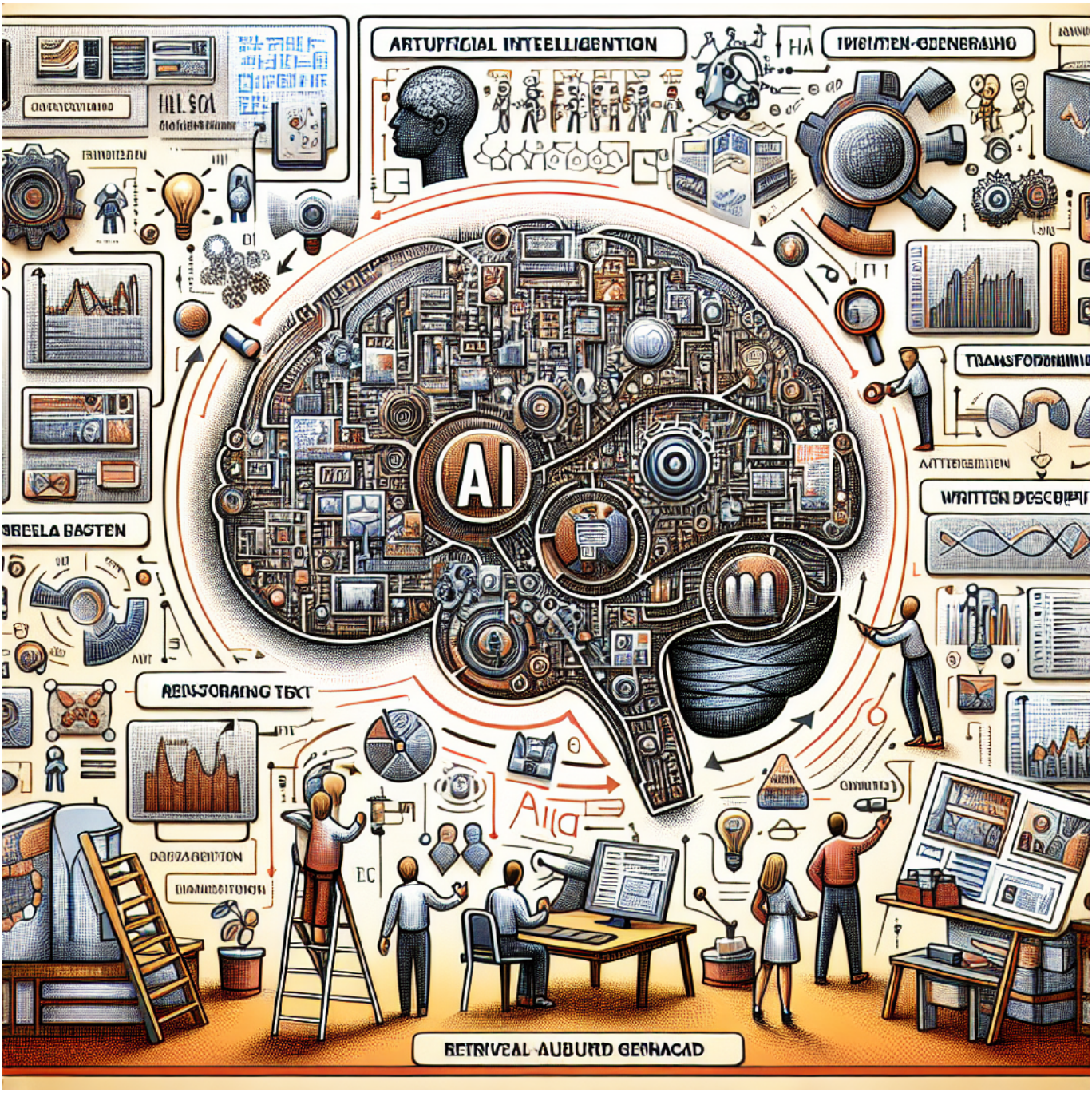
Semantic Kernel Cookbook



With the rise of LLM, AI has entered the 2.0 era. Compared with previous AI technologies, the threshold has been lowered and the applicability has been enhanced. It is no longer limited to the field of data science, and more different types of jobs and roles of people are participating in large-scale research in the application scenarios of the model. For how traditional engineering projects or enterprise applications to enter the field of LLM, a framework is an important key. Especially for traditional projects, companies must think about how to access LLM faster and at low cost. In 2023, the first year of LLM, the open source community has a lot of frameworks and solutions based on LLM applications. I personally like LangChain, BentoML, prompt flow , autogen and Semantic Kernel. But overall, Semantic Kernel is more suitable for traditional engineering and multi-language engineering teams, LangChain is suitable for data science personnel, and BenToML is suitable for multi-model deployment scenarios. In December 2023, when Semantic Kernel officially releases 1.0.1 based on .NET version, I also hope to use this manual to give you

some ways to learn and get started. Although Semantic Kernel still has many imperfections, it does not prevent everyone from learning and using it.

Session	Intro	.NET Samples	Python Samples
Getting started with LLM	Begin to know LLM, including OpenAI, Azure OpenAI Service and LLM on Hugging face		
Using Azure OpenAI Service With SDK	Learn how to use Azure OpenAI Service with SDK	Click	Click
Foundations of Semantic Kernel	What is Semantic Kernel? What are its advantages and disadvantages? Semantic Kernel related concepts, etc.	Click	Click
The skills of LLM - Plugins	We know that communicating with LLM requires the use of prompt engineering? For enterprise applications, there are many business-oriented prompt engineering. In Semantic Kernel we call it Plugins. In this session we will introduce how to use Semantic Kernel Plugins and how to define your own Plugins	Click	Click
Planner - Let LLM have planning work	Human beings need to complete a job step by step, and the same goes for LLMs. Semantic Kernel has a very powerful task planning capability - Planner, in this session we will explain in detail how to define and use Planner to make your application more intelligent	Click	Click
Embedding Skills	Building RAG applications is the most commonly used LLM solution at this stage. It is very convenient to build RAG applications through Semantic Kernel This session will tell you how to use Semantic Kernel Embeddings	Click	Click
HandsOnLab	Through three hands on labs projects, let everyone truly understand the application of Semantic Kernel	Click	Click

如果你需要中文，请 点击这里

Getting started with LLM

Since the 1950s, humans have begun to explore AI. Artificial intelligence has always given people the impression of being an epoch-making technology that requires practitioners in the field of data science with professional skills to use it. But starting from the end of 2022, the field of AI has undergone major changes. Since OpenAI released the GPT-3 model, artificial intelligence is no longer a solution for a single field or single scenario. The new multi-modal LLM changes the rules of the game. No matter what kind of work you are engaged in, you can use natural language to communicate with the LLM. The large language model feeds back to you generative content. This It includes text, pictures, videos, etc., greatly enhancing usability. Before entering the content of Semantic Kernel, I hope to tell you stories related to LLM models so that you can better understand LLM models.

Basic Concept of LLM

Large Language Model (LLM), also known as large language model, is an artificial intelligence model designed to understand and generate human language. They are trained on large amounts of text data and can perform a wide range of tasks, including text summarization, translation, sentiment analysis, and more. Large language models are characterized by their large size and contain billions of parameters, helping them learn complex patterns in language data. These models are often based on deep learning architectures as well as the Transformer algorithm. Large language model models are trained through self-supervised learning, which generates its own labels for the input data by predicting the next word or token in the sequence, given the previous words. The training process consists of two main steps: pre-training and fine-tuning. During the pre-training phase, the model learns from a large, diverse dataset, often containing billions of words from different sources, such as websites, books, and articles. In the fine-tuning phase, the model is further trained on a more specific and smaller data set related to the target task or domain, which helps the model fine-tune its understanding and adapt to the special requirements of the task. Many companies are now developing LLMs. The well-known ones include OpenAI's GPT-X and DALLE-X series, Meta's LLama, Google's Gemini, and Baidu's ERNIE. GPT-4 in OpenAI is the best LLM at this stage and has great advantages. But as time goes by, many good industry vertical field models have been born.

Transformer

The Transformer algorithm is a deep learning model based on the self-attention mechanism, which can be used to process natural language and other sequence data. It consists of two parts: an encoder and a decoder. Each part contains multiple layers, and each layer contains multi-head self-attention and feed-forward neural networks. The advantage of the Transformer algorithm is that it can process the entire sequence in parallel and capture long-distance dependencies without using recurrent neural networks or convolutional neural networks. The Transformer algorithm was originally proposed in the paper "Attention Is All You Need"1 for machine translation tasks. Later, it was widely used in other natural language processing tasks, such as text summarization, question answering, speech recognition, etc. Some famous Transformer-based models are BERT, GPT-3/4, T5, etc.

This series mainly focuses on OpenAI 3/3.5/4 and DALLE-3 of Azure OpenAI Services. As for other models, we will mention them in more advanced content in the future. You can follow my GitHub Repo.

Introduction to OpenAI and OpenAI's models

Although Transformer's algorithm comes from Google, it is OpenAI that really brings LLMs into the public eye. OpenAI is an artificial intelligence research and deployment company, and its mission is to ensure that artificial general intelligence (AGI) can benefit all mankind. Its vision is to create an AGI that can cooperate and compete with humans while adhering to human values and ethics. OpenAI was originally a non-profit organization founded in 2015 by some well-known figures in the technology industry, such as Elon Musk, Peter Thiel, Jerry Yang, etc.1. Its goal is to advance the development of digital intelligence so that it can benefit humanity to the greatest extent without being bound by monetary interests. Its research is open and transparent and can be accessed and used by anyone. OpenAI has pioneered research in the field of artificial intelligence, especially in generative models and security. It has developed some powerful large language models, such as GPT-3/3.5/4, ChatGPT, DALL·E, Whisper, etc., which can understand and generate various forms of data such as text, images, sounds, etc. It also seeks to explore the potential risks and impacts of AGI and how they can be aligned with human goals and interests.

GPT Models

GPT (Generative Pre-trained Transformer, Generative Pre-trained Transformer) is a natural language processing (NLP) model based on deep learning, developed by OpenAI. The GPT series of models is known for its power and flexibility, and performs well in a variety of language tasks. The following are some key features and development history of the GPT model:

Core features of GPT

1. Based on Transformer architecture: The GPT model is based on Transformer architecture, which is a deep learning model particularly suitable for processing sequence data (such as text).
2. Large-scale pre-training: GPT learns the common patterns and structures of language by pre-training on large amounts of text data. This includes text from various sources such as books, web pages, news articles, etc.
3. Fine-tuning application: After pre-training, the GPT model can be fine-tuned for specific tasks, such as question and answer, text generation, translation, etc.
4. Context-sensitive: The GPT model is able to understand and generate context-sensitive text, which makes it outstanding in generating coherent and relevant content.

GPT development history

GPT-1: A first release that demonstrates the potential of pre-training on large-scale unlabeled data and the effectiveness of fine-tuning on a variety of tasks.

GPT-2: Increases the model size and training data volume, significantly improving the quality and accuracy of text generation. GPT-2 has attracted widespread attention for its ability to generate text that is coherent and sometimes indistinguishable from human-written text.

GPT-3: Further expanded the model size, reaching an unprecedented 175 billion parameters. GPT-3 achieves revolutionary performance on multiple NLP tasks, especially when little or no fine-tuning is possible.

GPT-4 and beyond: With the continuous development of technology, subsequent GPT models may continue to improve in terms of model scale, understanding capabilities, and multi-modal capabilities.

GPT application areas GPT models are widely used in many fields, including but not limited to:

Text generation: such as article writing, creative writing, code generation, etc.

Chatbots: Provide a smooth conversational experience.

Natural language understanding: such as sentiment analysis, text classification, etc.

Translation and multilingual tasks: automatically translate different languages.

Knowledge extraction and question answering: Extract information from large amounts of text to answer specific questions.

Overall, the GPT model represents an important milestone in the current field of artificial intelligence and natural language processing. Its powerful capabilities and diverse application prospects continue to lead the trend of technological development.

GPT-3

GPT-3 is a LLMs developed by OpenAI that can understand and generate natural language. It is one of the LLMs currently, with 175 billion parameters, and can complete text summarization, machine translation, dialogue systems, code generation, etc. The characteristic of GPT-3 is that it can adapt to different tasks and fields through simple text prompts, that is, "few-shot learning", without requiring additional fine-tuning or labeling data. GPT-3 opened Pandora's box and changed the rules of the industry. GPT-3 has been used in many products and services, such as OpenAI API, OpenAI Codex, early GitHub Copilot, etc., which can make it easier for developers, creators, and scholars to use and learn artificial intelligence. GPT-3 has also triggered some discussions and thinking about the ethics, society and security of artificial intelligence, such as artificial intelligence's bias, explainability, responsibility, impact, etc.

GPT-3.5 and ChatGPT

GPT-3.5 and ChatGPT are both large language models based on the GPT-3 architecture, which can understand and generate natural language. They all have 175 billion parameters and can perform amazingly well on a variety of language processing tasks, such as text summarization, machine translation, dialogue systems, code generation, etc.

The main difference between GPT-3.5 and ChatGPT is their scope and purpose. GPT-3.5 is a general language model that can handle a variety of language processing tasks. ChatGPT, on the other hand, is a specialized model designed specifically for chat applications. It emphasizes interaction and communication with users, and can play different roles, such as cat ladies, celebrities, politicians, etc. It can also generate multimedia content such as images, music, and videos based on user input.

Another difference between GPT-3.5 and ChatGPT is their training data and training methods. GPT-3.5 is pre-trained on 570 GB of text data from different sources such as websites, books, articles, etc. It is trained through self-supervised learning, which generates its own labels for the input data by predicting the next word or token in the sequence, given the previous words. ChatGPT is based on GPT-3.5 and uses more conversation data, such as social media, chat records, movie scripts, etc., for further fine-tuning. Its training

method is through multi-task learning, that is, optimizing multiple goals at the same time, such as language model, dialogue generation, emotion classification, image generation, etc.

GPT-4

GPT-4 (Generative Pre-trained Transformer 4th Generation) is the latest generation of artificial intelligence language models developed by OpenAI. It is the successor of GPT-3 with more advanced and refined features. Here are some of the key features of GPT-4:

Larger knowledge base and data processing capabilities: GPT-4 can process larger amounts of data, and its knowledge base is broader and deeper than GPT-3.

Higher language understanding and generation capabilities: GPT-4 has significantly improved in understanding and generating natural language, and can more accurately understand complex language structures and meanings.

Multi-modal capabilities: GPT-4 can not only process text, but also understand and generate images, providing a multi-modal interactive experience.

Better contextual understanding: GPT-4 can better understand and maintain context in long conversations, providing more coherent and consistent responses.

Improved security and reliability: OpenAI has strengthened the filtering and control of inappropriate content in GPT-4 to provide a more secure and reliable user experience.

Wide range of application fields: GPT- can be used in various fields, including but not limited to chatbots, content creation, educational assistance, language translation, data analysis, etc.

Overall, GPT-4 has made significant improvements and enhancements over its predecessor model, providing more powerful and diverse features. GPT-4 has an absolute leadership position at this stage and is also the goal of many companies' large models.

GPT-4V

The full name of GPT-4V is GPT-4 Turbo with Vision. It can understand pictures, analyze pictures for users, and answer questions related to pictures. GPT-4V can accurately understand the content of images, identify objects in images, count the number of objects, provide image-related insights and information, extract text, etc. It can be said that GPT-4V is the king of LLMs, and it also allows LLMs to better understand the world. GPT-4V's main vision capabilities and application directions

Object Detection: GPT-4V is able to identify and detect a variety of common objects in images, such as cars, animals, and household items. Its recognition capabilities have been evaluated on standard image datasets.

Text Recognition: This model features optical character recognition (OCR) technology that finds printed or handwritten text in images and converts it into machine-readable text. This feature is proven in images such as documents, logos, and titles.

Face Recognition: GPT-4V is able to find and recognize faces in images. It also has a degree of ability to determine gender, age and racial attributes from facial features. The model's facial analysis capabilities have been tested on datasets such as FairFace and LFW.

CAPTCHA SOLVING: GPT-4V demonstrates visual reasoning capabilities in solving text- and image-based CAPTCHAs. This indicates that the model has advanced puzzle-solving skills.

Geolocation: GPT-4V is able to identify cities or geographical locations represented in landscape images. This shows that the model has mastered knowledge about the real world, but it also means that there is a risk of privacy leakage.

Complex Images: The model performs poorly when dealing with complex scientific diagrams, medical scans, or images with multiple overlapping text components. It cannot grasp contextual details.

DALL·E

DALL·E is an advanced artificial intelligence program developed by OpenAI specifically designed to generate images. It is a neural network model based on the GPT-3 architecture, but unlike GPT-3, which mainly processes text, DALL·E's expertise lies in generating corresponding images based on text descriptions. The name of this model is a tribute to the famous artist Salvador Dalí and the popular animated character WALL·E.

Key features of DALL·E Text to image conversion: DALL·E can generate images based on text descriptions provided by users. These descriptions can be very specific or creative, and the model will do its best to generate images that match the description.

Creativity and Flexibility: DALL·E displays amazing creativity when generating images, able to combine different concepts and elements to create unique and innovative visual works.

Variety and detail: The model is capable of generating multiple styles and types of images and can handle complex, detailed descriptions.

Application potential: DALL·E has extensive application potential in art creation, advertising, design and other fields.

DALL·E application scenarios include

Artistic Creation: Artists and designers can use DALL·E to explore new ideas and visual expressions.

Advertising and Media: Generate images that fit a specific theme or concept.

Education and Entertainment: Used in the production of instructional materials or the creation of entertainment content.

Research and exploration: Explore the possibilities of artificial intelligence in the field of visual arts.

The emergence of DALL·E marks an important progress in artificial intelligence in creative tasks and shows the huge potential of AI in the field of visual arts. Now the latest DALL·E model is the DALL·E 3 .

Whisper·

Whisper is an advanced automatic speech recognition (ASR) model developed by OpenAI. This model focuses on transcribing speech into text and has shown excellent performance in multiple languages and different environments. Here are some key features about the Whisper model:

Features

Multi-language support: Whisper models are capable of handling many different languages and dialects, making them widely applicable across the globe.

High-precision recognition: It can accurately recognize and transcribe speech, maintaining a high accuracy even in environments with a lot of background noise.

Adaptable to different contexts: Whisper can not only recognize standard voice input, but also adapt to various colloquial and informal conversation styles.

Easy to integrate and use: As a machine learning model, Whisper can be integrated into various applications and services to provide speech recognition capabilities.

Application

Automatic subtitles and transcription: Automatically generate subtitles or text for video and audio content.

Voice assistants and chatbots: Improve the ability of voice assistants and chatbots to recognize voice commands.

Accessibility Services: Help people with hearing impairments better understand audio content.

Meeting and Lecture Recording: Automatically record and transcribe meeting or lecture content.

Overall, the Whisper model represents an important advancement in the field of automatic speech recognition, and its multi-language and high-precision recognition capabilities make it extremely valuable in a variety of application scenarios.

Microsoft & OpenAI

The partnership between Microsoft and OpenAI is an important development in contemporary artificial intelligence. Microsoft has been an important partner and supporter of OpenAI since its inception. Here are some key aspects and impacts of their collaboration:

Investment and Cooperation

Financial support: Microsoft made significant investments in OpenAI in its early days, including hundreds of millions of dollars in funding. These investments help OpenAI develop its research projects and technology.

Cloud computing resources: Microsoft provides OpenAI with the resources of its Azure cloud computing platform, which is crucial for training and running large AI models, such as GPT and DALL·E series models.

Technical cooperation

Joint research and development: The two companies have cooperated on multiple AI projects and technologies to jointly promote the development of artificial intelligence.

Product integration: Some of OpenAI's technologies, such as GPT-3, have been integrated into Microsoft products and services, such as Microsoft Azure and other enterprise-level solutions.

Strategic Cooperation

Sustainable and safe AI: Both parties are committed to developing AI technology that is both sustainable and safe, and pay attention to AI ethics and safety issues.

Expand AI applications: Through cooperation, the two companies are committed to applying AI technology to a wider range of fields, such as health care, education, and environmental protection.

Influence

Accelerate the development of AI technology: This cooperation promotes the rapid development and innovation of AI technology.

Business applications and services: Microsoft has promoted the widespread application of artificial intelligence in the business field by applying OpenAI's technology to its products and services.

Promote AI democratization: This collaboration helps make advanced AI technology accessible and usable to more enterprises and developers.

Overall, the cooperation between Microsoft and OpenAI is a model of combining technological innovation and commercial applications. This cooperation has had a profound impact on the development and popularization of artificial intelligence technology. As cooperation between the two parties continues to deepen, it can be expected that they will continue to play an important role in the field of artificial intelligence.

Azure OpenAI Service

Azure OpenAI Service is a collaboration between Microsoft Azure and OpenAI. Azure OpenAI Service is a cloud-based platform that enables developers and data scientists to quickly and easily build and deploy artificial intelligence models. With Azure OpenAI, users can access a variety of AI tools and technologies to create intelligent applications, including natural language processing, computer vision, and deep learning. Azure OpenAI Service is designed to accelerate the development of AI applications, allowing users to focus on creating innovative solutions that create value for their organizations and customers.

Azure OpenAI Service provides REST API access to OpenAI's powerful language models, including GPT-4, GPT-4 Turbo with Vision, GPT-3.5-Turbo, and the family of embedded models. Additionally, the new GPT-4 and GPT-3.5-Turbo model series are now officially released. These models can be easily adapted to specific tasks, including but not limited to content generation, aggregation, image understanding, semantic

search, and natural language to code conversion. Users can access the service through a REST API, Python SDK, or a web-based interface in Azure OpenAI Studio.

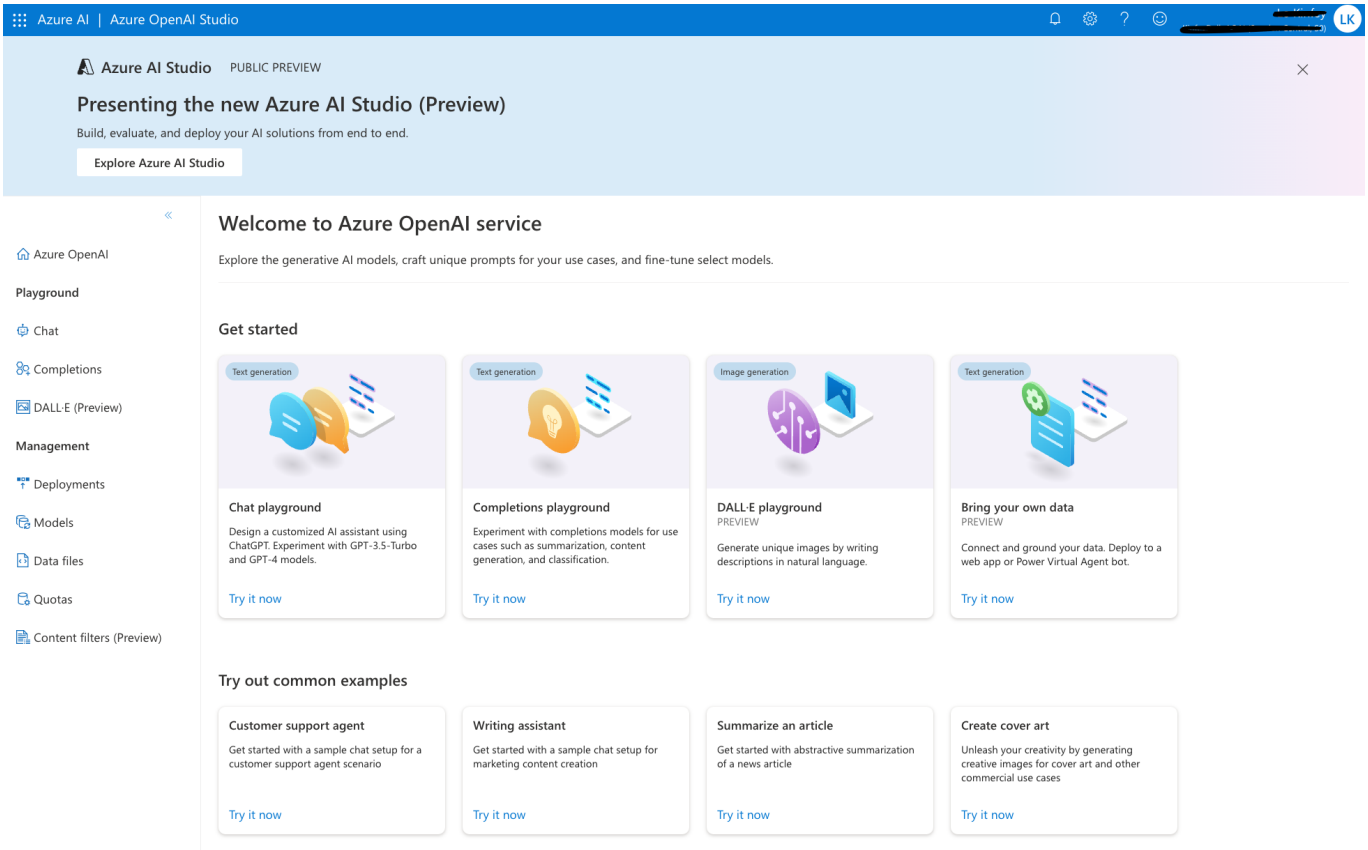
To use Azure OpenAI Service, you need to have an Azure account, then apply through this link, and wait 1-3 working days to use Azure OpenAI Service.

Azure OpenAI Studio

We can manage our models through Azure OpenAI Studio, as well as test our models in the Playground

Azure OpenAI Studio

We can manage our models through Azure OpenAI Studio, as well as test our models in the Playground



Note: All examples are based on Azure OpenAI Services

Hugging Face

Hugging Face is an artificial intelligence research company focusing on natural language processing (NLP), known for its open source projects and innovations in the field of NLP. The company was founded in 2016 and its headquarters are in New York, but it has a global presence and activities.

Products

Transformers library: Hugging Face's most famous contribution is its development of the "Transformers" library, which is a widely used Python library that contains a variety of pre-trained NLP models, such as BERT, GPT, T5, etc. This library makes it easier to access and use these complex models, and has a significant impact on promoting research and applications in the NLP field.

Model Sharing and Community: Hugging Face has built a strong community that promotes model and knowledge sharing among researchers and developers. Through its platform, anyone can upload, share and use pre-trained models.

Research and Collaboration: Hugging Face conducts active research in the field of artificial intelligence, collaborating with numerous teams in academia and industry.

Education and Resources: Hugging Face also provides a variety of educational resources, including tutorials, documentation, and research papers, to help people better understand and use NLP technology.

Influence

Technological innovation: Hugging Face has played an important role in promoting technological innovation in the field of NLP, especially in the development and application of pre-trained models.

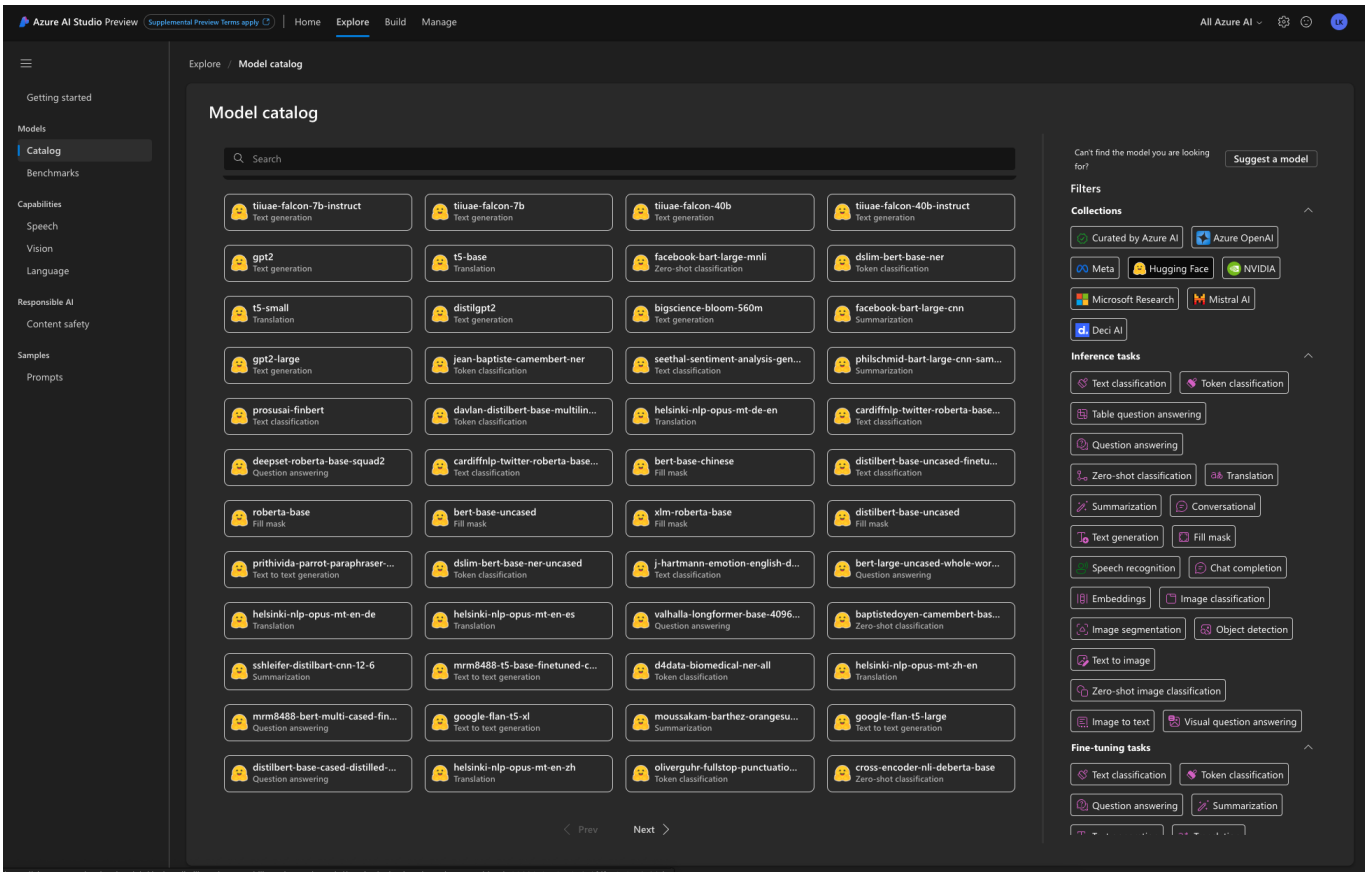
Lowering the technical threshold: By providing easy-to-use tools and resources, Hugging Face lowers the technical threshold for working in the field of NLP, allowing more researchers and developers to participate in this field.

Community building: Its strong community and open source culture promote knowledge sharing and collaboration, accelerating the development and innovation of NLP technology.

While Hugging Face originally started as a consumer-facing chatbot application, it quickly transformed into a company focused on providing NLP technology and resources. Now, it not only supports research and education, but also provides commercial solutions to enterprises, such as custom model training, data processing and machine learning consulting services.

In summary, Hugging Face is a key player in the NLP field, and its open source ethos and contributions to the community have played an important role in promoting the democratization and innovation of artificial intelligence technology.

Azure AI Studio also supports the introduction of the Hugging Face model, which allows enterprises to better combine business scenarios and use different models to solve problems in different application scenarios.



Summary

This chapter introduces the current knowledge related to LLM, especially the knowledge related to mainstream large-scale language model platforms such as OpenAI, Microsoft, and Hugging Face, as well as the application scenarios and performance of different models. For application scenarios, it is impossible for us to use only one model. In the AI 2.0 era, we need the support of different models to complete more intelligent application scenarios. Whether in the cloud or locally, the application scenarios of large language models will be a hot topic of concern in the next few years. As a beginner, what you need to do is to understand different models and complete application construction based on actual scenarios.

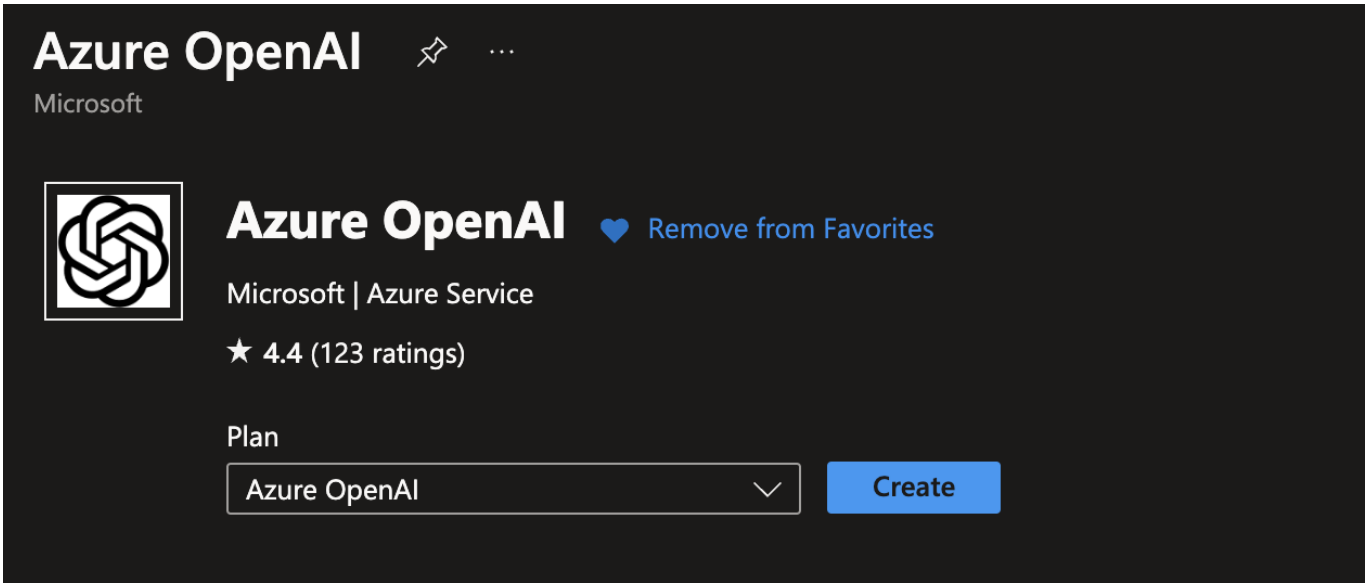
Using Azure OpenAI Service With SDK

In the preface, we learned about LLMs. Now I want to talk about how to use LLMs. Before Learning Semantic Kernel, I would like you to see how to correctly accessAzure OpenAI Service through the SDK.

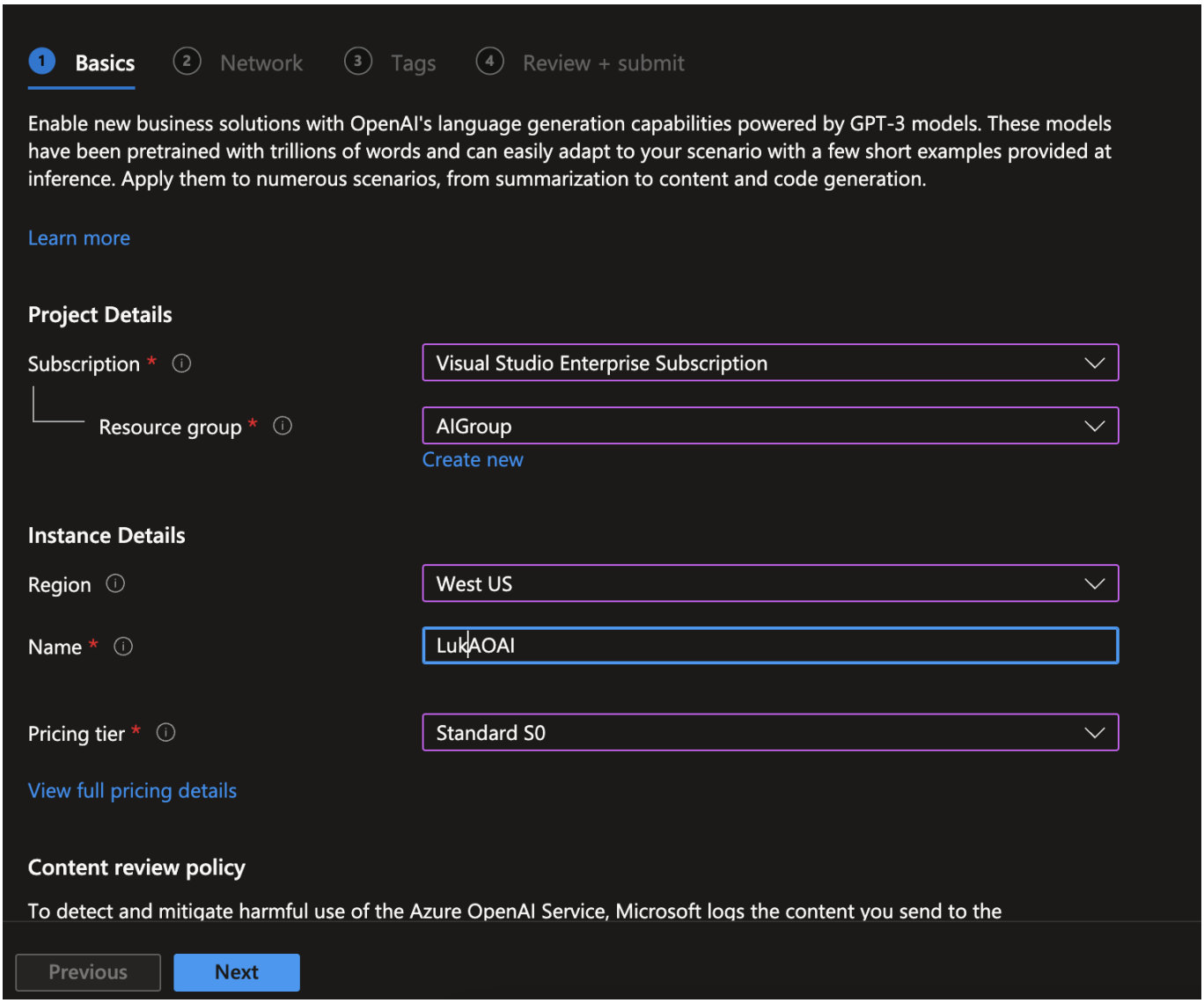
Deploy the model in Azure OpenAI Studio

Deploying an Azure OpenAI model is very easy. After successfully applying for Azure OpenAI Service, you can deploy it by creating resources in Azure Portal. Here are the steps:

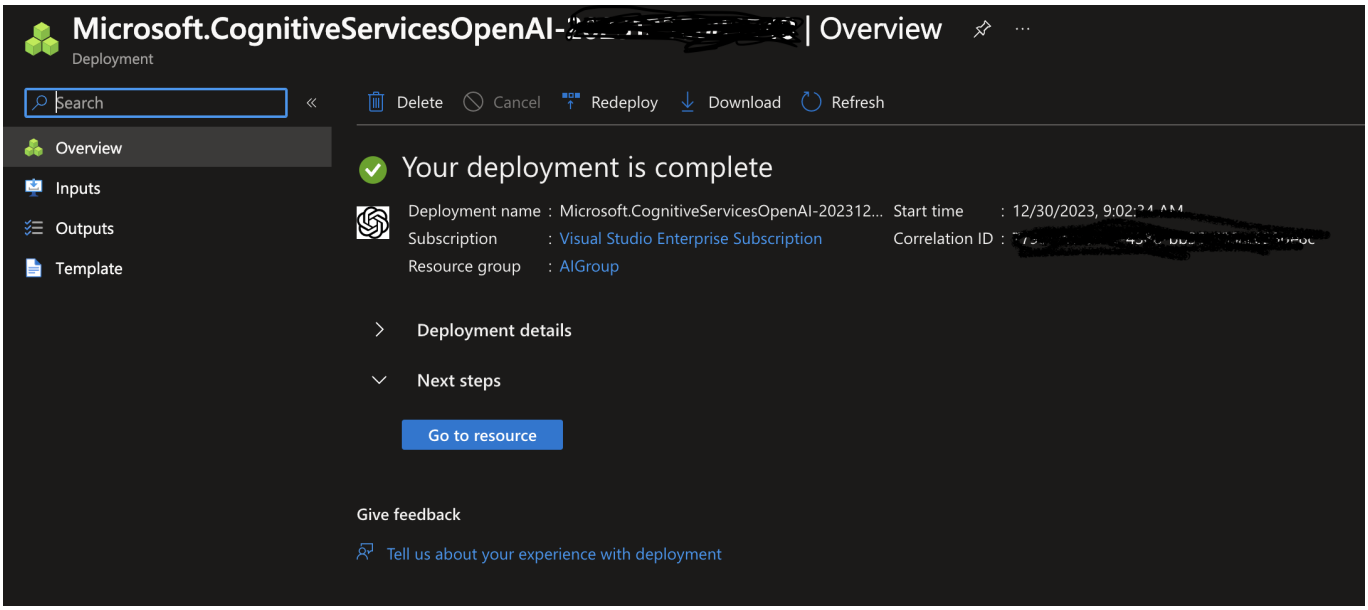
1. Select Azure OpenAI to create resources in Azure Portal



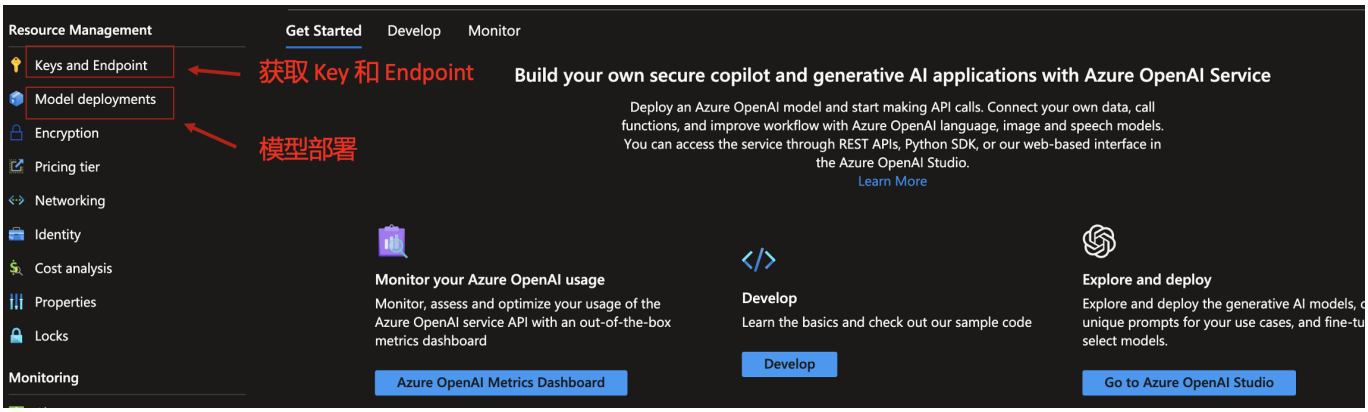
After click 'Create', configure the region where Azure OpenAI is located. Please note: Because the resource distribution is different, different regions have different OpenAI models. You must understand it clearly before using it.



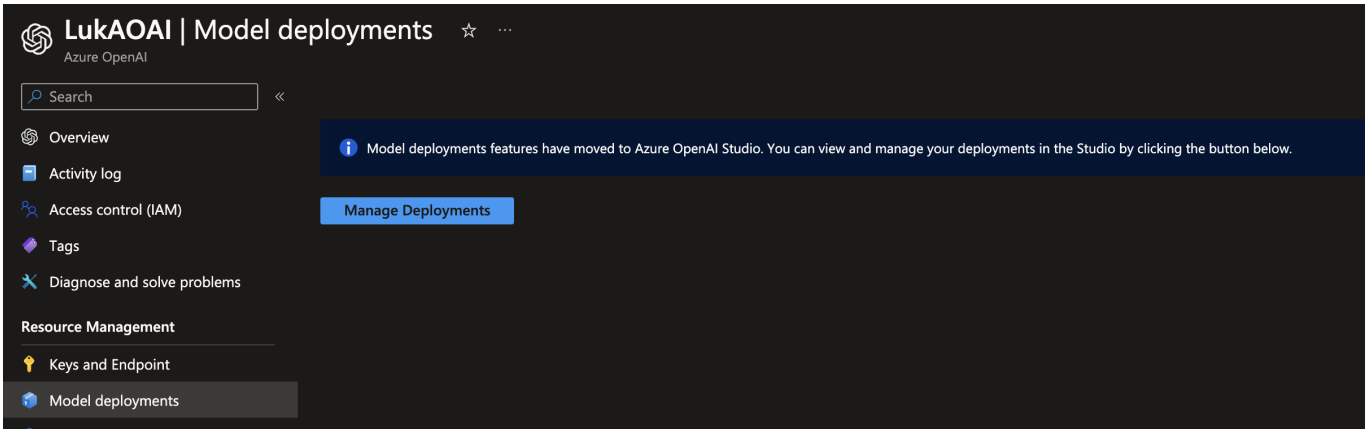
Wait for a moment



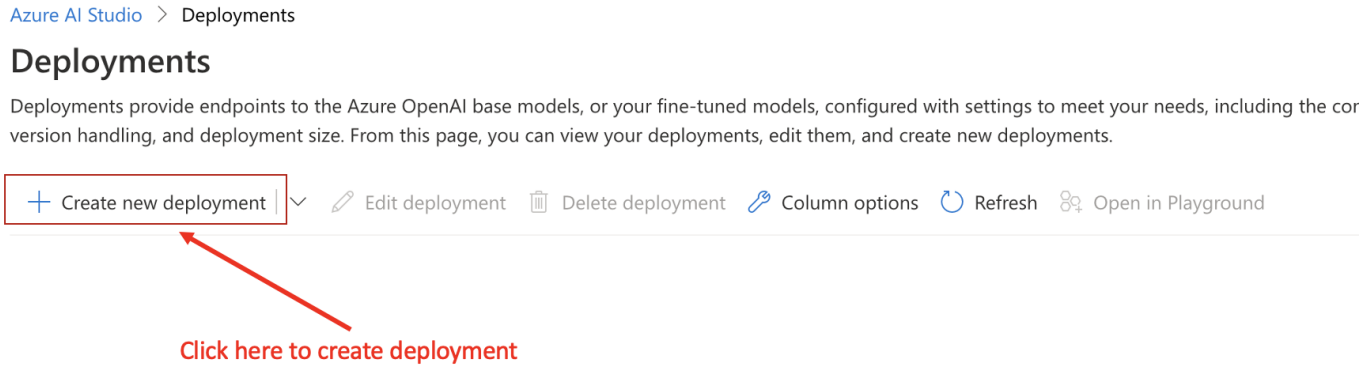
2. Go to the created resources, you can deploy the model, and obtain the Key and Endpoint required when calling the SDK



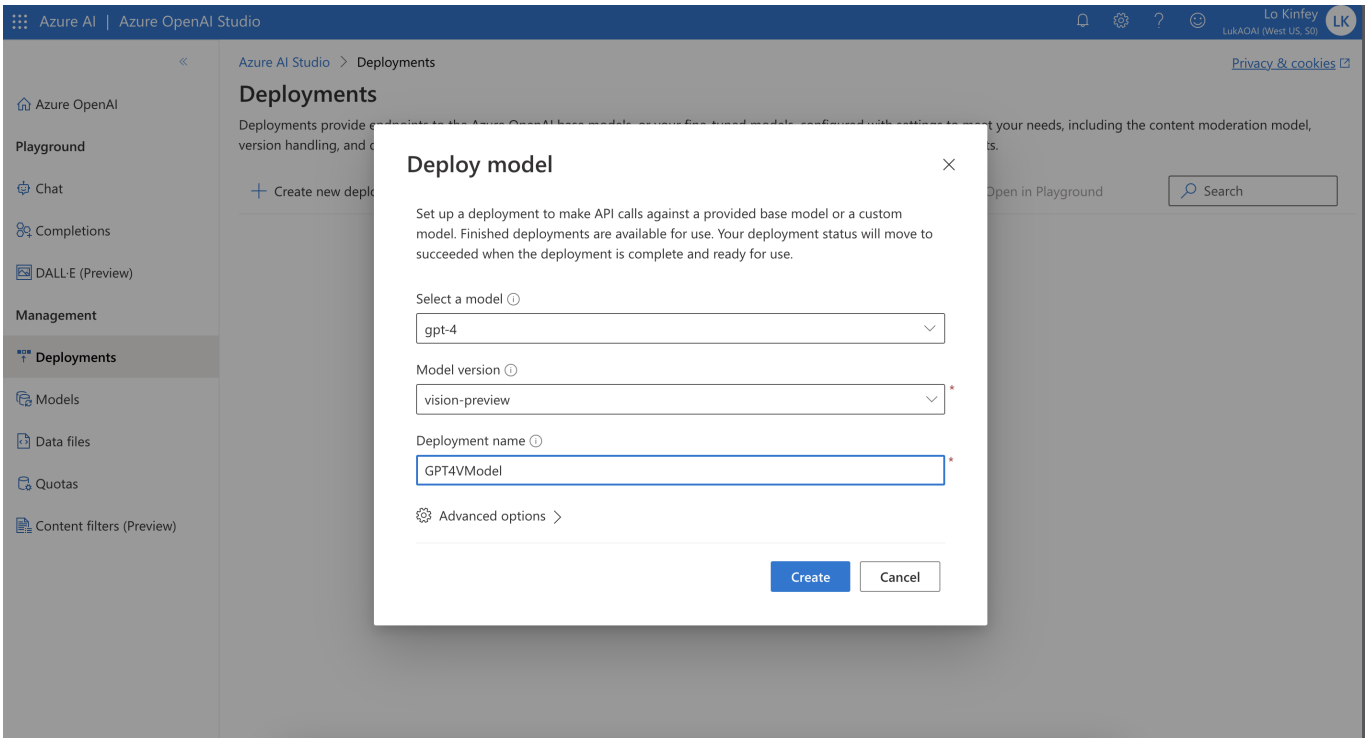
3. Enter 'Model Deployment' and select 'Management Deployment' to enter Azure OpenAI Studio



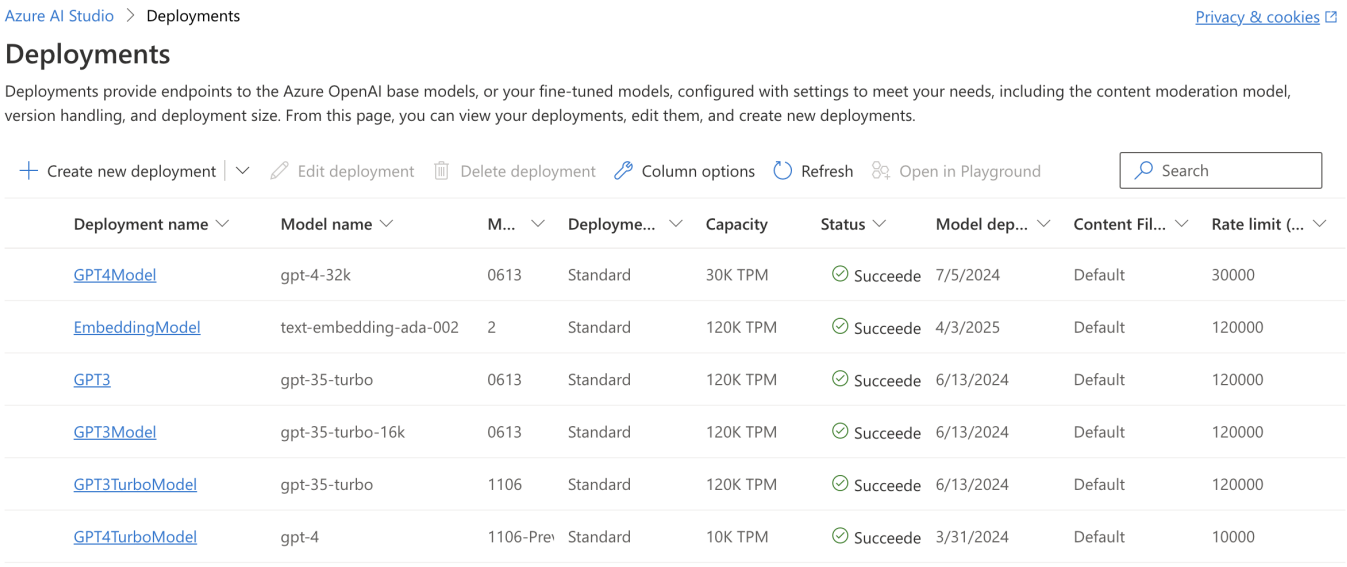
4. Deploy your model in Azure OpenAI Studio



Choose the model you need



this is your model list



Congratulations, you have successfully deployed the model. Now you can use the SDK to connect it.

Using SDK with Azure OpenAI Service

The SDK that interfaces with Azure OpenAI Service includes the SDK released by OpenAI for the Python version, and the SDK released by Microsoft for .NET. As a beginner, it is recommended to use it in a Notebook environment so that it is easier to understand the key steps of execution.

Python SDK

The official Python SDK released by OpenAI supports linking OpenAI and Azure OpenAI Service. Now OpenAI SDK has released version 1.x, but many people on the market are using version 0.2x. ***The content of this course will be based on OpenAI SDK version 1.x and use Python 3.10.x. ***

```
! pip install openai -U
```

.NET SDK

Microsoft releases an SDK based on Azure OpenAI Service. You can get the latest package through Nuget to complete .NET generative AI applications. ***The content of this course will be based on .NET 8 and the latest Azure.AI.OpenAI SDK to demonstrate examples. Of course, Polyglot Notebook will also be used as the environment***

```
#r "nuget: Azure.AI.OpenAI, *-*"
```

We have configured the SDK environment based on .NET / Python above. Next, we need to create the linked class to complete the related initialization work.

Getting started with the .NET environment

```
string endpoint = "Your Azure OpenAI Service Endpoint";
string key = "Your Azure OpenAI Service Key";

OpenAIClient client = new(new Uri(endpoint), new AzureKeyCredential(key));
```

Getting started with the Python environment

```
client = AzureOpenAI(
  azure_endpoint = 'Your Azure OpenAI Service Endpoint',
  api_key='Your Azure OpenAI Service Key',
  api_version="Your Azure OpenAI API version"
)
```

Using SDK to call Azure OpenAI Service API

1. Completion API

This is based on the gpt-35-turbo-instruct model, which is a very important API for text completion.

Completion API with .NET

```
CompletionsOptions completionsOptions = new()
{
    DeploymentName = "gpt-35-turbo-instruct",
    Prompts = { "Can you introduce what is generative AI ?" },
};

Response<Completions> completionsResponse =
client.GetCompletions(completionsOptions);

string completion = completionsResponse.Value.Choices[0].Text;
```

Completion API with Python

```
start_phrase = 'Can you introduce what is generative AI ?'

response = openai.Completion.create(engine=deployment_name,
prompt=start_phrase, max_tokens=1000)

text = response['choices'][0]['text'].replace('\n', '').replace(' .',
'.').strip()
```

2. Chat API

This is an API based on the gpt-35-turbo and gpt-4 models for the chat scenario

Chat with .NET

```
var chatCompletionsOptions = new ChatCompletionsOptions()
{
    DeploymentName = "gpt-4",
    Messages =
    {
        new ChatRequestSystemMessage("You are my coding assistant."),
        new ChatRequestUserMessage("Can you tell me how to write python
flask application?"),
    },
    MaxTokens = 10000
};

Response<ChatCompletions> response =
client.GetChatCompletions(chatCompletionsOptions);
```

Chat with Python

```
response = client.chat.completions.create(
    model="gpt-35-turbo", # model = "deployment_name".
    messages=[
        {"role": "system", "content": "You are my coding assistant."},
        {"role": "user", "content": "Can you tell me how to write python
flask application?"}
    ]
)

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

3. Generate images API

Scenario of Generate images based on DallE 3 model

Generate images with .NET

```
Response imageGenerations = await client.GetImageGenerationsAsync(
        new ImageGenerationOptions()
        {
            DeploymentName = "Your Azure OpenAI Service Dall-E 3 model
Deployment Name",
            Prompt = "Chinese New Year picture for the Year of the
Dragon",
            Size = ImageSize.Size1024x1024,
        });
```

Generate images with Python

```
result = client.images.generate(
    model="dalle3",
    prompt="Chinese New Year picture for the Year of the Dragon",
    n=1
)

json_response = json.loads(result.model_dump_json())
```

4. Embeddings API

Based on text-embedding-ada-002 model, implementation based on vector conversion

Embeddings with .NET

```
EmbeddingsOptions embeddingOptions = new()
{
    DeploymentName = "text-embedding-ada-002",
    Input = { "Kinfey is Microsoft Cloud Advocate" },
};

var returnValue = openAIClient.GetEmbeddings(embeddingOptions);

foreach (float item in returnValue.Value.Data[0].Embedding.ToArray())
{
    Console.WriteLine(item);
}
```

Embeddings with Python

```
client.embeddings.create(input = ['Kinfey is Microsoft Cloud Advocate'],
model='text-embedding-ada-002 model').data[0].embedding
```

Samples

Examples related to the above APIs are listed below. Please click here

Python examples Please visit Click here

.NET examples Please visit Click here

Summary

We use the most original and basic SDK to deal with Azure OpenAI Service. This is also our first step towards generative AU programming. We can understand different interfaces more quickly without using a framework, and it also lays the foundation for us to enter Semantic Kernel

Foundations of Semantic Kernel

We have already told you about LLMs and how to connect to Azure OpenAI Service using .NET and Python SDK. Next we will enter the world of Semantic Kernel. In 2023, we will have a lot of frameworks based on LLMs born. Why should we choose Semantic Kernel? What are the advantages and disadvantages of Semantic Kernel? And how do you use Semantic Kernel as a traditional developer? I will explain it in detail in this chapter.

What is Semantic Kernel

Semantic Kernel is a lightweight open source framework. Through Semantic Kernel, you can quickly use different programming languages (C#/Python/Java) combined with LLMs (OpenAI, Azure OpenAI, Hugging Face and other models) to build intelligent applications. After we entered generative AI, the way humans and machines communicate has changed a lot. We can use natural language to complete conversations with machines, and the threshold has been lowered a lot. Combining hint engineering and LLMs, we can complete different businesses at a lower cost. But how to introduce prompt engineering and LLMs into projects? We need a framework like Semantic Kernel as the basis for opening the door to intelligence. In May 2023, Microsoft CTO Kevin Scott proposed the concept of Copilot Stack, with artificial intelligence orchestration at its core. Semantic Kernel has the ability to be combined with LLMs and plug-ins composed of various prompt engineerings/software, so it is also regarded as the best practice of Copilot Stack. Through Semantic Kernel, you can easily build solutions based on Copilot Stack, and it can also be seamlessly connected to traditional projects.

Semantic Kernel vs LangChain

We have no choice but to make some objective comparisons. After all, LangChain has more user groups. It is undeniable that LangChain now has more features than Semantic Kernel in practical scenarios, especially in the entry-level reference examples. Let's make a more comprehensive comparison:

LangChain is an open source framework based on Python and Javascript and contains many prefabricated components. Developers can complete the development of intelligence applications without writing additional prompt engineerings. Especially in complex application scenarios, developers can quickly integrate multiple predefined components to complete the synthesis. From a development perspective, it is more suitable for developers with a foundation in data science or artificial intelligence.

Semantic Kernel You can use open source frameworks based on C#, Python, and Java. The greater advantage lies in engineering. After all, it is more like a programming paradigm. Traditional developers can quickly master the framework for application development, and can better combine customized plug-ins and prompt projects to complete the enterprise's customized business intelligence work.

The two have a lot in common, and both are still in version iteration. We need to make a choice based on the team structure, technology stack, and application scenarios.

Features of Semantic Kernel

1. **Powerful Plugins** - You can solve intelligentce business problems by combining custom/predefined plugins. Let traditional codes and intelligentce plugins work together to flexibly connect to application

scenarios, simplifying the process of transforming traditional applications into intelligent ones.

2. **Multi-model support** - Configure the "brain" for your intelligent application, which can be from Azure OpenAI Service, OpenAI, and various offline models on Hugging Face. Through the linker, you can quickly connect to different "brains" to make your application more smarter.

3. **Various connectors** - In addition to linking the "brain", the connectors can also link to vector databases, various business software, and different business middleware, allowing more business scenarios to enter the intelligent world possible

4. **Convenient Development** - Simple and easy to use, developers can get started at zero cost

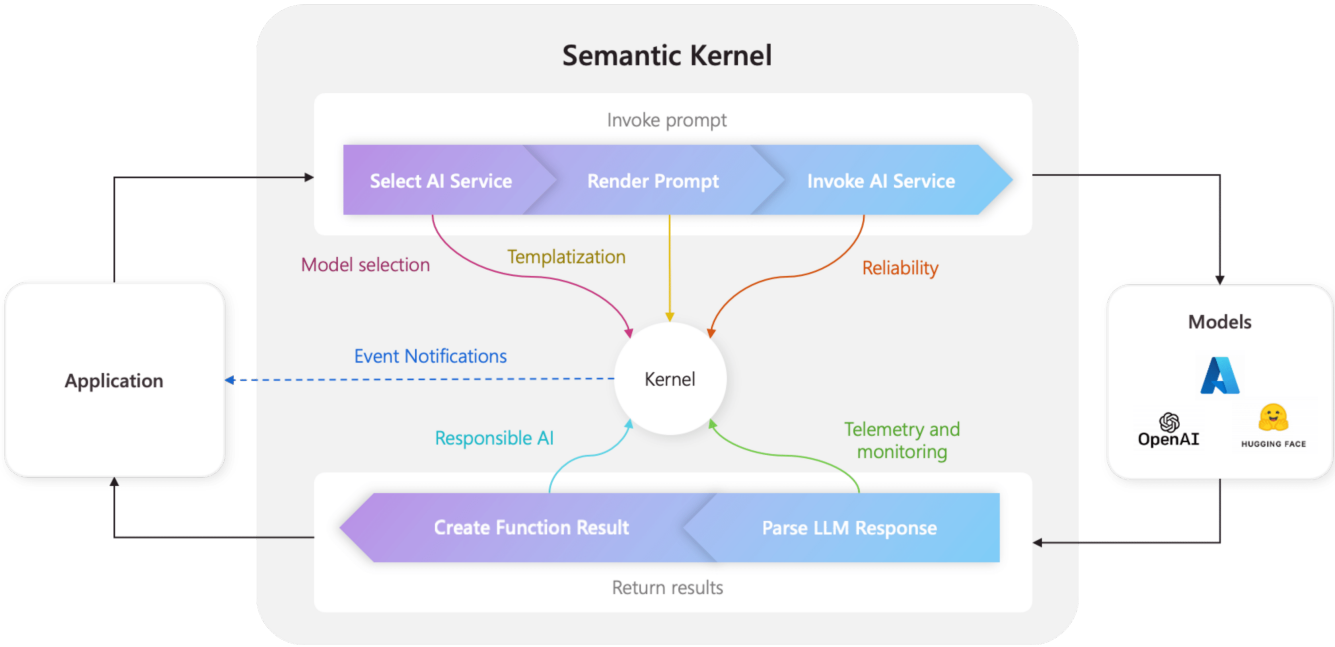
Disadvantages of Semantic Kernel

After all, LLMs are still developing, with many new models being added, many new functions, and new concepts being introduced. Open source frameworks such as Semantic Kernel and LangChain are working hard to adapt to this new Moore's Law, but there will be uncertain changes in version iterations. Therefore, when using it, developers need to pay more attention to the change log on the corresponding GitHub Repo.

In addition, Semantic Kernel needs to take into account multiple programming languages, so the progress is inconsistent, which will also lead to the choice of Semantic Kernel among people with different technology stacks.

Semantic Kernel's Kernel

If Semantic Kernel is regarded as Copilot Stack best practice, then Kernel is the center of AI orchestration and is also mentioned in the official documentation. Kernel can be linked with different plug-ins, services, logs and different models. All Semantic Kernel applications start with the Kernel.



Build a simple translation project with Semantic Kernel

After talking about some basic knowledge, we started to learn how to introduce Semantic Kernel into .NET and Python project projects. We use four steps to complete the implementation of a translation

Step 1: Import the Semantic Kernel library

.NET

Note: We are using the latest Semantic Kernel 1.0.1 version here, and using Polyglot Notebook to complete related learning

```
#r "nuget: Microsoft.SemanticKernel, *-*"
```

Python

Note: We are using the latest version of Semantic Kernel 0.4.3.dev0 here, and using Notebook to complete related learning

```
! pip install semantic-kernel -U
```

Step 2: Create Kernel object

Note here that we need to put the Endpoint, Key and Deployment Name of the model related to Azure OpenAI Service in a file to facilitate calling and setting. In .NET environment I set it in Settings.cs environment, in Python environment I set it in .env environment

.NET

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


Kernel kernel = Kernel.CreateBuilder()
            .AddAzureOpenAIChatCompletion("Your Azure OpenAI Service
Deployment Name" , "Your Azure OpenAI Service Endpoint", "Your Azure
OpenAI Service API Key")
            .Build();
```

Python

```
import semantic_kernel as sk
import semantic_kernel.connectors.ai.open_ai as skaoai
```

```
kernel = sk.Kernel()
deployment, api_key, endpoint = sk.azure_openai_settings_from_dot_env()
kernel.add_chat_service("azure_chat_competion_service",
skaoai.AzureChatCompletion(deployment,endpoint,api_key=api_key,api_version
= "2023-07-01-preview"))
```

Step 3: Import plugins

In Semantic Kernel, we have different plugins. Users can use predefined plugins or custom plugins. If you want to know more, you can pay attention to the next chapter, where we will explain the use of plugins in detail. In this example, we are using a custom plug-in, which is already in the plugins directory.

.NET

```
var plugin =
kernel.CreatePluginFromPromptDirectory(Path.Combine(pluginDirectory,
"TranslatePlugin"));
```

Python

```
pluginFunc =
kernel.import_semantic_skill_from_directory(base_plugin,"TranslatePlugin")
```

Step 4: Running

.NET

```
var transalteContent = await kernel.InvokeAsync( plugin["Basic"],new()
{["input"] = "你好，我是你的 AI 编排助手 - Semantic Kernel"});

transalteContent.GetValue();
```

Python

```
translateFunc = pluginFunc["Basic"]

result = translateFunc("你好，我是你的 AI 编排助手 - Semantic Kernel")
```

you can visit

.NET Samples Click here

Python Samples Click here

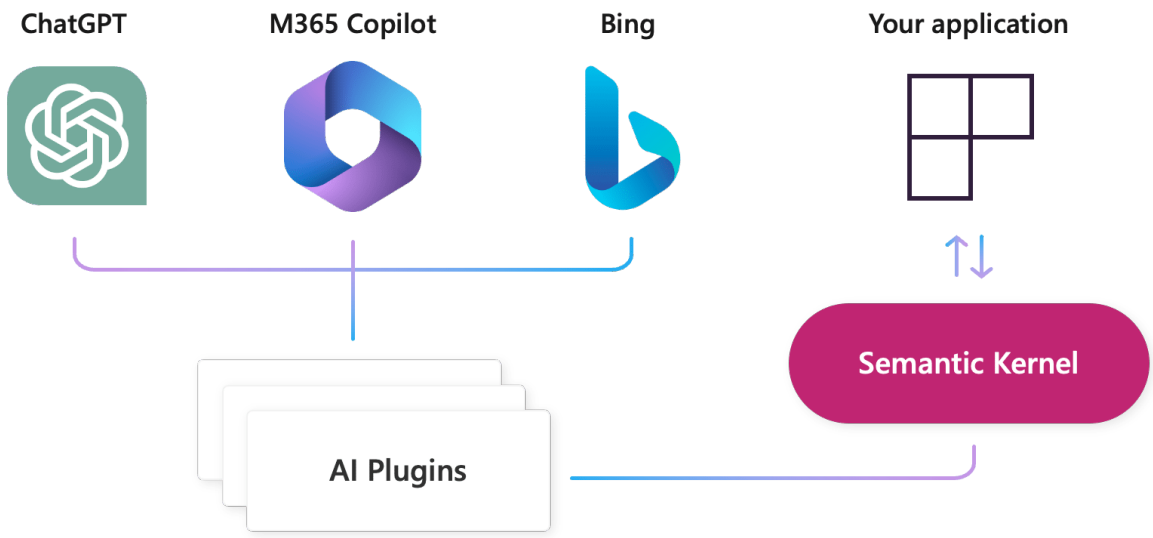
Summary

How did you feel when you first came into contact with Semantic Kernel? In this chapter, you learned the basics of Semantic Kernel and related knowledge. And understanding the comparison between Semantic Kernel and LangChain, as well as the related advantages and disadvantages, will be helpful to you in the process of project implementation. We have also completed a translation through Semantic Kernel in four steps, which is a hello world in the era of large models, giving you confidence when getting started. Next, you can open the next chapter for advanced learning to learn more about Semantic Kernel.

The skills of LLM - Plugins

We have learned the foundation of Semantic Kernel. One of the major features of Semantic Kernel is its powerful plugins, which solve intelligent business problems by combining custom/predefined plugins. Let traditional code and smart plugins work together to flexibly connect to application scenarios to simplify the process of transforming traditional applications into intelligent ones. In this chapter, we mainly introduce how to use plugins.

What's Plugins



We know that the original data of LLMs is time-limited, and if you want to add real-time content or enterprise knowledge, there are considerable shortcomings. OpenAI connects ChatGPT to third-party applications via plugins. These plugins enable ChatGPT to interact with developer-defined APIs, thereby enhancing ChatGPT's functionality and allowing a wider range of operations, such as:

1. Retrieve real-time information, such as sports scores, stock prices, latest news, etc.
2. Retrieve knowledge base information, such as company documents, personal notes, etc.
3. Assist users to perform related operations, such as booking flights, ordering meals, etc.

Semantic Kernel follows the plugin specifications of OpenAI plug-ins and can easily access and export plug-ins (such as plug-ins based on Bing, Microsoft 365, OpenAI), which allows developers to easily call different plugin services. In addition to plug-ins compatible with OpenAI, Semantic Kernel also has its own plugin definition method. Not only can Plugins be defined in a specified template format, but Plugins can also be defined within a function.

Plugins Style

In early preview versions of Semantic Kernel, Plugins were defined as Skills. As mentioned above, on par with OpenAI. But the specific goals have not changed. You can understand that the plug-in of Semantic Kernel is to provide developers with various functions to complete intelligent business requirements. You can think of Semantic Kernel as the base of Lego. To complete the construction of a Great Wall, you need various functional modules. And these functional modules are what we call plug-ins.

We may have a business-based plug-in set, such as HRPlugins, which contains different functions, such as:

Plugins	Intro
Holidays	holiday content
Contracts	About employee contracts
Departments	About organizational structure

These sub-functions can be effectively combined according to different requirements to complete different planned tasks. For example, the following structure:

```
|-plugins
    |-HRPlugins
        |-Holidays
        |-Contract
        |-Departments
    |-EmailsPlugins
        |-HRMail
        |-CustomMail
    |-PeoplesPlugins
        |-Managers
        |-Workers
```

We issue an instruction to the LLMs, "Please send an email to the manager whose contract has expired." In fact, we find a combination from different components, and finally complete the relevant work based on Contract + Managers + HRMail.

Semantic Kernel's plugins

Define plugins through templates

We know we can have conversations with LLMs through prompt engineering. For an enterprise or start-up company, when we deal with business, it may not be a prompt project, but may need a collection of prompt engineerings. We can put these prompt engineerings sets for business capabilities into the Semantic Kernel plugin collection. For plugins that combine prompt projects, Semantic Kernel has a fixed template. Prompt engineerings are placed in the skprompt.txt file, and related parameter settings are placed in the config.json file. The final file structure is like this

```
|-plugins
    |-HRPlugins
        |-Holidays
            |-skprompt.txt
            |-config.json
        |-Contract
```

```
            |-skprompt.txt
            |-config.json
        |-Departments
            |-skprompt.txt
            |-config.json
    |-EmailsPlugins
        |-HRMail
            |-skprompt.txt
            |-config.json
        |-CustomMail
            |-skprompt.txt
            |-config.json
    |-PeoplesPlugins
        |-Managers
            |-skprompt.txt
            |-config.json
        |-Workers
            |-skprompt.txt
            |-config.json
```

Let's first take a look at the definition of **skprompt.txt**. This is generally where business-related prompts are placed and can support multiple parameters. Each parameter is placed in {{$parameter's name}}, as in the following format:

```
Translate {{$input}} into {{$language}}
```

Our job here is to translate the input content into a specific language. Input and language are two parameters, which means you can give any value to these two parameters.

config.json contains configuration-related content. In addition to setting parameters related to LLMs, you can also set input parameters and related descriptions.

```
{
    "schema": 1,
    "type": "completion",
    "description": "Translate sentenses into a language of your choice",
    "completion": [
      {
        "max_tokens": 2000,
        "temperature": 0.7,
        "top_p": 0.0,
        "presence_penalty": 0.0,
        "frequency_penalty": 0.0,
        "stop_sequences": [
```

```
            "[done]"
          ]
        }
      ],
      "input": {
        "parameters": [
          {
            "name": "input",
            "description": "sentense to translate",
            "defaultValue": ""
          },
           {
            "name": "language",
            "description": "Language to translate to",
            "defaultValue": ""
          }
        ]
      }
  }
```

Define plugins through functions

We can also define different plugins through functions. This is a bit like Function Calling released by gpt-3.5-turbo on June 13, which enhances the capabilities of the OpenAI model by adding external functions and calling them. As mentioned at the beginning of this chapter.

Function Calling

By describing functions, LLMs call JSON objects of the parameters of these functions. This is a way to connect GPT functionality with external tools and APIs. Supports Azure OpenAI Service's or OpenAI's models:

- gpt-4
- gpt-4-1106-preview
- gpt-4-0613
- gpt-3.5-turbo
- gpt-3.5-turbo-1106
- gpt-3.5-turbo-0613

Semantic Kernel also supports Function Calling, but it is rarely used unless you have a Function Calling method customized for your business.

Semantic Kernel definition function plugin

Semantic Kernel defines function plugins, which is simpler than using Function Calling, and there are related definition methods before the birth of Function Calling. It is not limited by the model. You can use this method to perform knowledge and analysis on LLMs in early models. Data augmentation. It is recommended that all custom functions be placed in the same folder of plugins for easy management.

.NET

To define function extension, macro definition needs to be added

```
[KernelFunction,Description("search weather")]
public string WeatherSearch(string text)
{
    return "Guangzhou, 2 degree,rainy";
}
```

Note: It is recommended to use business class encapsulation to define different extension functions to make it easier to call, such as

```
using Microsoft.SemanticKernel;
using System.ComponentModel;
using System.Globalization;


public class CompanySearchPlugin
{
    [KernelFunction,Description("search employee infomation")]
    public string EmployeeSearch(string input)
    {
        return "talk about hr information";
    }

    [KernelFunction,Description("search weather")]
    public string WeatherSearch(string text)
    {
        return text + ", 2 degree,rainy";
    }
}
```

The calling method :

```
var companySearchPluginObj = new CompanySearchPlugin();

var companySearchPlugin =
kernel.ImportPluginFromObject(companySearchPluginObj,
"CompanySearchPlugin");

var weatherContent = await kernel.InvokeAsync(
companySearchPlugin["WeatherSearch"],new(){["text"] = "guangzhou"});
```

```
weatherContent.GetValue<string>()
```

Python

To define function extension, macro definition needs to be added

```
@sk_function_context_parameter(name="city", description="city string")
def ask_weather_function(self, context: SKContext) -> str:
    return "Guangzhou's weather is 30 celsius degree , and very hot."
```

Place all custom functions in **native_function.py** and define them through classes, such as

```
from semantic_kernel.skill_definition import sk_function,
sk_function_context_parameter
from semantic_kernel import SKContext


class API:
    @sk_function(
        description = "Get news from the web",
        name = "NewsPlugin"
    )
    @sk_function_context_parameter(name="location", description="location
name")
    def get_news_api(self, context: SKContext) -> str:
        return """Get news from the """ + context["location"] + """."""

    @sk_function(
            description="Search Weather in a city",
            name="WeatherFunction"
        )
    @sk_function_context_parameter(name="city", description="city string")
    def ask_weather_function(self, context: SKContext) -> str:
        return "Guangzhou's weather is 30 celsius degree , and very hot."

    @sk_function(
            description="Search Docs",
            name="DocsFunction"
        )
    @sk_function_context_parameter(name="docs", description="docs string")
    def ask_docs_function(self, context: SKContext) -> str:
```

```
        return "ask docs :" + context["docs"]
```

Calling functions：

```
api_plugin = kernel.import_native_skill_from_directory(base_plugin ,
"APIPlugin")

context_variables = sk.ContextVariables(variables={
    "location": "China"
})

news_result = await api_plugin["NewsPlugin"].invoke_async(
variables=context_variables)

print(news_result)
```

Predefined plugin

Semantic Kernel has a lot of predefined plug-ins as related capabilities for solving general business. Of course, as Semantic Kernel matures, more built-in plug-ins will be incorporated.

Samples

.NET Samples Click

Python Samples Click

Summary

Through plugins, you can complete more work based on business scenarios. Through studying this chapter, you have initially mastered the definition of plugins and how to use plugins to work. I hope you can create a plugin library for your enterprise based on business scenarios in real business work.

Planner - Let LLM have planning work

In Last Chapter, we learned a very important function of Semantic Kernel - plug-ins, through which work in different fields can be completed. LLMs change the way human-computer interaction uses natural language to talk to LLMs and let the LLMs complete the work. But often the instructions we give are not just about completing a single task, such as "Please send a dressing reminder email to people on business trips based on the weather in Seattle." We have always wanted artificial intelligence to compare with humans. If we think about the above instructions using human thinking, we will have the following split:

1. Check the weather in Seattle
2. Query the business travelers and their contact information from the company system
3. Reminder email template
4. Send email

LLMs actually have the same thinking. There is a powerful Planner function in the Semantic Kernel to split tasks. This chapter will tell you the relevant content.

What's Planner

Planner is an important component of the Semantic Kernel. It can receive task instructions and then correspond to the built-in plug-ins or custom plug-ins that have been defined in the Kernel, so that the task instructions can work step by step. As mentioned at the beginning, in fact, for the instruction "please send a dressing reminder email to business people based on the weather in Seattle", the relevant plugins will be defined in plugins and registered through Kernel, Semantic Kernel will Assist you with the pairing steps.

How to use Planner

Now Planner is still in the evolutionary stage. In the latest .NET version, we found that the Semantic Kernel version of the Planner component Microsoft.SemanticKernel.Planners.Handlebars is different from the core Microsoft.SemanticKernel. Some users are questioning whether the Semantic Kernel version number is confusing. You can understand that the Semantic Kernel team has completed the core part in 2023, and component-based functions such as Planner, Memory, and some Connectors are still evolving. After all, these features are relevant to the development of LLMs.

If you need to use Planner, you need to consider your business scenario. For example, adding Planner to some business processes and adding Planner to tool chains are very useful. After all, humans think a lot about work automation.

.NET

add related component libraries about Planner

```
#r "nuget: Microsoft.SemanticKernel.Planners.Handlebars, *-*"
```

import library

```
using Microsoft.SemanticKernel.Planning;
```

Note: When using HanlerBars, you need to note that these features are still evolving, and please ignore SKEXP0060 when using them.

```
#pragma warning disable SKEXP0060
```

Python

import library

```
from semantic_kernel.planning.basic_planner import BasicPlanner
```

or

```

```

```
from semantic_kernel.planning.sequential_planner import SequentialPlanner
```

Note: The settings of Python's Planner and .NET's Planner are different here. Python should be synchronized with .NET, so when using Python, you need to be prepared for future changes.

The Changing Planner

In the official blog, changes in Planner are mentioned https://devblogs.microsoft.com/semantic-kernel/migrating-from-the-sequential-and-stepwise-planners-to-the-new-handlebars-and -stepwise-planner/ combines Function Calling to rearrange the different Planner integrations in the preview version. You can pay attention to this content to learn more.

If you want to understand the principles of Planner implementation, please refer to

https://github.com/microsoft/semantic-kernel/blob/main/dotnet/src/Planners/Planners.Handlebars/Handlebars/CreatePlanPrompt.handlebars

Sample

.NET Sample Please click here

Python Sample Please click here

Summary

The addition of Planner greatly improves the usability of Semantic Kernel, especially for business and tool scenarios. Building an enterprise-level plugin library is also very important for the implementation of Planner. After all, we use plug-ins to combine different tasks to complete the work.

Embedding Skills

Many industries hope to have the capabilities of LLMs and hope that LLMs can solve their own internal problems. This includes employee-related content such as onboarding instructions, leave and reimbursement processes, and benefit inquiries. Enterprise business flow-related content includes relevant documents, regulations, execution processes, etc., as well as some customer-oriented inquiries. Although LLMs have strong knowledge capabilities, industry-based data and knowledge cannot be obtained. So how to inject this industry-based knowledge content? This is also an important step for LLMs to enter the corporate world. In this chapter, we will talk to you about how to inject industry data and knowledge to make LLMs more professional. This is the basis on which we create RAG applications.

Let's start with vectors in natural language

In the field of natural language, we know that the finest granularity is words, words constitute sentences, and sentences constitute paragraphs, chapters and final documents. Computers don't know words, so we need to convert words into mathematical representations. This representation is a vector, that is, Vector. Vector is a mathematical concept. It is a directional quantity with magnitude and direction. With vectors, we can effectively vectorize text, which is also the basis of the field of computer natural language. In the field of natural language processing, we have many vectorization methods, such as One-hot, TF-IDF, Word2Vec, Glove, ELMO, GPT, BERT, etc. These vectorization methods have their own advantages and disadvantages, but they are all based on word vectorization, that is, word vectors. Word vectors are the basis of natural language processing and the basis of all OpenAI models. Let's look at several common methods in word vectors respectively.

One-hot

One-hot encoding uses 0 and 1 encoding to represent words. For example, we have 4 words, namely: I, love, Beijing, and Tiananmen. Then we can use 4 vectors to represent these 4 words, which are:

```
I = [1, 0, 0, 0]
love = [0, 1, 0, 0]
Beijing = [0, 0, 1, 0]
Tiananmen = [0, 0, 0, 1]
```

In traditional natural language application scenarios, we regard each word as represented by a One-Hot vector as a unique discrete symbol. The number of words in our vocabulary is the dimension of the vector. For example, the above example contains a total of four words, so we can use a four-dimensional vector to represent it. In this vector, each word is unique, which means that each word is independent and has no relationship. Such vectors are called One-Hot vectors. The advantage of One-Hot vector is that it is simple, easy to understand, and each word is unique without any relationship. However, the disadvantage of One-Hot vector is also obvious, that is, the dimension of the vector will increase as the number of words increases. For example, if we have 1000 words, then our vector will be 1000 dimensions. Such a vector is very sparse, that is, most of its values are 0. Such vectors will cause a very large amount of computer

calculations and are not conducive to computer calculations. Therefore, the disadvantage of One-Hot encoding is that the vector dimension is large, the calculation amount is large, and the calculation efficiency is low.

TF-IDF

TF-IDF is a statistic that evaluates the importance of a word to a corpus. TF-IDF is the abbreviation of Term Frequency - Inverse Document Frequency, which is called Term Frequency-Inverse Document Frequency in Chinese. The main idea of TF-IDF is: if a word appears frequently in an article and rarely appears in other articles, then this word is the keyword of this article. Generally, we are used to splitting this concept into two parts: TF and IDF.

TF - Term Frequency

Term Frequency refers to the frequency with which a word appears in an article. The formula for calculating word frequency is as follows:

```
TF = The number of times a word appears in the article / the total number
of words in the article
```

One problem with TF is that if a word appears many times in an article, the TF value of the word will be very large. In this case, we will consider this word to be the keyword of this article. But in this case, we will find that many words are keywords in this article. In this case, we will not be able to distinguish which words are keywords in this article. So we need to make some adjustments to TF, and this adjustment is IDF.

IDF - Inverse document frequency

Inverse Document Frequency refers to the frequency of a certain word appearing in all articles. The formula for calculating inverse document frequency is as follows:

```
IDF = log(Total number of documents in the corpus / (number of documents
containing the word + 1))
```

IDF 的计算公式中，分母加 1 是为了避免分母为 0 的情况。IDF 的计算公式中，语料库的文档总数是固定的，所以我们只需要计算包含该词的文档数就可以了。如果一个词在很多文章中都出现，那么这个词的 IDF 值就会很小。如果一个词在很少的文章中出现，那么这个词的 IDF 值就会很大。这样的话，我们就可以通过 TF 和 IDF 的乘积来计算一个词的 TF-IDF 值。TF-IDF 的计算公式如下：

```
TF-IDF = TF * IDF
```

In the calculation formula of IDF, 1 is added to the denominator to avoid the situation where the denominator is 0. In the calculation formula of IDF, the total number of documents in the corpus is fixed, so we only need to calculate the number of documents containing the word. If a word appears in many articles, the IDF value of this word will be small. If a word appears in few articles, the IDF value of this word will be large. In this case, we can calculate the TF-IDF value of a word by multiplying TF and IDF. The calculation formula of TF-IDF is as follows:

Word2Vec

We also call Word2Vec Word Embeddings, which is called word embedding in Chinese. The main idea of Word2Vec is that the semantics of a word can be determined by its context. Word2Vec has two models, CBOW and Skip-Gram. CBOW is the abbreviation of Continuous Bag-of-Words, which is called the continuous bag-of-words model in Chinese. Skip-Gram is the abbreviation of Skip-Gram Model, which is called skip model in Chinese. The idea of the CBOW model is to predict a word through its context. The idea of the Skip-Gram model is to predict the context of a word from a word. The advantage of Word2Vec is that it can get the semantics of words and the relationship between words. Compared with One-Hot encoding and TF-IDF encoding, the advantage of Word2Vec encoding is that it can obtain the semantics of words and the relationship between words. The disadvantage of Word2Vec encoding is that it is computationally intensive and requires a large corpus.

We mentioned before that the dimension of One-Hot encoding is the number of words, while the dimension of Word2Vec encoding can be specified. Generally we will specify 100 dimensions or 300 dimensions. The higher the dimensionality of Word2Vec encoding, the richer the relationships between words, but the greater the computational complexity. The lower the dimensionality of Word2Vec encoding, the simpler the relationship between words, but the smaller the calculation amount. The dimensionality of Word2Vec encoding is generally 100 or 300 dimensions, which can meet most application scenarios.

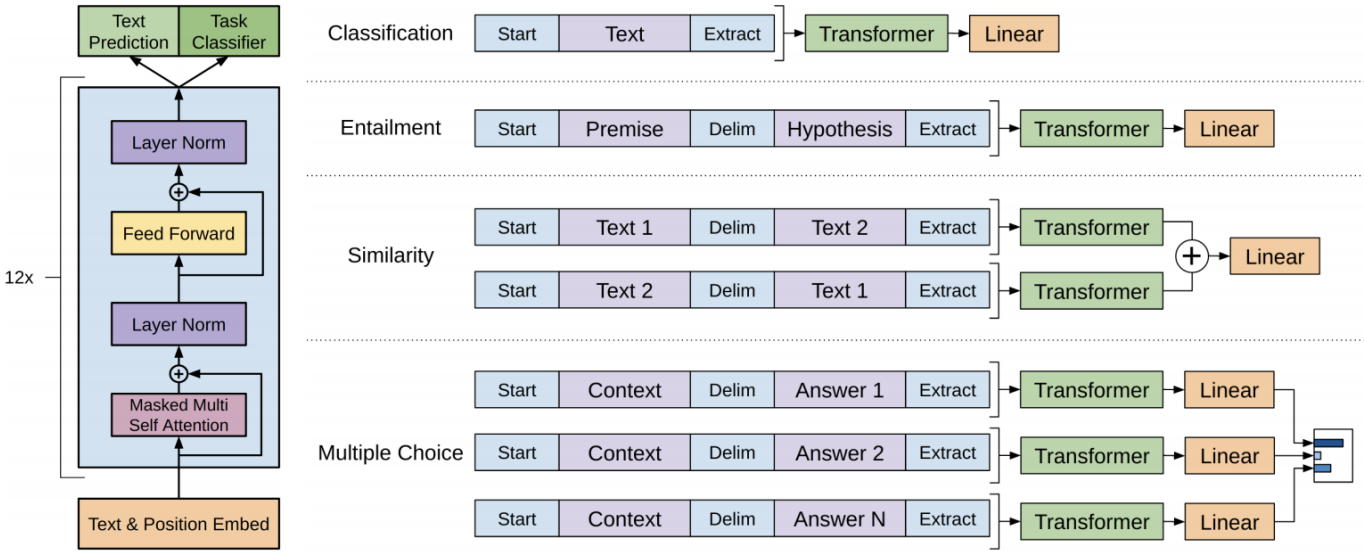
The calculation formula of Word2Vec encoding is very simple, which is Word Embeddings. Word Embeddings is a word vector whose dimensions can be specified. The dimensions of Word Embeddings are generally 100 or 300 dimensions, which can meet most application scenarios. The calculation formula for Word Embeddings is as follows:

```
Word Embeddings = Semantics of words + relationships between words
```

Think of Word2Vec as a simplified neural network.

GPT

The full name of the GPT model is Generative Pre-Training, which is called pre-training generative model in Chinese. The GPT model was proposed by OpenAI in 2018. Its main idea is that the semantics of a word can be determined through its context. The advantage of the GPT model is that it can obtain the semantics of words and the relationship between words. The disadvantage of the GPT model is that it is computationally intensive and requires a large corpus. The structure of the GPT model is a multi-layered unidirectional Transformer structure. Its structure is shown in the figure below:



The training process is divided into two stages, the first stage is pre-training and the second stage is fine-tuning. The pre-training corpora are Wikipedia and BookCorpus, and the fine-tuning corpora are different natural language tasks. The goal of pre-training is to predict a word through its context, and the goal of fine-tuning is to fine-tune the semantic model to obtain different models based on different natural language tasks, such as text classification, text generation, question and answer systems, etc.

The GPT model has gone through four stages, the most famous of which are GPT-3.5 and GPT 4 used by ChatGPT. GPT opens a new era, and its emergence allows us to see the infinite possibilities of natural language processing. The advantage of the GPT model is that it can obtain the semantics of words and the relationship between words. The disadvantage of the GPT model is that it is computationally intensive and requires a large corpus. Many people hope to have a GPT that benchmarks their own industry. This is also a problem we need to solve in this chapter.

BERT

BERT is the abbreviation of Bidirectional Encoder Representations from Transformers, which is called Transformer of Bidirectional Encoder in Chinese. BERT is a pre-trained model, and its training corpus is Wikipedia and BookCorpus. The main idea of BERT is that the semantics of a word can be determined by its context. The advantage of BERT is that it can obtain the semantics of words and the relationship between words. Compared with One-Hot encoding, TF-IDF encoding and Word2Vec encoding, the advantage of BERT encoding is that it can obtain the semantics of words and the relationship between words. The disadvantage of BERT encoding is that it is computationally intensive and requires a large corpus.

Vector embedding

We mentioned One-Hot encoding, TF-IDF encoding, Word2Vec encoding, BERT encoding, and GPT models. These codes and models are all a type of Embeddings embedding technology. Embeddings The main idea of embedding technology is that the semantics of a word can be determined by its context. The advantage of Embeddings technology is that it can obtain the semantics of words and the relationship between words. Embeddings are the basis of natural language deep learning. Its emergence allows us to see the infinite possibilities of natural language processing.

For the Embeddings method of text content, let's combine it with the previous section. You will find that since the birth of word2vec technology, the Embeddings of text content have been continuously

strengthened. From word2vec to GPT to BERT, the effect of Embeddings technology is getting better and better. The essence of Embeddings technology is "compression", using fewer dimensions to represent more information. The advantage of this is that it can save storage space and improve computing efficiency.

In Azure OpenAI Service, Embeddings technology is widely used to convert text strings into floating-point vectors and measure the similarity between texts by the distance between vectors. If different industries want to add their own data, we can use these enterprise-level data to query the vectors through the OpenAI Embeddings - text-embedding-ada-002 model, and save them through mapping. When using them, we can also convert the questions into vectors, and use similar Compare the algorithms to find the closest TopN results, so as to find the enterprise content related to the problem.

We can vectorize the enterprise data through the vector database and save it, and then use the text-embedding-ada-002 model to query through the similarity of the vectors to find the enterprise content associated with the problem. Commonly used vector databases include Qdrant, Milvus, Faiss, Annoy, NMSLIB, etc.

Open AI 的 Embeddings Model

Correlation of text strings with text embedding vectors from OpenAI. Embedding is usually used in the following scenarios

- Search (results sorted by relevance to query string)
- Clustering (where text strings are grouped by similarity)
- Recommend (recommend items with relevant text strings)
- Anomaly detection (identify outliers with little correlation)
- Diversity measurement (analyzing similarity distribution)
- Classification (where text strings are classified by their most similar tags)

Embeddings are vectors (lists) of floating point numbers. The distance between two vectors measures their correlation. Small distances indicate high correlation, and large distances indicate low correlation. For example, if you have a string "dog" with an embedding of [0.1,0.2,0.3], that string is more similar to a string "cat" with an embedding of [0.2,0.3,0.4] than to a string "cat" with an embedding of [0.9,0.8 ,0.7] the string "car" is more relevant.

Semantic Kernel's Embeddings

The support for Embeddings in Semantic Kernel is very good. In addition to supporting text-embedding-ada-002, it also supports vector databases. Semantic Kernel abstracts the vector database, and developers can use a consistent API to call the vector database. ***This case uses Qdrant as an example***. In order for you to run the example smoothly, please install Docker first, install the Qdrant container and run it. The running script is as follows:

```
docker pull qdrant/qdrant

docker run -p 6333:6333 qdrant/qdrant
```

.NET

Add Nuget library

```
#r "nuget: Microsoft.SemanticKernel.Connectors.Qdrant, *-*"
```

Reference library

```
using Microsoft.SemanticKernel.Memory;
using Microsoft.SemanticKernel.Connectors.Qdrant;
```

Create instances and Memory bindings

```
var textEmbedding = new AzureOpenAITextEmbeddingGenerationService("Your
Azure OpenAI Service Embedding Models Deployment Name" , "Your Azure
OpenAI Service Endpoint", "Your Azure OpenAI Service API Key");


var qdrantMemoryBuilder = new MemoryBuilder();
qdrantMemoryBuilder.WithTextEmbeddingGeneration(textEmbedding);
qdrantMemoryBuilder.WithQdrantMemoryStore("http://localhost:6333", 1536);

var qdrantBuilder = qdrantMemoryBuilder.Build();
```

Note: Semantic Kernel Memory component is still in the adjustment stage, so you need to pay attention to the risk of interface changes, and you also need to ignore the following information

```
#pragma warning disable SKEXP0003
#pragma warning disable SKEXP0011
#pragma warning disable SKEXP0026
```

Python

Reference library

```
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion,
AzureTextEmbedding
from semantic_kernel.connectors.memory.qdrant import QdrantMemoryStore
```

Add module support

```
kernel.add_text_embedding_generation_service(
        "embeddings_services", AzureTextEmbedding("EmbeddingModel",
endpoint,api_key=api_key,api_version = "2023-07-01-preview")
)
```

Add Memory

```
qdrant_store = QdrantMemoryStore(vector_size=1536,
url="http://localhost",port=6333)
await qdrant_store.create_collection_async('aboutMe')
kernel.register_memory_store(memory_store=qdrant_store)
```

Note: The Memory component is still in the adjustment stage, so you need to pay attention to the risk of interface changes.

Save and search your vectors in the Semantic Kernel

In Semantic Kernel, the methods of different vector data are unified through abstraction. You can easily save and search your vectors.

.NET

Save vector data

```
await qdrantBuilder.SaveInformationAsync(conceptCollectionName, id:
"info1", text: "Kinfey is Microsoft Cloud Advocate");
await qdrantBuilder.SaveInformationAsync(conceptCollectionName, id:
"info2", text: "Kinfey is ex-Microsoft MVP");
await qdrantBuilder.SaveInformationAsync(conceptCollectionName, id:
"info3", text: "Kinfey is AI Expert");
await qdrantBuilder.SaveInformationAsync(conceptCollectionName, id:
"info4", text: "OpenAI is a company that is developing artificial general
intelligence (AGI) with widely distributed economic benefits.");
```

Search vector data

```
string questionText = "Do you know kinfey ?";
var searchResults =  qdrantBuilder.SearchAsync(conceptCollectionName,
questionText, limit: 3, minRelevanceScore: 0.7);

await foreach (var item in searchResults)
{
    Console.WriteLine(item.Metadata.Text + " : " + item.Relevance);
}
```

Python

Save vector data

```
await kernel.memory.save_information_async(base_vectordb, id="info1",
text="Kinfey is Microsoft Cloud Advocate")
await kernel.memory.save_information_async(base_vectordb, id="info2",
text="Kinfey is ex-Microsoft MVP")
await kernel.memory.save_information_async(base_vectordb, id="info3",
text="Kinfei is AI Expert")
await kernel.memory.save_information_async(base_vectordb, id="info4",
text="OpenAI is a company that is developing artificial general
intelligence (AGI) with widely distributed economic benefits.")
```

Search vector data

```
ask = "who is kinfey?"

memories = await kernel.memory.search_async(
    base_vectordb, ask, limit=3, min_relevance_score=0.8
)

i = 0
for memory in memories:
    i = i + 1
    print(f"Top {i} Result: {memory.text} with score {memory.relevance}")
```

You can easily and conveniently access any vector database to complete related operations, which also means that you can build RAG applications very simply.

Sample

.NET Sample Please click here

Python Sample Please click here

Summary

Many enterprise data enter LLMs using Embeddings to build RAG applications. Semantic Kernel gives us a very simple way to complete related functions in both Python and .NET, so it is very helpful for those who want to add RAG applications to their projects.