG/02_02_Embedding/01_TextEmbedding.ipynb

Image Embedding

/02_RAG/02_02_Embedding/02_ImageEmbedding.ipynb

Cosine Similarity

/02_RAG/02_02_Embedding/03_CosineSimilarity.ipynb



02 RAG | 02 Embeddings | 01 Text Embeddings

Azure Environment

Necessary parameter are imported from [./Configuration/application.env]. Check Create Environment to setup the necessary demo environment.

Step 1: Create OpenAIClient / EmbeddingClient

02 RAG | 02 Embeddings | 02 Image Embeddings

Azure Environment

Necessary parameter are imported from [./Configuration/application.env]. Check Create Environment to setup the necessary demo environment.

02 RAG | 02 Embeddings | 03 Cosine Similarity

Intro

Cosine similarity is a metric used to determine the cosine of the angle between two non-zero vectors in a multi-dimensional space. It's a measure of similarity between two vectors, with a value ranging from -1 (completely dissimilar) to 1 (completely similar).

This metric is widely used in various domains including text analysis, recommendation systems, and machine learning. The appeal of cosine similarity lies in its effectiveness, especially in high-dimensional spaces, and its independence from vector magnitude which can be particularly useful in text analysis where the length of the documents can vary significantly.

Step 1: Load existing embeddings

Embeddings:

- Vector created from `Path.Combine(assetsFolder, "Embedding", "WikiSuperBowl2024.txt")`, which contains information about the Super Bowl 2024 is loaded.
- Vector created from `Path.Combine(assetsFolder, "Embedding", "WikiAKS.txt")`, which contains information about Azure Kubernetes Service (AKS) is loaded.
- Vector created from the phrase `The Kansas City Chiefs won the Super Bowl 2024` is loaded
- Vector created from the question `Who won the Super Bowl 2024` is loaded

The vectors / embeddings are created in the TextEmbeddings Notebook

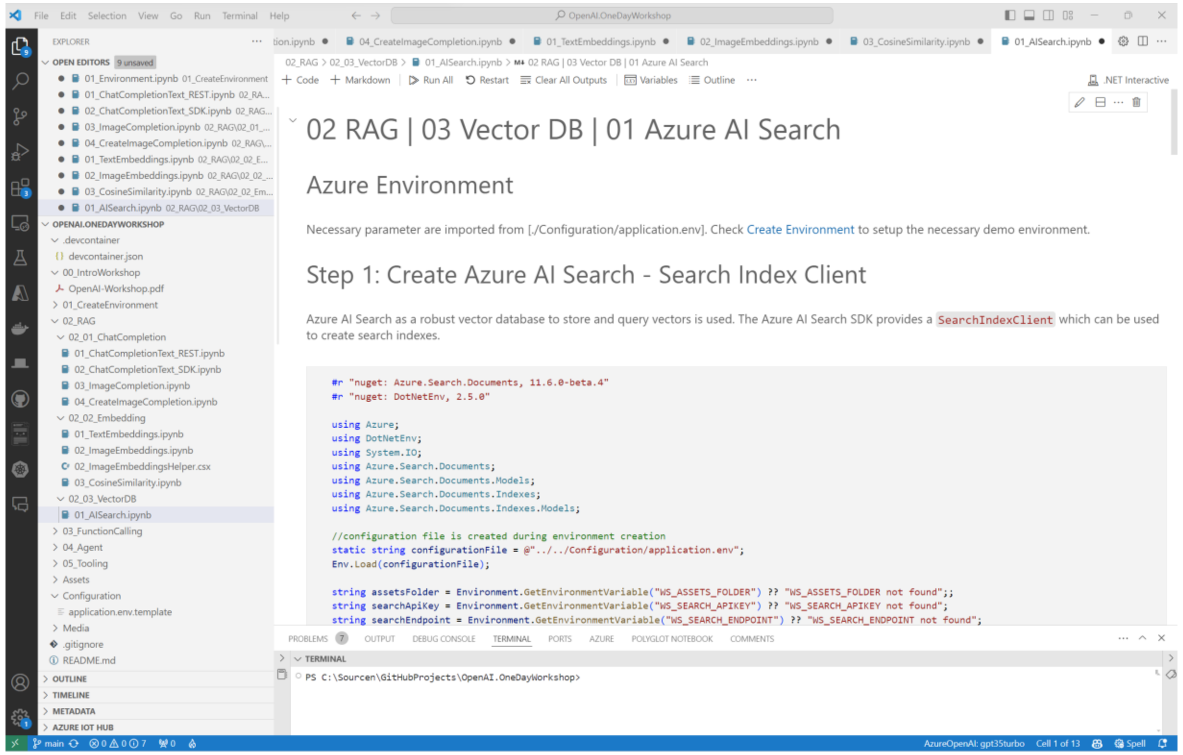
```
#r "nuget: DotNetEnv, 2.5.0"

using DotNetEnv;
using System.IO;
```

```
PS C:\Sourcen\GitHubProjects\OpenAI.OneDayWorkshop>
```

Vector DB

Azure AI Search

/02_RAG/02_03_VectorDB/01_AISearch.ipynb



02 RAG | 03 Vector DB | 01 Azure AI Search

Azure Environment

Necessary parameter are imported from [./Configuration/application.env]. Check Create Environment to setup the necessary demo environment.

Step 1: Create Azure AI Search - Search Index Client

Azure AI Search as a robust vector database to store and query vectors is used. The Azure AI Search SDK provides a `SearchIndexClient` which can be used to create search indexes.

```
#r "nuget: Azure.Search.Documents, 11.6.0-beta.4"
#r "nuget: DotNetEnv, 2.5.0"

using Azure;
using DotNetEnv;
using System.IO;
using Azure.Search.Documents;
using Azure.Search.Documents.Models;
using Azure.Search.Documents.Indexes;
using Azure.Search.Documents.Indexes.Models;

//configuration file is created during environment creation
static string configurationFile = @"../../Configuration/application.env";
Env.Load(configurationFile);

string assetsFolder = Environment.GetEnvironmentVariable("WS_ASSETS_FOLDER") ?? "WS_ASSETS_FOLDER not found";;
string searchApiKey = Environment.GetEnvironmentVariable("WS_SEARCH_APIKEY") ?? "WS_SEARCH_APIKEY not found";
string searchEndpoint = Environment.GetEnvironmentVariable("WS_SEARCH_ENDPOINT") ?? "WS_SEARCH_ENDPOINT not found";
```

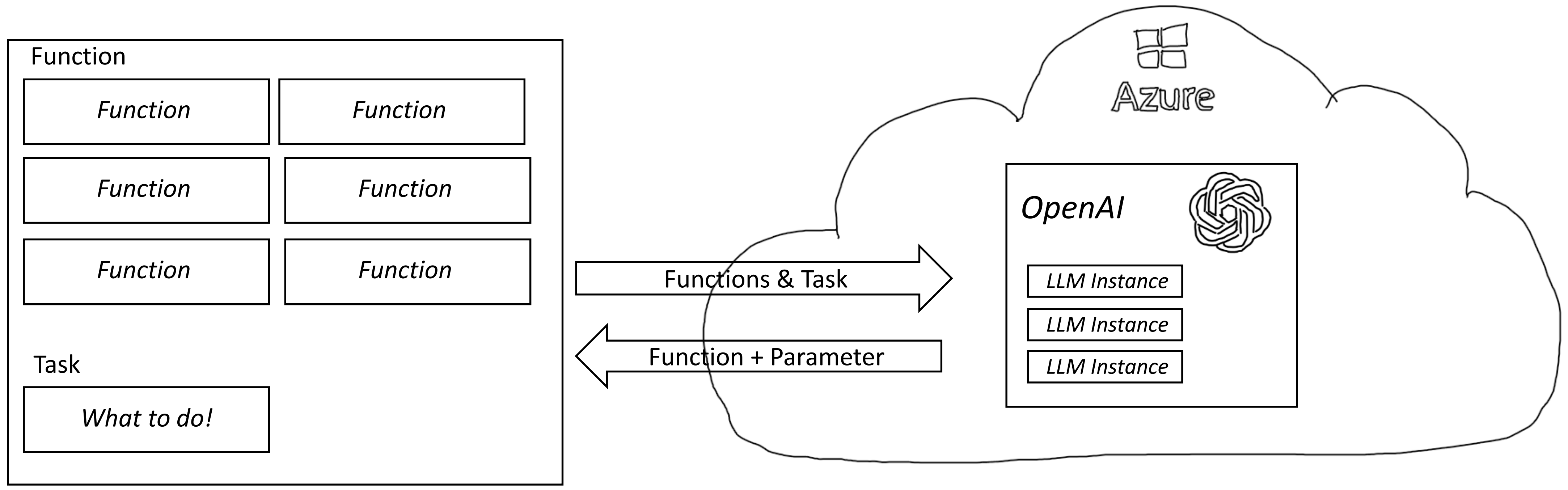
```
PS C:\Sourcen\GitHubProjects\OpenAI.OneDayWorkshop>
```

Get to work

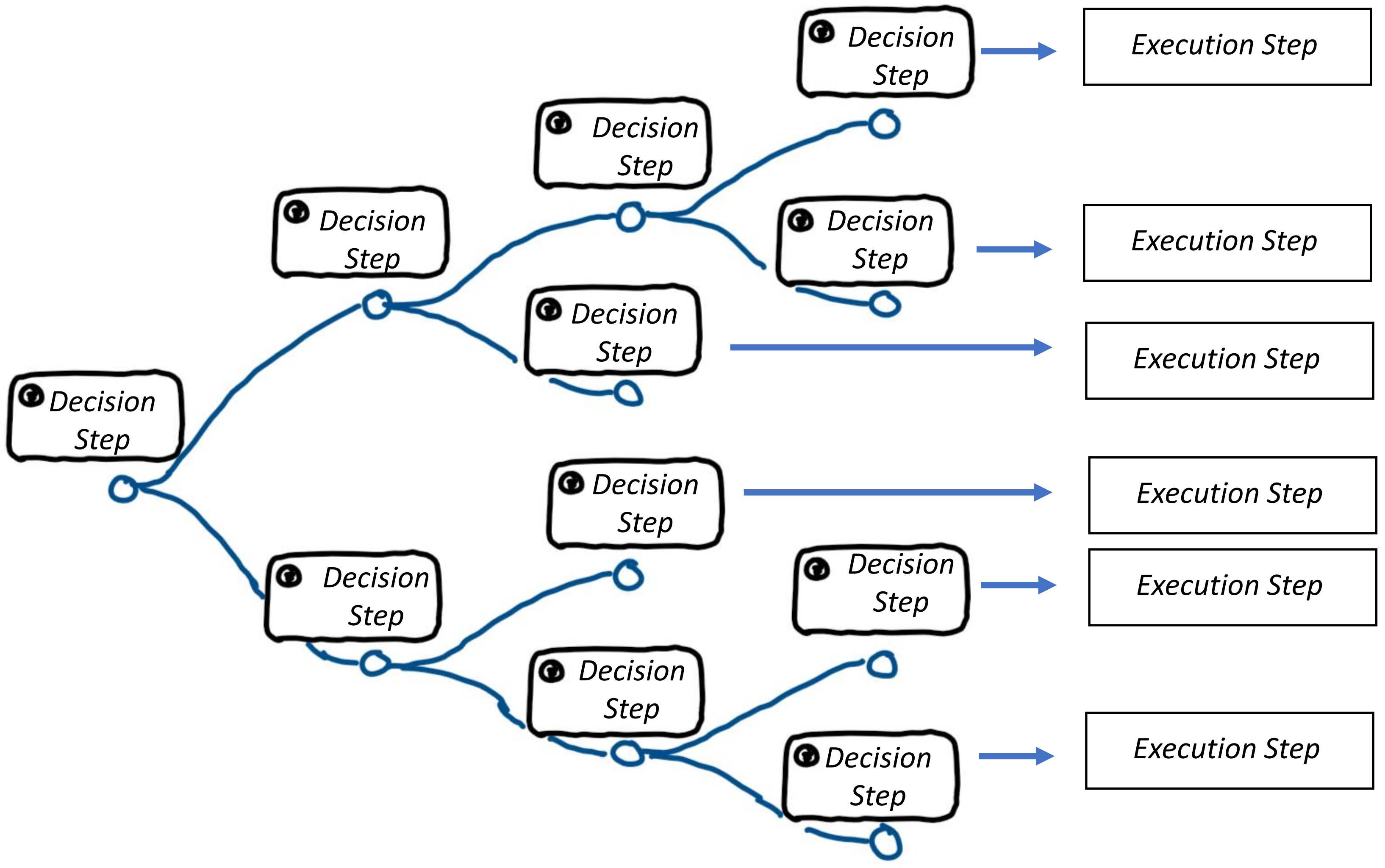
Congratulation

Function Calling

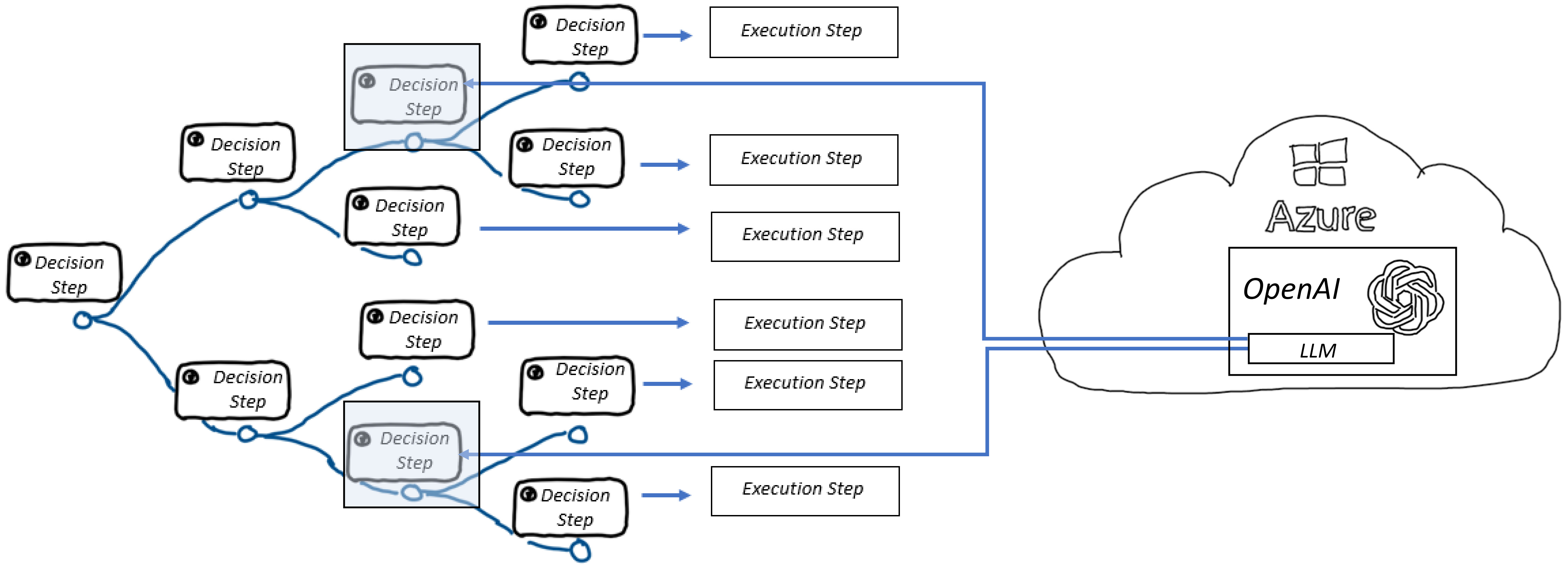
Tool Use



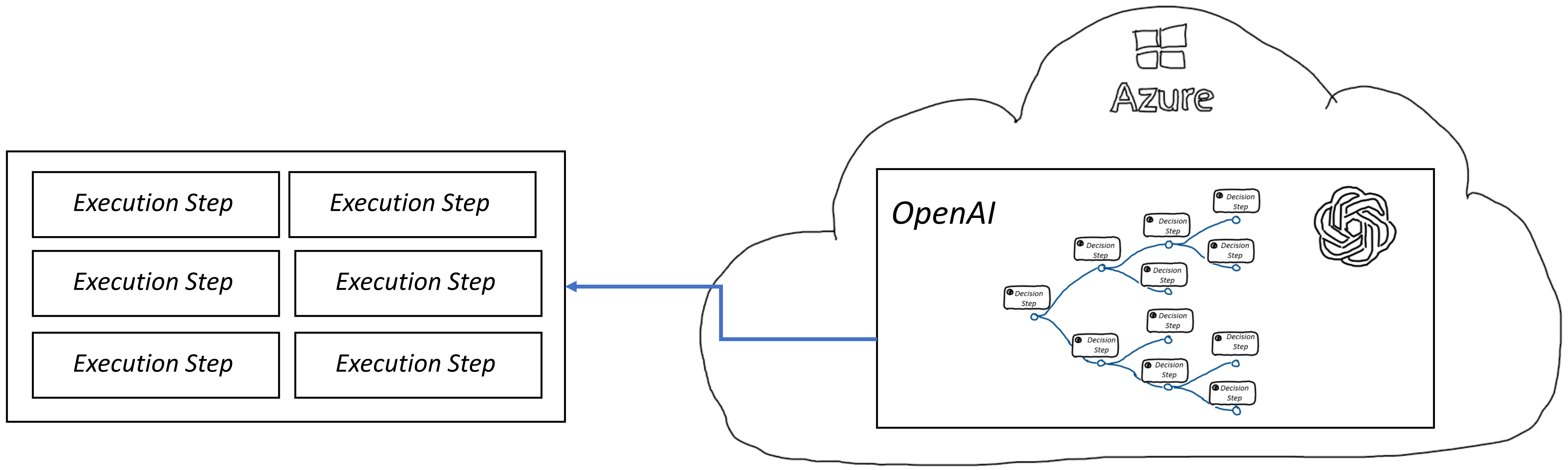
Function Calling



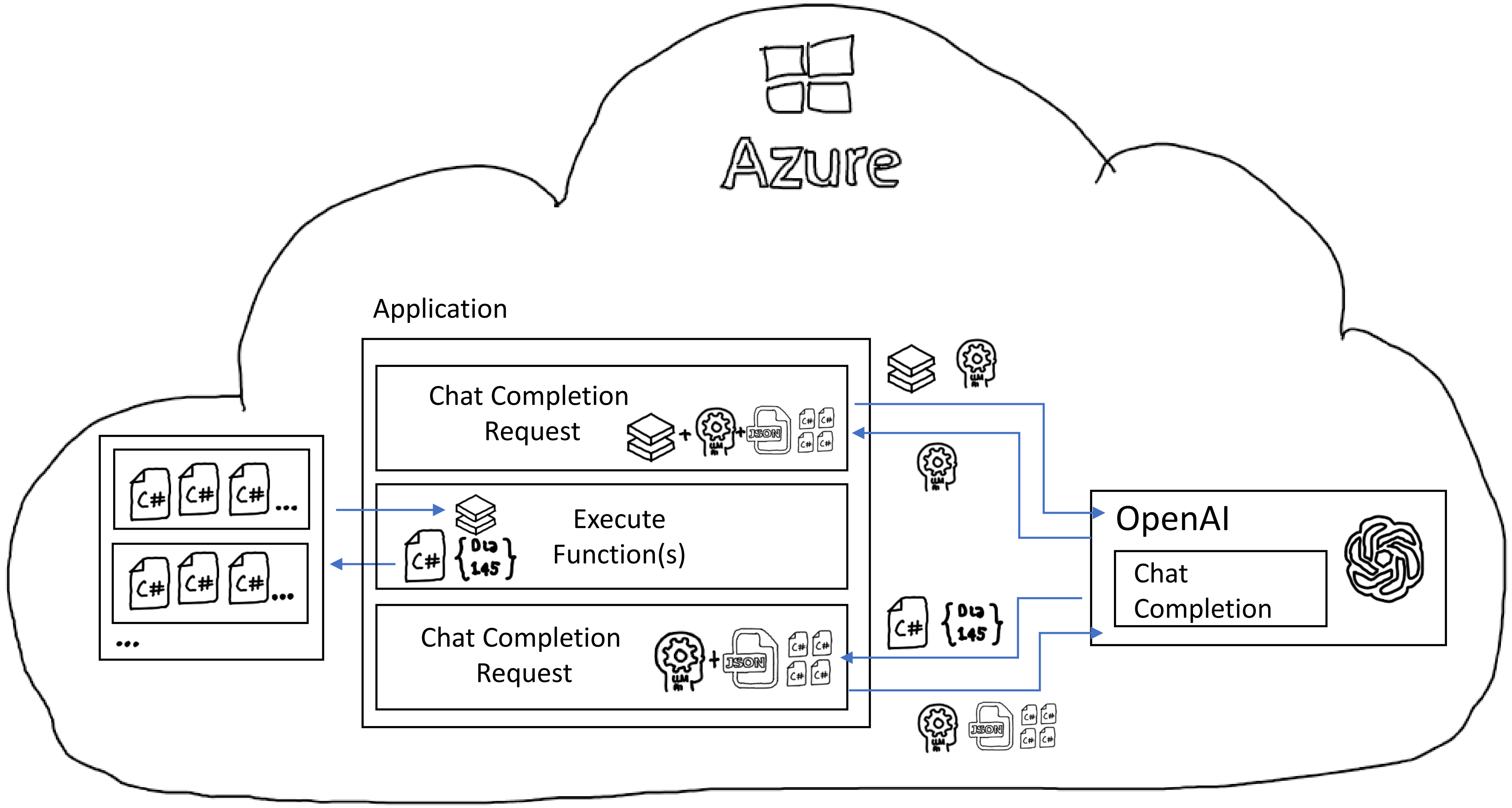
Function Calling



Function Calling



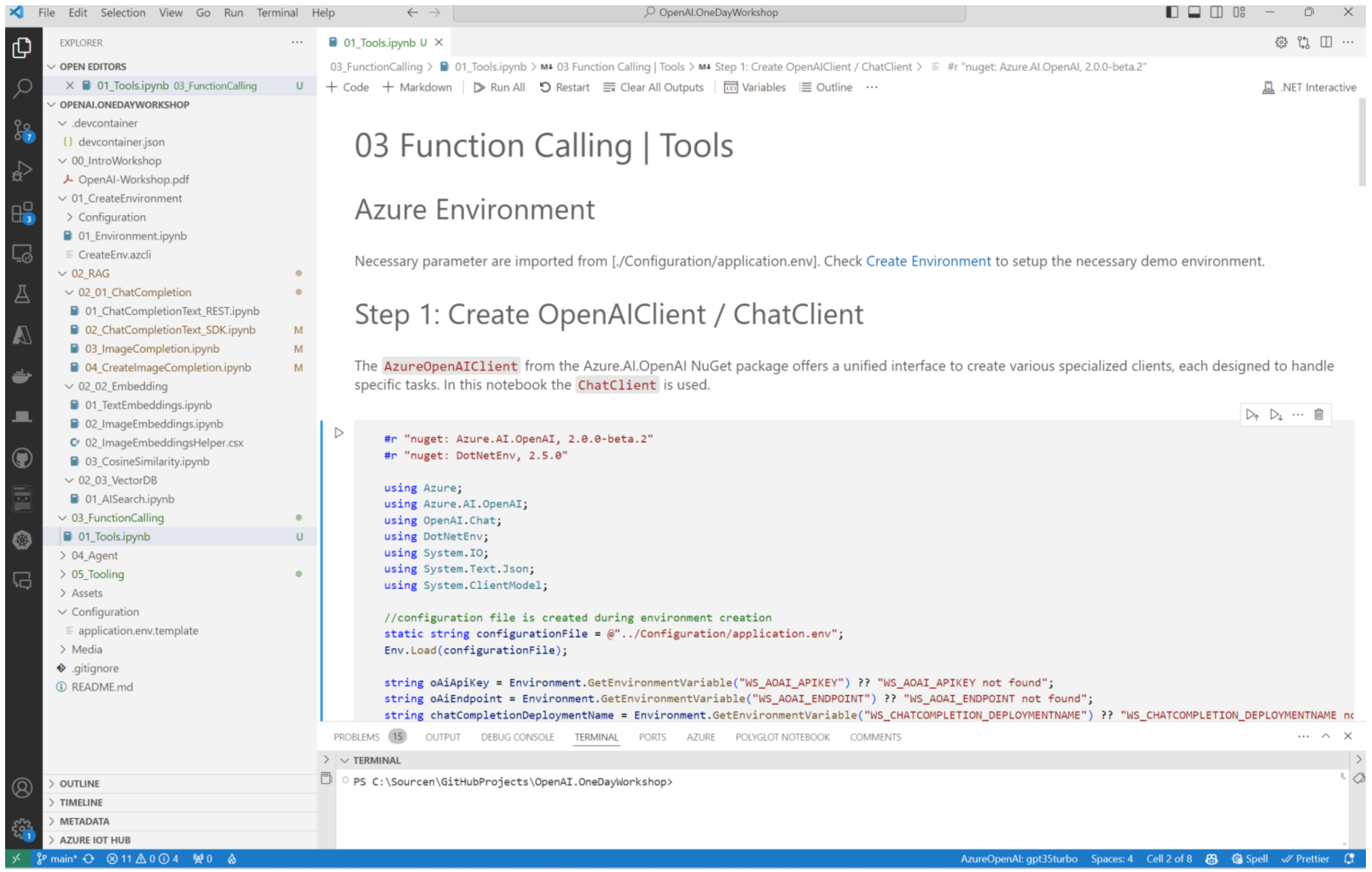
LLM Function Calling / Tools



LLM Function Calling / Tools

Function Calling

/03_FunctionCalling/01_toolsipynb



03 Function Calling | Tools

Azure Environment

Necessary parameter are imported from [./Configuration/application.env]. Check Create Environment to setup the necessary demo environment.

Step 1: Create OpenAIClient / ChatClient

The `AzureOpenAIClient` from the Azure.AI.OpenAI NuGet package offers a unified interface to create various specialized clients, each designed to handle specific tasks. In this notebook the `ChatClient` is used.

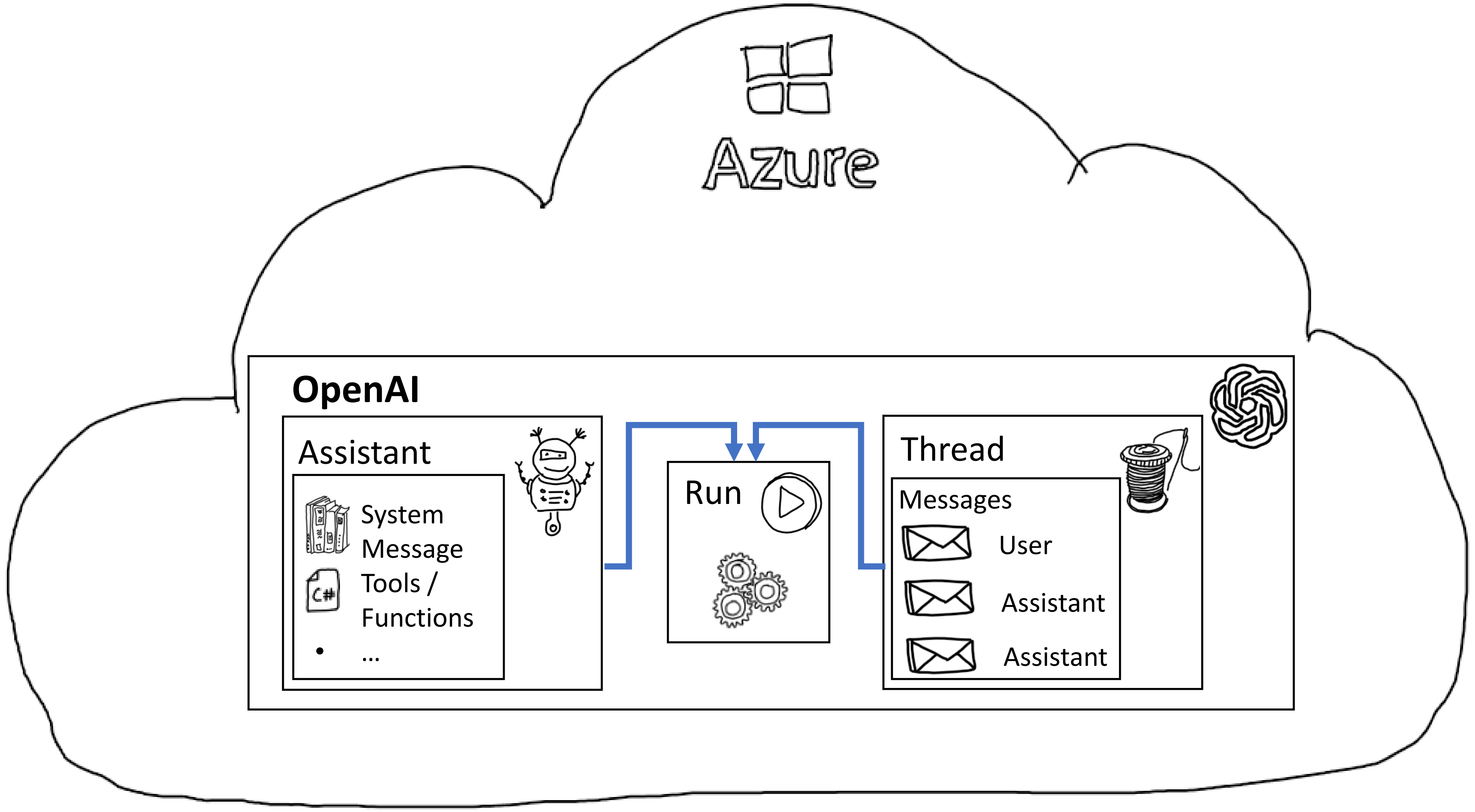
```
#r "nuget: Azure.AI.OpenAI, 2.0.0-beta.2"
#r "nuget: DotNetEnv, 2.5.0"

using Azure;
using Azure.AI.OpenAI;
using OpenAI.Chat;
using DotNetEnv;
using System.IO;
using System.Text.Json;
using System.ClientModel;

//configuration file is created during environment creation
static string configurationFile = @"../Configuration/application.env";
Env.Load(configurationFile);

string oAiApiKey = Environment.GetEnvironmentVariable("WS_AOAI_APIKEY") ?? "WS_AOAI_APIKEY not found";
string oAiEndpoint = Environment.GetEnvironmentVariable("WS_AOAI_ENDPOINT") ?? "WS_AOAI_ENDPOINT not found";
string chatCompletionDeploymentName = Environment.GetEnvironmentVariable("WS_CHATCOMPLETION_DEPLOYMENTNAME") ?? "WS_CHATCOMPLETION_DEPLOYMENTNAME nc
```

```
PS C:\Sourcen\GitHubProjects\OpenAI.OneDayWorkshop>
```

Get to work

Congratulation

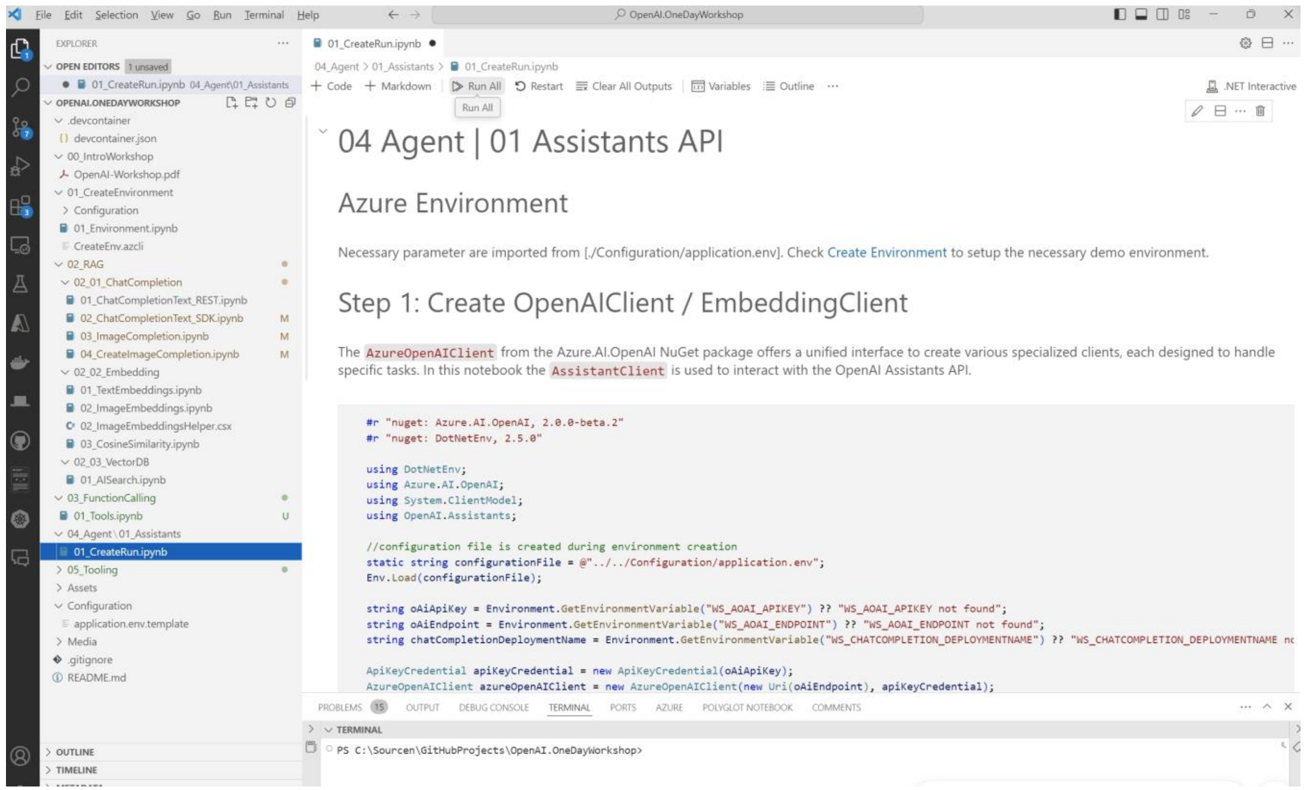
Assistants API



Assistants API

Assistants API

/04_Agent/01_Assistants/01_CreateRun.ipynb



04 Agent | 01 Assistants API

Azure Environment

Necessary parameter are imported from [./Configuration/application.env]. Check Create Environment to setup the necessary demo environment.

Step 1: Create OpenAIClient / EmbeddingClient

The `AzureOpenAIClient` from the Azure.AI.OpenAI NuGet package offers a unified interface to create various specialized clients, each designed to handle specific tasks. In this notebook the `AssistantClient` is used to interact with the OpenAI Assistants API.

```
#r "nuget: Azure.AI.OpenAI, 2.0.0-beta.2"
#r "nuget: DotNetEnv, 2.5.0"

using DotNetEnv;
using Azure.AI.OpenAI;
using System.ClientModel;
using OpenAI.Assistants;

//configuration file is created during environment creation
static string configurationFile = @"../../Configuration/application.env";
Env.Load(configurationFile);

string oAiApiKey = Environment.GetEnvironmentVariable("WS_AOAI_APIKEY") ?? "WS_AOAI_APIKEY not found";
string oAiEndpoint = Environment.GetEnvironmentVariable("WS_AOAI_ENDPOINT") ?? "WS_AOAI_ENDPOINT not found";
string chatCompletionDeploymentName = Environment.GetEnvironmentVariable("WS_CHATCOMPLETION_DEPLOYMENTNAME") ?? "WS_CHATCOMPLETION_DEPLOYMENTNAME nc

ApiKeyCredential apiKeyCredential = new ApiKeyCredential(oAiApiKey);
AzureOpenAIClient azureOpenAIClient = new AzureOpenAIClient(new Uri(oAiEndpoint), apiKeyCredential);
```

```
PS C:\Sourcen\GitHubProjects\OpenAI.OneDayWorkshop>
```

Congratulation

Tools

Semantic Kernel

Plugins

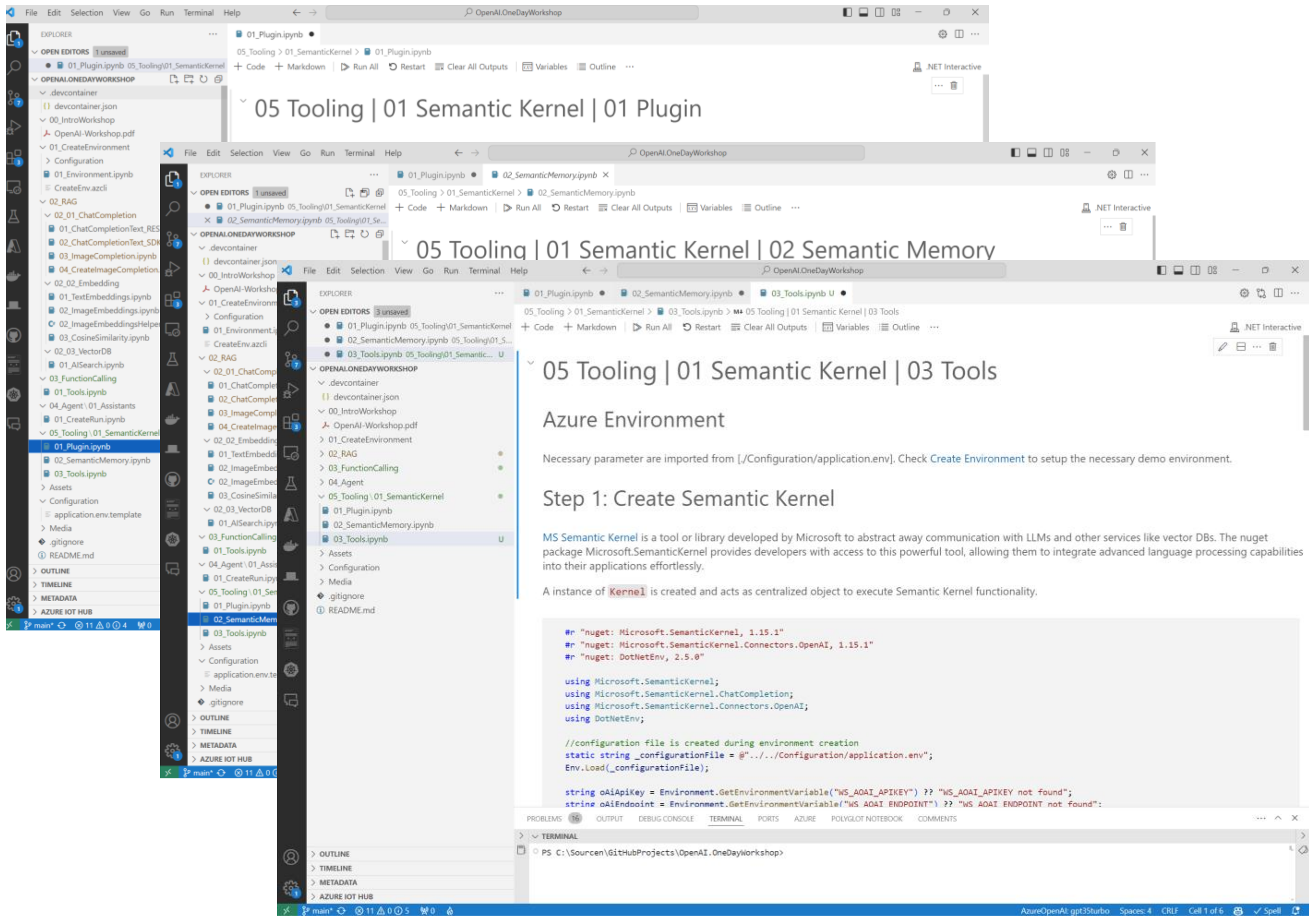
/05_Tooling/01_SemanticKernel/01_Plugin.ipynb

Semantic Memory

/05_Tooling/01_SemanticKernel/02_SemanticMemory.ipynb

Tools

/05_Tooling/01_SemanticKernel/03_Tools.ipynb



05 Tooling | 01 Semantic Kernel | 01 Plugin

05 Tooling | 01 Semantic Kernel | 02 Semantic Memory

05 Tooling | 01 Semantic Kernel | 03 Tools

Azure Environment

Necessary parameter are imported from [./Configuration/application.env]. Check Create Environment to setup the necessary demo environment.

Step 1: Create Semantic Kernel

MS Semantic Kernel is a tool or library developed by Microsoft to abstract away communication with LLMs and other services like vector DBs. The nuget package Microsoft.SemanticKernel provides developers with access to this powerful tool, allowing them to integrate advanced language processing capabilities into their applications effortlessly.

A instance of `Kernel` is created and acts as centralized object to execute Semantic Kernel functionality.

```
#r "nuget: Microsoft.SemanticKernel, 1.15.1"
#r "nuget: Microsoft.SemanticKernel.Connectors.OpenAI, 1.15.1"
#r "nuget: DotNetEnv, 2.5.0"

using Microsoft.SemanticKernel;
using Microsoft.SemanticKernel.ChatCompletion;
using Microsoft.SemanticKernel.Connectors.OpenAI;
using DotNetEnv;

//configuration file is created during environment creation
static string _configurationFile = @"../../Configuration/application.env";
Env.Load(_configurationFile);

string oAiApiKey = Environment.GetEnvironmentVariable("WS_AOAI_APIKEY") ?? "WS_AOAI_APIKEY not found";
string oAiEndpoint = Environment.GetEnvironmentVariable("WS_AOAI_ENDPOINT") ?? "WS_AOAI_ENDPOINT not found";
```

```
PS C:\Sourcen\GitHubProjects\OpenAI.OneDayWorkshop>
```

Summary

Closing