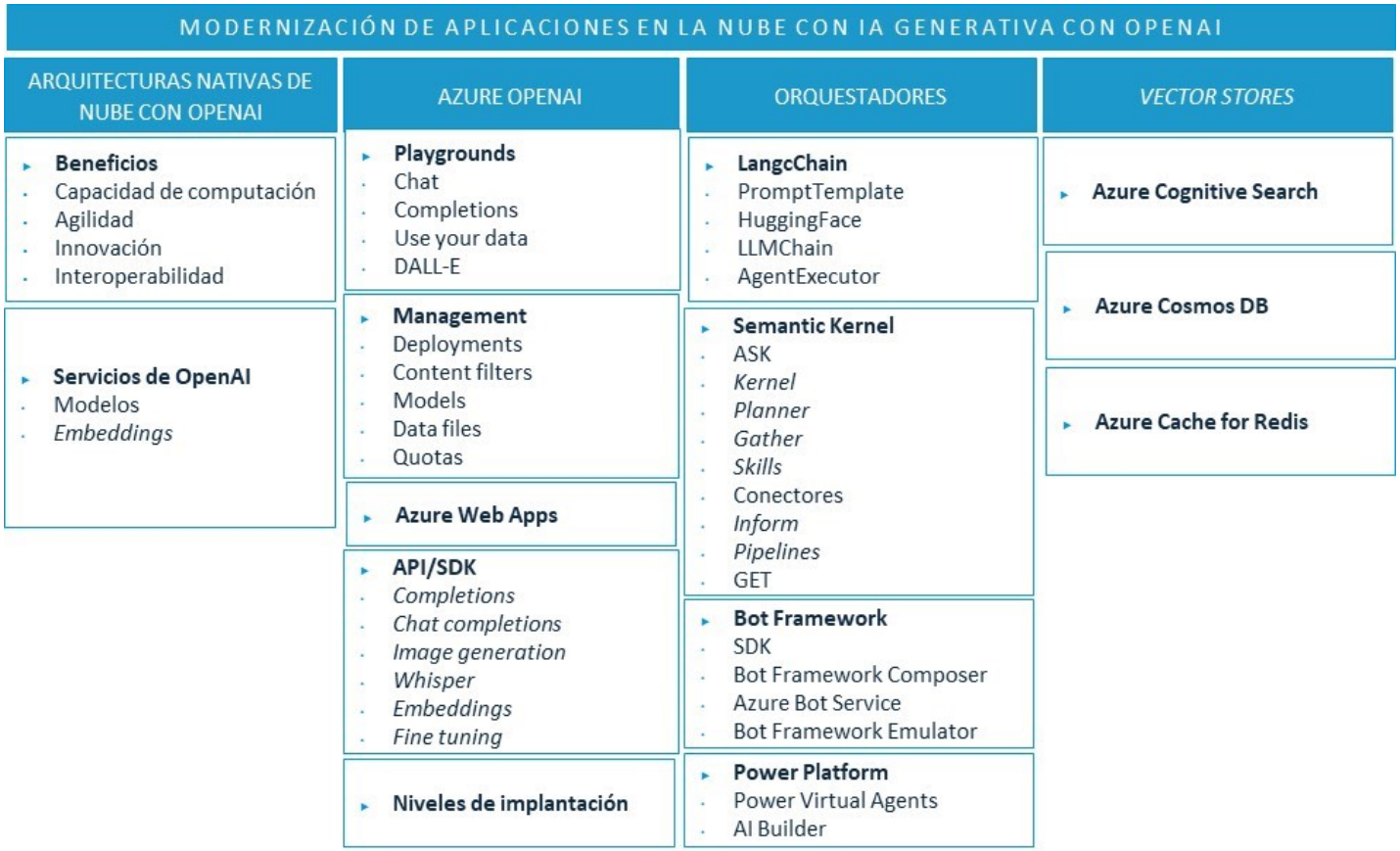
Herramientas para la Computación en la Nube Dirigida a Inteligencia Artificial

Tema 5. Modernización de aplicaciones en la nube con IA generativa con OpenAI

Índice

Esquema



5.1. Introducción y objetivos

En este tema, se describen los enfoques que pueden utilizarse para integrar capacidades de **IA generativa** en aplicaciones a través de **Azure OpenAI.**

OpenAI es la principal plataforma de IA generativa en la actualidad. Permite mantener conversaciones, responder a preguntas complejas, resumir textos, generar contenidos y fabricar imágenes a partir de una descripción.

Azure es la plataforma de computación en la nube de Microsoft. **Microsoft** ha establecido una colaboración con OpenAI para integrar sus soluciones en Azure. Como resultado de esta colaboración, Azure incluye una colección de servicios basados en OpenAI.

En primer lugar, en este tema se describe qué es una **arquitectura nativa** en la nube y cómo se beneficia de ella la IA generativa. A continuación, se describen las funcionalidades de la herramienta Azure OpenAI Studio, que permite construir soluciones de IA generativa a través de una interfaz gráfica y desplegarla en una aplicación web sin necesidad de programación. Sin embargo, también se incluyen ejemplos de uso de las APIs y SDKs que proveen Azure y OpenAI para construir estas soluciones de forma programática y repetible.

También se describen dos piezas clave en la arquitectura de IA generativa para usuarios avanzados: los **orquestadores de LLMs** (LangChain y Semantic Kernel) y los *vector stores,* que se utilizan para trabajar con *embeddings.*

Ideas clave

Por último, este tema se plantea los siguientes objetivos para el alumno:

- Familiarizarse con la plataforma líder del mercado en IA generativa a través de uno de los principales proveedores de computación en la nube.
- Ser capaces de implementar soluciones que incorporan IA generativa a través de la integración con servicios en la nube.
- Adquirir experiencia con orquestadores para combinar en una única solución múltiples elementos que contribuyen a generar respuestas precisas.

5.2. Diseño de arquitecturas nativas en la nube para IA generativa

Conceptos generales

Las **arquitecturas nativas de nube** son aquellas que han sido diseñadas para explotar las capacidades que proporciona un entorno de computación en la nube. Estas arquitecturas permiten implementar servicios de forma ágil a partir de la colección de herramientas que ofrecen los proveedores de nube.

Los servicios basados en IA generativa implementados a partir de una arquitectura nativa de nube se benefician de las siguientes **ventajas:**

- **Capacidad de cómputo:** para entrenar modelos de IA se requiere el procesamiento de grandes volúmenes de datos e infraestructura costosa. Los entornos de nube satisfacen ambos requisitos.
- **Agilidad:** los sistemas de IA generativa deben evolucionar para responder a cambios en los negocios. Los entornos de nube proporcionan herramientas que reducen el tiempo necesario para desplegar un evolutivo.
- **Innovación:** el ritmo de innovación en IA es elevado. Los entornos de nube ofrecen la versión más actualizada de los modelos de IA generativa como servicio para que el usuario disponga siempre de las últimas innovaciones de forma transparente.
- **Interoperabilidad:** los sistemas de IA generativa necesitan integrarse con otros sistemas para proporcionar valor en un proceso de negocio. Estas integraciones están disponibles de forma nativa en un entorno de nube.

Ideas clave

En Microsoft Azure, la arquitectura nativa de nube de referencia para los servicios de OpenAI se denomina **Azure OpenAI Landing Zone.** Esta arquitectura describe la forma en que se consumen y combinan diferentes servicios de Azure para implementar un caso de uso de IA generativa:

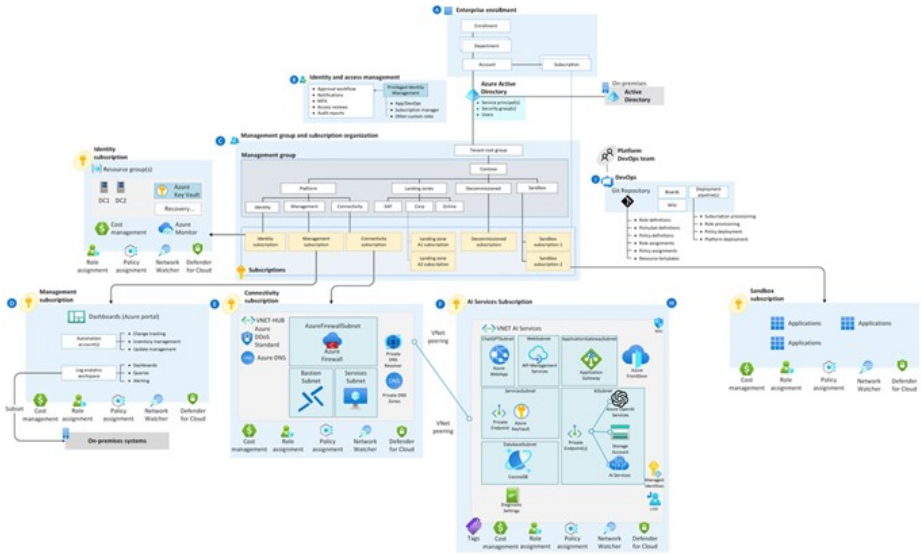


Figura 1. Arquitectura de referencia Azure OpenAI Landing Zone. Fuente: Ayala, 2023.

Los principales bloques son:

- **Gestión de APIs.** Proporciona un *gateway* que da acceso y securiza las APIs de OpenAI y el resto de los servicios.
- **Azure Web Apps.** Proporciona una plataforma de desarrollo y hospedaje de aplicaciones web.
- **AI Services.** Incluye los servicios de OpenAI y otros de Azure relacionados con inteligencia artificial.
- **Gestión de identidades.** Proporciona cuentas de usuario (con determinados roles y permisos) a las aplicaciones para que puedan acceder al resto de servicios de la arquitectura.

- **Azure Application Gateway.** Protege la aplicación web de intentos de intrusión a través de un Web Application Firewall.
- **Servicios DNS.** Permite que todos los servicios y componentes de la arquitectura puedan comunicarse entre sí a través de nombres de dominio.
- **Azure Key Vault.** Almacena contraseñas o certificados que necesitan los componentes para acceder a recursos y servicios.
- **Redes virtuales.** Proporciona las redes que interconectan los componentes y servicios incluidos en la arquitectura.
- ***Network security groups*** **(NSG).** Definiciones de seguridad para limitar las comunicaciones entre componentes al mínimo necesario.

Como se puede observar, en comparación con el tamaño de esta arquitectura de referencia, la componente de IA tiene un peso reducido. Por ello, es importante tener claro desde el inicio del proyecto que la complejidad asociada a la implementación de un nuevo servicio va mucho más allá de las funcionalidades basadas en IA generativa que deseemos poner en marcha.

Servicios de OpenAI

Los servicios de OpenAI que proporciona Microsoft Azure se basan en dos tipos de recursos: los **modelos preentrenados**, que pueden ser personalizados mediante *finetuning*, y los ***embeddings,*** que pueden utilizarse para generar nuevas bases de conocimiento a través de un *vector store.*

Modelos preentrenados

Categoría	Nombre	Descripción
Lenguaje	GPT-3.5 turbo	▸ Puede entender y generar lenguaje natural. ▸ Es utilizado por ChatGPT. ▸ Admite contextos de tamaño variable (entre 4k y 16k *tokens*).
	GPT-4, GPT-4 turbo	▸ Versión mejorada que proporciona un mayor rendimiento para abordar tareas más complejas ▸ Admite contextos de mayor tamaño que GPT-3.5 turbo.
	Whisper	▸ Permite convertir audio en texto. ▸ Puede consumirse a través del catálogo de modelos mediante el servicio AI Speech Service o a través de la herramienta Azure OpenAI Studio.
Programación	Codex	▸ Permite generar código fuente a partir de instrucciones en lenguaje natural
Imágenes	DALL-E	▸ Permite generar imágenes a partir de descripciones en lenguaje natural.

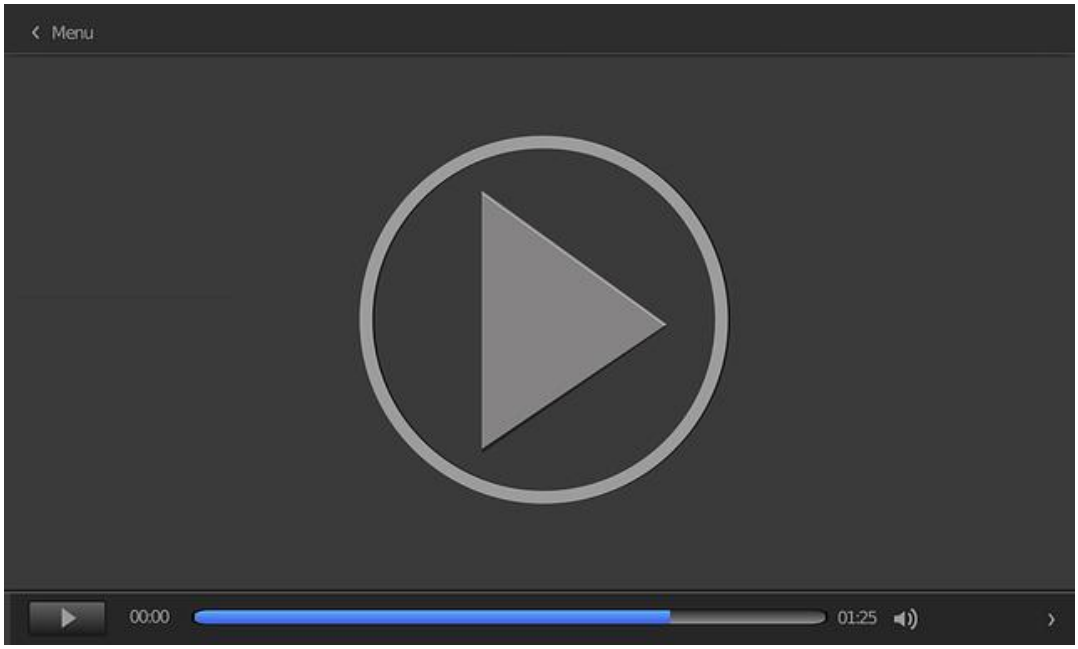
Tabla 1. Modelos OpenAI disponibles en Azure. Fuente: elaboración propia.

Finalmente, es importante tener en cuenta que para utilizar un modelo debe generarse una instancia de este. Esta instancia se denomina ***deployment.***

En los siguientes vídeos, se muestran las posibilidades de los modelos de OpenAI a través de ejemplos. En particular, se crean asistentes virtuales y se integran con herramientas externas para personalizar su comportamiento.

Ideas clave

Creación de asistentes virtuales con Assistant API de OpenAI

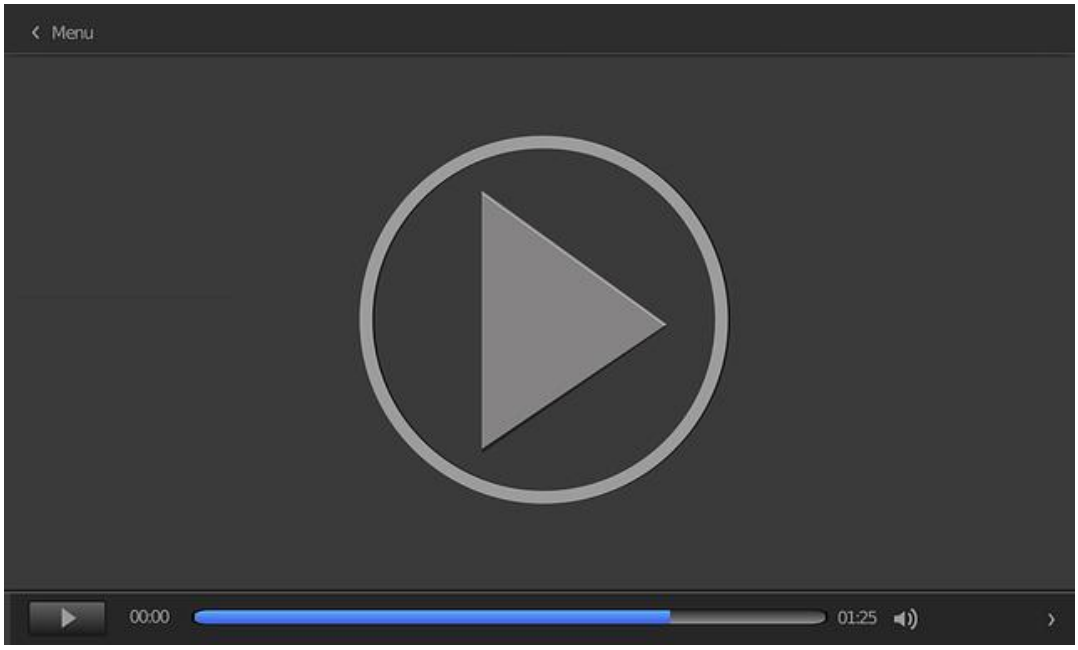


Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=f9e3eacb-2065-4fe2-b0b0-b16500da0d08

Ideas clave

Integración de asistentes virtuales con herramientas externas mediante Assistant API de OpenAI



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=7d894c8e-d962-4a04-b235-b16500ee6e96

Embeddings

Un *embedding* es una representación matemática de una palabra o texto. Permite capturar el significado de la palabra o el texto mediante un vector con valor numéricos. Las palabras o textos relacionados adquieren valores similares en sus respectivos vectores.

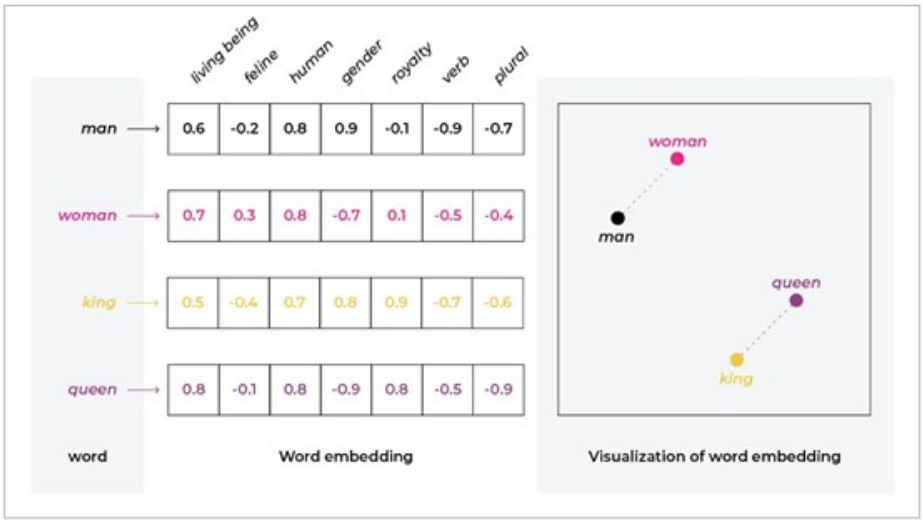


Figura 2. Ejemplo de *embeddings.* Fuente: Castillo, 2023.

El uso de *embeddings* permite resolver los siguientes escenarios:

- Detección de textos similares.
- Búsqueda de información acerca de un tema.
- Generación de código fuente a partir de texto en lenguaje natural.

El proceso para generar *embeddings* incluye los siguientes pasos:

- Recopilar textos sobre una temática y procesarlos para generar una base de conocimiento o *knowledge base.*

Ideas clave

- Convertir esta base de conocimiento en *embeddings* a través de la Embeddings API de Azure.
- Almacenar los *embeddings* generados en una base de datos de *embeddings,* denominada *vector database* en Microsoft Azure.

Desde ese momento, los usuarios pueden realizar preguntas sobre los contenidos de la base de conocimiento. Para ello, se utiliza también la Embeddings API. Cada pregunta se traduce a *embeddings* que se utilizan para consultar la *vector database.* Los resultados de dicha consulta son las respuestas más precisas a la pregunta formulada.

5.3. Implementación de capacidades de IA generativa con Azure OpenAI

Azure OpenAI Studio es el punto de entrada los servicios de OpenAI en Azure. A continuación, veremos las diferentes herramientas que proporciona OpenAI Studio para integrar IA generativa en aplicaciones y servicios.

Playgrounds

Esta herramienta proporciona entornos de experimentación para resolver casos de uso habituales a través de OpenAI. Permite la implementación de funcionalidades basadas en IA generativa sin necesidad de código fuente.

Chat Playground

Permite implementar asistentes basados en lenguaje natural o chatbots.

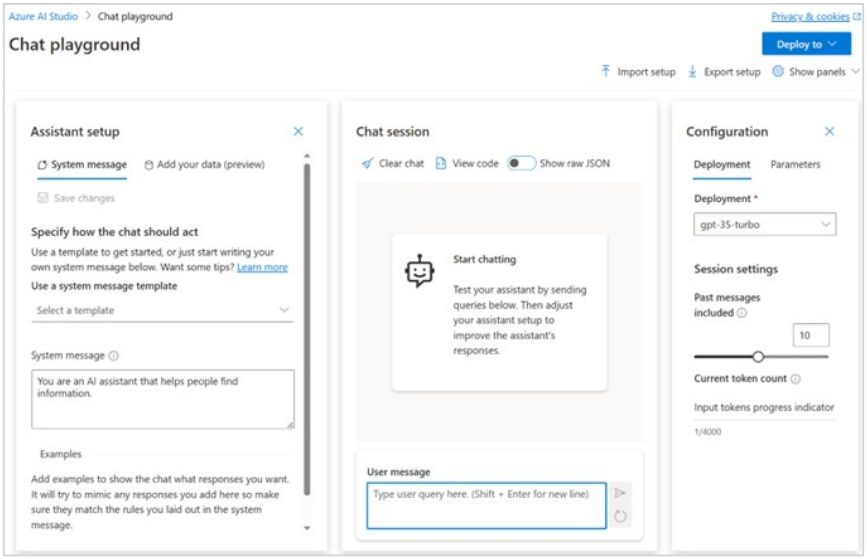


Figura 3. Pantalla principal de Chat Playground en Azure AI Studio. Fuente: Mrbullwinkle, KarlErickson, nitinme, mgreenegit, dbradish-microsoft, eric-urban, v-alju, 2023.

La pantalla principal se divide en tres secciones:

- **Assistant setup.** En esta sección se puede configurar el propósito del chatbot e indicar cómo debe comportarse («System message»), así como proporcionar ejemplos de preguntas/respuestas para que el chatbot amplíe su base de conocimiento o adquiera determinados patrones de comportamiento.
- **Chat session.** En esta sección se puede probar el comportamiento del chatbot enviando preguntas y observando las respuestas proporcionadas.
- **Configuration.** En esta sección se pueden configurar el resto de los parámetros relacionados con el chatbot:
  - **Deployment:** modelo de OpenAI en el que se basará el chabot.
  - **Temperatura:** un valor reducido de temperatura desencadena respuestas más precisas o ajustadas a los datos, mientras que un valor elevado incorpora respuestas creativas o inesperadas.
  - **Número máximo de tókenes** (palabras) incluidas en cada interacción, incluyendo los de la pregunta y los de la respuesta.
  - **Top probabilities:** similar al parámetro de temperatura, controla la aleatoriedad de las respuestas.
  - **Multiturn conversations:** número de mensajes anteriores (preguntas y respuestas) incluidos como contexto en cada nueva pregunta que se lanza al chatbot.
  - **Stop sequences:** permite especificar qué palabras provocan la finalización anticipada de la respuesta del chatbot.

Con el botón «Deploy to» es posible desplegar el chatbot en una aplicación. Para ello, se pueden utilizar los servicios Azure Web Apps (aplicación web) o Power Virtual Agents (aplicación multiplataforma).

Completions

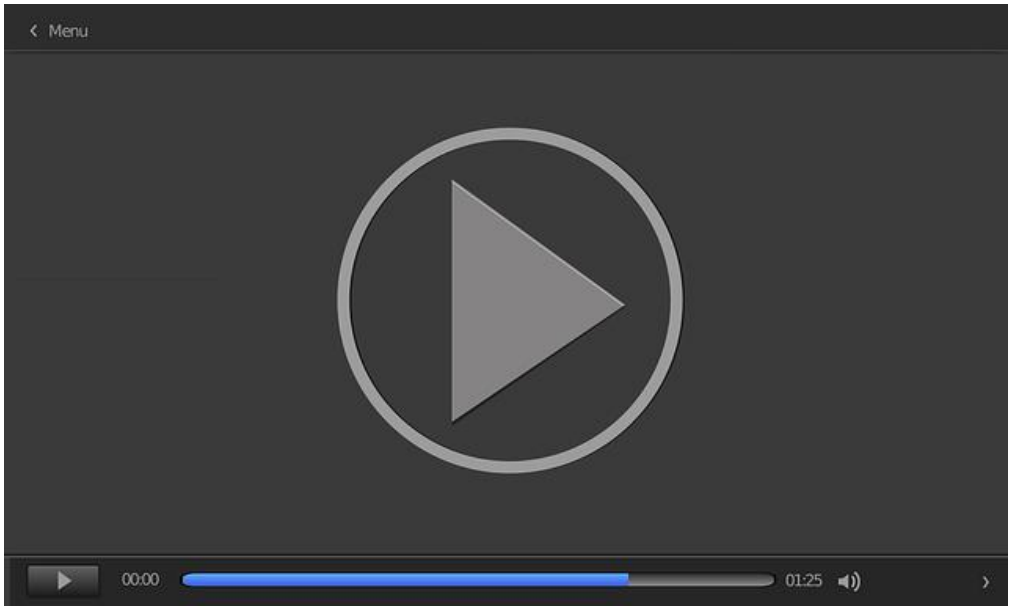
Se utiliza para realizar un amplio abanico de tareas relacionadas con procesamiento de textos a partir de un texto proporcionado como entrada. Se diferencia del Playground anterior en que no mantiene contexto, es decir, cada respuesta proporcionada es independiente de las interacciones previas.

Permite, por ejemplo, resumir el procedimiento para resolver una incidencia a partir de una conversación, resumir un informe financiero, resumir un artículo, dar ideas para el nombre de un producto, escribir un correo electrónico, describir los puntos clave de un producto, generar la descripción de una oferta de trabajo, escribir un examen de tipo test, clasificar un texto, detectar el propósito de un texto o analizar el tono de este.

En el siguiente vídeo, se muestran las capacidades de detección de contenido inadecuado, tanto en *prompts* como en las respuestas del chatbot, que proporciona la API de moderación de OpenAI.

Ideas clave

Moderación de contenidos en chatbots mediante Moderation API de OpenAI



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=a1947f4b-9da4-47ff-9aff-b16500f07bf5

Use your data

Es una variante del Playground de chatbot, que permite incorporar información adicional a la utilizada para el entrenamiento del modelo de OpenAI elegido. Para ello, se deben subir los ficheros (texto, PDF, etc.) a un Azure Blob Storage, el sistema de almacenamiento basado en objetos de Azure (similar al sistema de ficheros de un PC), y a continuación indexar su información mediante Azure Cognitive Search, de manera que esté disponible para el chatbot.

DALL-E

Es una interfaz de acceso a DALL-E, el modelo de OpenAI utilizado para **generar imágenes** a partir de una descripción en lenguaje natural proporcionada como entrada. Permite configurar el tamaño y número de imágenes generadas y mantener un álbum con todas las descripciones proporcionadas previamente y sus resultados.

Management

En este apartado se incluyen funcionalidades para crear o gestionar recursos asociados a los servicios de OpenAI. Estos recursos forman parte de la implementación de una aplicación o servicio con IA generativa a partir de modelos de OpenAI.

Las **secciones** incluidas en Management son las siguientes:

- **Deployments.** Enumera las instancias de modelos preentrenados de OpenAI, a los que podemos acceder mediante llamadas API. También permite crear una nueva instancia.
- **Content filters.** Permite incorporar o gestionar filtros de contenido. De esta forma, se evita que el chatbot acepte preguntas o proporcione respuestas relacionadas con determinadas temáticas (sexo, violencia).

- **Models.** Enumera los modelos de OpenAI disponibles en la región desde la que operamos en Azure.
- **Data files.** Permite preparar los datasets utilizados para realizar fine-tuning de los modelos preentrenados, es decir, para adaptarlos a bases de conocimiento específicas o comportamientos determinados.
- **Quotas.** Se utiliza para consultar los límites de consumo aplicables a las API que acceden a los modelos de OpenAI. También se pueden solicitar ampliaciones de las quotas actuales.

Azure Web Apps

Las funcionalidades creadas en Azure OpenAI Studio pueden disponibilizarse a través del servicio Azure Web Apps. Por ejemplo, es posible desplegar un chatbot haciendo clic en el botón «Deploy to» en el Chat Playground. Para ello, se deben ejecutar los siguientes **pasos:**

- Seleccionar la *web app,* en caso de que esté creada, o crear una nueva, especificando el nombre.
- Seleccionar el plan de precios de Azure App Service, que ofrece distintos perfiles de *hardware* para desplegar la *web app*.
- Activar o desactivar la opción «Chat history», que permite al usuario recuperar conversaciones previas con el chatbot. Estas conversaciones se almacenan en la base de datos de Azure CosmosDB.

Una vez pulsado el botón «Deploy», la aplicación web estará disponible a través de la siguiente URL: https://<appname>.azurewebsites.net

En la Figura 4, se muestra un ejemplo de chatbot desplegado a través de Azure Web Apps:

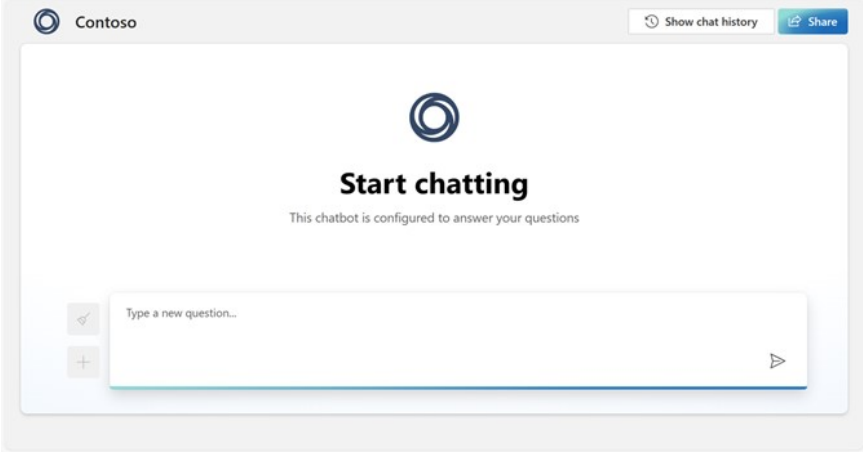


Figura 4. Ejemplo de chatbot desplegado en Azure mediante Azure Web Apps. Fuente: Aahill, mrbullwinkle, 2023.

Programación: APIs y SDKs

Las funcionalidades de Azure OpenAI también están disponibles a través de APIs. Las llamadas a estas APIs están empaquetadas en SDKs para los lenguajes más populares (C#, Go, Java, Spring, Javascript, Python, PowerShell).

A continuación, se incluyen ejemplos de uso de la API y del SDK para Python. Este SDK se instala mediante el siguiente comando:

```
pip install openai
```

En todos los ejemplos es preciso definir dos variables de entorno, una con el valor de la **API key** de OpenAI que utilizaremos para autenticación, y otra con la **URL del *endpoint*** de OpenAI al que dirigiremos las peticiones.

Pueden definirse en **Bash** con el siguiente comando:

```
export AZURE_OPENAI_API_KEY="[VALOR DE LA CLAVE]"

export AZURE_OPENAI_ENDPOINT="[URL DEL ENDPOINT]"
```

Completions – Python SDK

En el siguiente ejemplo, se pide a OpenAI que diseñe un eslogan para una heladería:

```
import os

from openai import AzureOpenAI



client = AzureOpenAI(

    api_key=os.getenv("AZURE_OPENAI_API_KEY"),

    api_version="2024-02-01",

    azure_endpoint = os.getenv("AZURE_OPENAI_ENDPOINT")

    )



deployment_name='REPLACE_WITH_YOUR_DEPLOYMENT_NAME'



print('Sending a test completion job')

start_phrase = 'Write a tagline for an ice cream shop. '

response = client.completions.create(model=deployment_name, prompt=start_phrase,
max_tokens=10)

print(start_phrase+response.choices[0].text)
```

En primer lugar, se inicializa el cliente que permite acceder a la API de OpenAI (client). A continuación, se especifica el nombre de nuestra instancia (deployment) y el texto de entrada (start_phrase). La respuesta se obtiene tras la llamada a la función create() .

La salida de este *script* es la siguiente:

```
Sending a test completion job

Write a tagline for an ice cream shop. The coldest ice cream in town!
```

Chat completions – Python SDK

En el siguiente ejemplo, se pide a OpenAI que configure un chatbot a partir de un *system message* y un ejemplo de pregunta/respuesta, y a continuación se lanza una pregunta para que el chatbot responda.

```
import os

from openai import AzureOpenAI



client = …



response = client.chat.completions.create(

    model="gpt-35-turbo",

    messages=[

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

        {"role": "user", "content": "Does Azure OpenAI support customer managed
keys?"},

        {"role": "assistant", "content": "Yes, customer managed keys are supported
by Azure OpenAI."},

        {"role": "user", "content": "Do other Azure AI services support this too?"}

    ]

)
```

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

La llamada a `create()` recibe como parámetros el *deployment* que queremos utilizar y una lista de mensajes, que incluye: el *system message* (rol system), un ejemplo (roles user/assistant) y la pregunta (rol user).

La salida de este *script* es la siguiente:


```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "Yes, most of the Azure AI services support customer managed
keys.
         However, not all services support it. You can check the documentation of
each
         service to confirm if customer managed keys are supported.",
        "role": "assistant"
      }
    }
  ],
  "created": 1679001781,
  "id": "chatcmpl-6upLpNYYOx2AhoOYxl9UgJvF4aPpR",
  "model": "gpt-3.5-turbo-0301",
  "object": "chat.completion",
```

```
  "usage": {
    "completion_tokens": 39,
    "prompt_tokens": 58,
    "total_tokens": 97
  }
}
Yes, most of the Azure AI services support customer managed keys.
However, not all services support it. You can check the documentation
 of each service to confirm if customer managed keys are supported.
```

La respuesta incluye el texto generado por OpenAI y otros datos adicionales, como el número de tókenes de la pregunta y la respuesta (prompt_tokens , completion_tokens , respectivamente), la funcionalidad utilizada (object), el modelo empleado (model) y un *timestamp* (created).

La lista de choices incluye, por defecto, un único mensaje de respuesta. Sin embargo, podemos especificar con el parámetro n el número de respuestas que queremos recibir. También es posible utilizar el parámetro temperature para regular el determinismo o creatividad de las respuestas.

Image generation – Python SDK

En el siguiente ejemplo se pide a OpenAI que genere una imagen de un oso caminando por el bosque:

```
from openai import AzureOpenAI
import os
import requests
from PIL import Image
import json
```

```
client = …


result = client.images.generate(
    model="dalle3",
    prompt="a close-up of a bear walking throughthe forest",
    n=1
)


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


image_dir = os.path.join(os.curdir, 'images')
if not os.path.isdir(image_dir):
    os.mkdir(image_dir)
image_path = os.path.join(image_dir, 'generated_image.png')


image_url = json_response["data"][0]["url"]
generated_image = requests.get(image_url).content
with open(image_path, "wb") as image_file:
    image_file.write(generated_image)
image = Image.open(image_path)
```

```
image.show()
```

En la función `generate()` se especifica el *deployment* que queremos utilizar, el *prompt* (descripción de la imagen) y el número de imágenes que se deben generar. La respuesta se recibe en formato JSON. A continuación, se crea la ruta local en la que almacenaremos la imagen, y se solicita esta imagen con `requests.get()`. Por último, se escribe el contenido de la imagen en la ruta local y se muestra a través del editor de imágenes por defecto.

Whisper – OpenAI API

En el siguiente ejemplo se pide a OpenAI que convierta a texto el contenido de un audio en formato `.wav` (`wikipediaOcelot.wav`):

```
curl
$AZURE_OPENAI_ENDPOINT/openai/deployments/YourDeploymentName/audio/transcriptions?
api-version=2024-02-01 \

-H "api-key: $AZURE_OPENAI_API_KEY" \

-H "Content-Type: multipart/form-data" \

-F file="@./wikipediaOcelot.wav"
```

Se utiliza `curl` para enviar la llamada a la API. Se debe especificar el nombre del *deployment* (`YourDeploymentName`), la API key y la ruta al fichero de audio.

El contenido de la respuesta es el siguiente:

```
{"text":"The ocelot, Lepardus paradalis, is a small wild cat native to the
southwestern United States, Mexico, and Central and South America. This medium-
sized cat is characterized by solid black spots and streaks on its coat, round
ears, and white neck and undersides. It weighs between 8 and 15.5 kilograms, 18 and
34 pounds, and reaches 40 to 50 centimeters 16 to 20 inches at the shoulders. It
was first described by Carl Linnaeus in 1758. Two subspecies are recognized, L. p.
paradalis and L. p. mitis. Typically active during twilight and at night, the
ocelot tends to be solitary and territorial. It is efficient at climbing, leaping,
and swimming. It preys on small terrestrial mammals such as armadillo, opossum, and
lagomorphs."}
```

Este texto se corresponde con la locución del fichero `.wav` enviado en la llamada.

Embeddings – Python SDK

En el siguiente ejemplo se pide a OpenAI que genere *embeddings* para el texto que se facilita como parámetro.

```
import os

from openai import AzureOpenAI


client = …


response = client.embeddings.create(
    input = "Your text string goes here",
    model= "text-embedding-ada-002"
)
print(response.model_dump_json(indent=2))
```

La **salida del *script*** es la siguiente:

```
{
  "object": "list",
  "data": [
    {
      "object": "embedding",
      "index": 0,
      "embedding": [ -0.006929283495992422, -0.005336422007530928, ...,
```

```
-4.547132266452536e-05, -0.024047505110502243 ],
    }
  ],
  "model": "text-embedding-ada-002",
  "usage": { "prompt_tokens": 5, "total_tokens": 5 }
}
```

En la sección `data` del objeto JSON recibido aparece el *embedding* correspondiente a la frase `Your text string goes here` .

Fine tuning – Python SDK

El proceso de *fine-tuning* se utiliza para incorporar bases de conocimiento nuevas sobre temáticas especializadas. Por ejemplo, es posible reentrenar un modelo de OpenAI para que comprenda los procedimientos internos de una organización y sea capaz de resolver cuestiones acerca de ellos. Este proceso se compone de los **pasos** que se indican a continuación. En cada paso se incluye el **código Python** relevante:

- **1. Preparar los datos de entrenamiento y validación.** Se deben generar en formato JSON. A continuación, se incluye un *dataset* abreviado. Para cada ejemplo se especifica el *system message,* la pregunta (`prompt`) y la respuesta (`completion`). Para ello se utilizan los roles `system` , `user` y `assistant` , respectivamente.

```
{"messages": [

{"role": "system", "content": "You are an Xbox customer support agent whose primary
goal is to help users
 with issues they are experiencing with their Xbox devices. You are friendly and
concise. You only provide
factual answers to queries, and do not provide answers that are not related to
Xbox."},

{"role": "user", "content": "Is Xbox better than PlayStation?"},

{"role": "assistant", "content": "I apologize, but I cannot provide personal
```

```
opinions.
My primary job is to assist you with any issues related to your Xbox device.
Do you have any Xbox-related issues that need addressing?"}

]}

{"messages": [{…}]}

{"messages": [{…}]}
```

- **2. Subir los ficheros de datos de entrenamiento y validación a Azure.** Para ello, se utiliza `client.files.create()` , indicando que se trata de ficheros utilizados para `fine-tuning` . Una vez subido al entorno de Azure, cada fichero recibe un **identificador** que puede utilizarse en las llamadas siguientes:

```
import os

from openai import AzureOpenAI



client = AzureOpenAI(…)

training_file_name = 'training_set.jsonl'

training_response = client.files.create(file=open(training_file_name, "rb"),
purpose="fine-tune")

training_file_id = training_response.id



validation_file_name = 'validation_set.jsonl'

validation_response = client.files.create(file=open(validation_file_name, "rb"),
purpose="fine-tune")

validation_file_id = validation_response.id



print("Training file ID:", training_file_id)
```

```
print("Validation file ID:", validation_file_id)
```

- **3. Ejecutar el** fine-tuning **a través de un** job. Para ello se utiliza client.fine_tuning.jobs.create() , indicando los identificadores de los ficheros de entrenamiento y validación. Con el identificador del job (response_id), es posible comprobar su estado para determinar cuándo ha finalizado:

```
response = client.fine_tuning.jobs.create(
    training_file=training_file_id,
    validation_file=validation_file_id,
    model="gpt-35-turbo-0613"
)

job_id = response.id
print("Job ID:", response.id)
print("Status:", response.id)
print(response.model_dump_json(indent=2))
```

- **4. Una vez completado el** job **de** fine-tuning **, el último paso es desplegar el modelo.** Para ello, se necesita la suscripción de Azure, el grupo de recursos, el recurso de tipo OpenAI y el nombre del *deployment* que vamos a crear. Esta información, junto con el nombre del modelo (<fine_tuned_model>) que aparece en response.model_dump_json() del paso anterior, se utiliza para generar una petición API que solicita la creación del *deployment*.

```
import json
import os
import requests
```

```
token= os.getenv("<TOKEN>")

subscription = "<YOUR_SUBSCRIPTION_ID>"

resource_group = "<YOUR_RESOURCE_GROUP_NAME>"

resource_name = "<YOUR_AZURE_OPENAI_RESOURCE_NAME>"

model_deployment_name ="gpt-35-turbo-ft"



deploy_params = {'api-version': "2023-05-01"}

deploy_headers = {'Authorization': 'Bearer {}'.format(token), 'ContentType':
'application/json'}

deploy_data = {

    "sku": {"name": "standard", "capacity": 1},

    "properties": {

        "model": {

            "format": "OpenAI",

            "name": <"fine_tuned_model">,

            "version": "1"

        }

    }

}

deploy_data = json.dumps(deploy_data)



request_url =
```

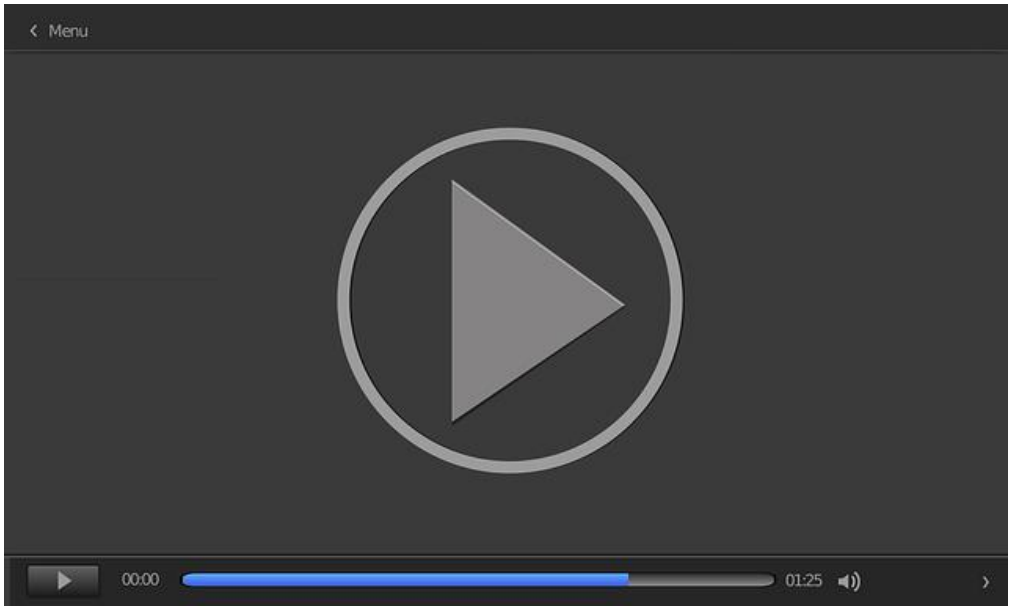
```
f'https://management.azure.com/subscriptions/{subscription}/resourceGroups/{resource_group}
/providers/Microsoft.CognitiveServices/accounts/{resource_name}/deployments/{model_deployment_name}'

r = requests.put(request_url, params=deploy_params, headers=deploy_headers,
data=deploy_data)

print(r.json())
```

- **5.** Una vez creado el *deployment,* puede utilizarse como hemos visto en ejemplos anteriores para **enviar *prompts* y recibir respuestas.**

Por último, en el siguiente vídeo, *Creación de modelos de clasificación personalizados mediante fine-tuning en OpenAI,* se muestra, a través de un Jupyter *notebook* en Python, cómo se puede llevar a cabo un proceso de *fine-tuning* en OpenAI para incorporar una nueva habilidad al chatbot a partir de un *dataset* propio.



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=05509b22-388d-4a66-8445-b16500f8cb97

Niveles de implantación

Una vez descritas las diferentes técnicas de implantación de capacidades de IA generativa a través de OpenAI, el siguiente paso es determinar el nivel de implantación que se requiere en nuestro caso de uso.

E l **nivel de implantación** se refiere a la profundidad con la que se utilizan las funcionalidades proporcionadas por Azure OpenAI. Naturalmente, el objetivo es utilizar la aproximación más sencilla que cumpla con todos los requisitos de nuestro caso de uso. Ese **punto de equilibrio** es precisamente el nivel de implantación óptimo.

En la Figura 5 se muestran los **elementos** que pueden incluirse en los diferentes niveles de implantación:

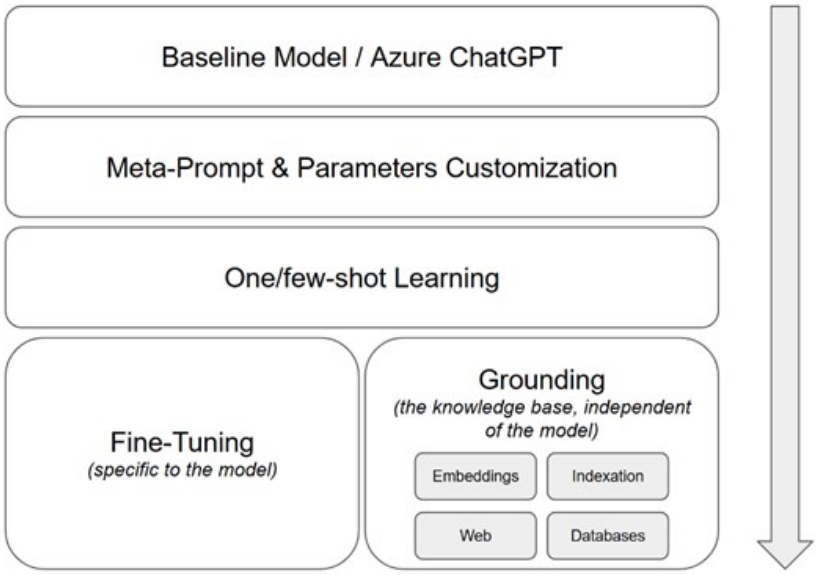


Figura 5. Elementos disponibles para configurar el nivel de implantación de Azure OpenAI. Fuente: Sánchez, 2024.

Nivel 1: implantación básica

En este nivel de implantación se utiliza una instancia propia de un modelo por defecto (preentrenado) de OpenAI. El objetivo es disponer de un **entorno de ChatGPT privado,** en el que los datos enviados por los usuarios están protegidos.

Para alcanzar este nivel de implantación, solo es necesario crear una **nueva instancia en Azure OpenAI Studio.** Desde el Chat Playground es posible especificar el *system message* y algunos parámetros básicos (longitud de la respuesta, temperatura), así como probar el funcionamiento del chatbot. Una vez que se obtienen los resultados esperados, solo hay que hacer clic en el botón «Deploy to» para desplegarlo a través de una Azure Web App, como se ha descrito en secciones previas.

Nivel 2: implantación básica con ejemplos

Este nivel de implantación es similar al anterior, pero en este caso se añaden **ejemplos de preguntas** *(prompts)* y respuestas para introducir un primer grado de personalización en el comportamiento y conocimiento del chatbot.

Estos ejemplos pueden proporcionarse a través del propio Playground en Azure OpenAI Studio o bien vía llamadas API, como las que se han mostrado en la sección «Programación: APIs y SDKs».

Nivel 3: implantación avanzada con *fine-tuning*

Este nivel de implantación **sustituye** los ejemplos del nivel anterior por un **proceso de reentrenamiento más completo** *(fine-tuning),* cuyo objetivo es que el chatbot adquiera una base de conocimiento nueva y especializada. Para ello, se proporcionan dos ficheros de datos en formato JSON, utilizados para entrenamiento y validación. La subida de los ficheros a Azure OpenAI y el reentrenamiento posterior pueden realizarse a través de la API/SDK, como se ha mostrado en el ejemplo de la sección anterior o bien a través de Azure Open AI Studio, haciendo clic en «Create custom model».

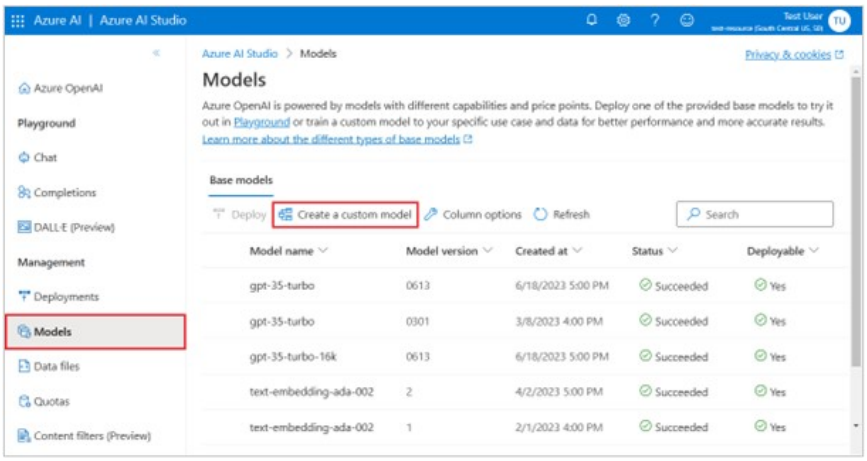


Figura 6. Creación de modelo personalizado en Azure AI Studio. Fuente: Mrbullwinkle, dbradish-microsoft, Pookam90, erc-urban, varun-dhawan, mgreenegit, jimmystridh, HeidiSteen, nitinme, MSFTeegardem, learn-build-servce-prod[bot], gahl-levy, 2024.

Nivel 4: Implantación avanzada con fuentes adicionales

En este nivel de implantación se **incorporan sistemas adicionales** para complementar el conocimiento de los modelos de OpenAI tanto aquellos proporcionados por defecto como los resultantes de un proceso de *fine-tuning.*

Estas fuentes adicionales pueden adquirir alguno de los siguientes **formatos:**

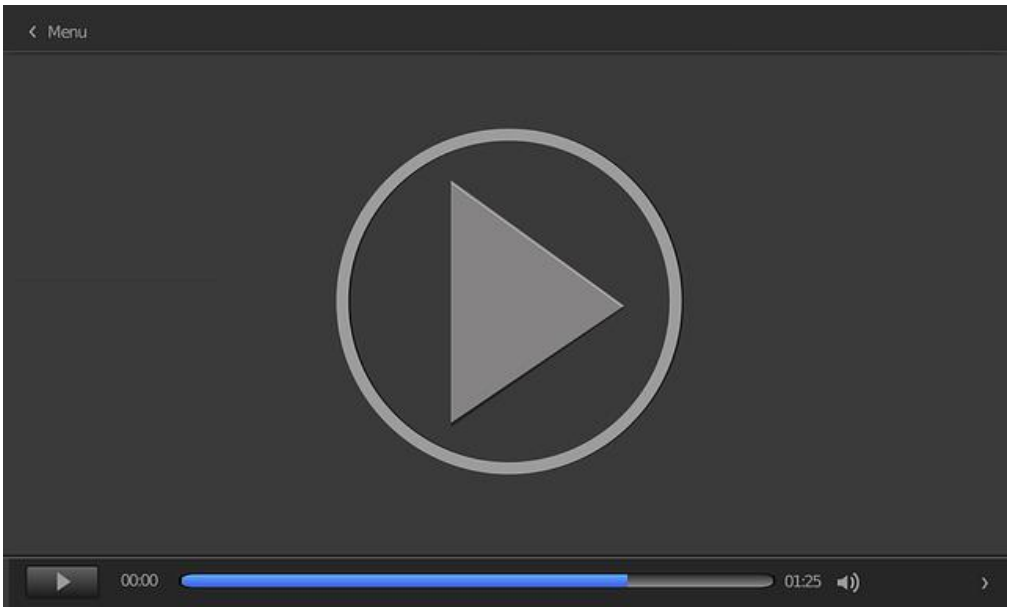
- ***Embeddings.*** En este caso se generan los *embeddings* correspondientes a una base de conocimiento especializada, y se almacenan en un *vector store.* Es posible realizar preguntas y búsquedas a través del *vector store.* La respuesta incluye los textos relacionados con la pregunta o textos similares a las palabras de búsqueda proporcionadas.

- **Indexación.** Consiste en proporcionar una colección de documentos al servicio Azure Cognitive Search. Este servicio se encarga de indexarlos y extraer la información relevante de cada uno de ellos. Esta información puede ser utilizada por el chatbot para completar las respuestas que proporciona con datos adicionales procedentes de los documentos indexados.

- **Resultados web.** Consiste en utilizar la API de Bing Web Search, el buscador de Microsoft, para completar las respuestas del modelo de OpenAI con información procedente de sitios web. Para ello, se generan consultas al buscador a partir de la pregunta realizada y se extrae la información relevante de los sitios web que retorna el buscador.

- **Bases de datos.** En este escenario se utiliza como fuente adicional una base de datos. El modelo de OpenAI se emplea para transformar las preguntas de los usuarios en consultas SQL que permite recuperar la información solicitada de las bases de datos conectadas.

Por último, en el siguiente vídeo, *Uso de visión artificial y llamadas a funciones en GPT-4 para la creación de agentes autónomos,* se incluye un ejemplo de uso de las capacidades de visión artificial y toma de decisiones de los modelos GPT-4. Para ello, se transforma un chatbot en un agente capaz de analizar automáticamente los paquetes devueltos por los clientes e iniciar la acción de respuesta oportuna.



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=ccd5bf99-6ec2-4e80-8c36-b165011d03f1

5.4. Desarrollo, orquestación e integración de large language models (LLM)

Introducción

En los últimos dos años han aparecido múltiples herramientas que extienden la funcionalidad de los servicios de Azure OpenAI. El *target* de estas herramientas son los usuarios más especializados, principalmente desarrolladores de *software.* Su principal finalidad es la **integración de OpenAI y otros LLMs en aplicaciones,** completando, por tanto, la arquitectura nativa de nube para IA generativa que describimos al principio del tema. A continuación, se incluye una descripción de las más populares.

LangChain

Es una plataforma *open-source* que permite desarrollar aplicaciones integradas con LLMs *(large language models),* incluyendo OpenAI. Proporciona un entorno de orquestación de funcionalidades de diferentes LLMs bajo una interfaz común.

Sus principales **funcionalidades** son:

- **Interfaz común para los LLMs de cualquier proveedor.** La lista completa puede consultarse en: https://python.langchain.com/docs/integrations/chat/
- **Generación y optimización de *prompts.*** Permite seleccionar e incorporar ejemplos al *prompt,* diseñarlo de acuerdo con una plantilla o integrar bloques separados en un único *prompt.*
- ***Chains.*** Una *chain* es una secuencia de llamadas para completar una determinada tarea. Es el equivalente a un *pipeline* analítico. Puede involucrar a un único LLM o a varios simultáneamente. LangChain proporciona una colección de *chains* para resolver las tareas más habituales y la posibilidad de crear *chains* personalizadas.

- ***Data augmented generation.*** Permite incorporar información de fuentes externas en la respuesta *(grounding).* También permite resumir textos para extraer la información relevante que puede completar el contenido de una respuesta.

- ***Agents.*** Los agentes son capaces de ejecutar acciones, observar el resultado y razonar acerca de la siguiente acción a tomar. Se diferencian de las *chains* en que estas son estáticas, es decir, siempre ejecutan la misma secuencia. Y como en las *chains,* existe una colección de agentes predefinida y la posibilidad de crear un agente personalizado.

- **Memoria.** LangChain implementa mecanismos para persistir el resultado de interacciones previas entre ejecuciones de *chains* o agentes. Esta información de estado permite construir interacciones más complejas que pueden arrojar resultados más precisos.

- **Evaluación.** LangChain implementa mecanismos para evaluar la calidad de los LLMs a través de *datasets* con *prompts* y respuestas esperadas.

En el siguiente ejemplo se muestra cómo utilizar LangChain con una *chain* sencilla, que genera un *prompt* y obtiene una respuesta. En este caso, se utiliza un LLM de Hugging Face. El *prompt* se construye a partir de un *template* y se especifica la temperatura (grado de creatividad) deseada para la respuesta:

```
from langchain.prompts import PromptTemplate

from langchain.llms import HuggingFace

from langchain.chains import LLMChain



prompt = PromptTemplate(

    input_variables=["city"],
```

```
    template="Describe a perfect day in {city}?",
)

llm = HuggingFace(
          model_name="gpt-neo-2.7B",
          temperature=0.9)

llmchain = LLMChain(llm=llm, prompt=prompt)
llmchain.run("Paris")
```

En el siguiente ejemplo se muestra cómo utilizar un agente para responder a la pregunta «¿Qué es LangChain?». Durante la ejecución, se imprimen mensajes que muestran el razonamiento utilizado por el agente para decidir la siguiente acción (I should...).

```
agent_executor.invoke({"input": "what is LangChain?"})
> Entering new AgentExecutor chain...

I should research LangChain to learn more about it.
Action: tavily_search_results_json
Action Input: "LangChain"[{'url': 'https://www.ibm.com/topics/langchain',
'content': 'LangChain is essentially a library of abstractions for Python
and Javascript … . It supports Python and Javascript languages and supports
various LLM providers, including OpenAI, Google, and IBM.'}]
```

```
I should read the summary and look at the different features and
integrations of LangChain.

Action: tavily_search_results_json

Action Input: "LangChain features and integrations"[{'url':
'https://www.ibm.com/topics/langchain', 'content': "LangChain provides
integrations for over 25 different embedding methods, ..."}]



I should take note of the launch date and popularity of LangChain.

Action: tavily_search_results_json

Action Input: "LangChain launch date and popularity"[{'url':
'https://www.ibm.com/topics/langchain', 'content': "LangChain is …"}]



I now know the final answer.

Final Answer: LangChain is an opensource orchestration framework for
building applications using large language models (LLMs) like chatbots and
virtual agents. It was launched by Harrison Chase in October 2022 and has
gained popularity as the fastest-growing opensource project on Github in
June 2023.



> Finished chain.
```

Semantic Kernel

Semantic Kernel es un SDK desarrollado por Microsoft, con funcionalidades similares a LangChain. Además de Python, soporta los lenguajes de programación C# y Java. Se basa en la ejecución de un **proceso con varios pasos** para completar cada solicitud. Dicho proceso se muestra en la Figura 7:

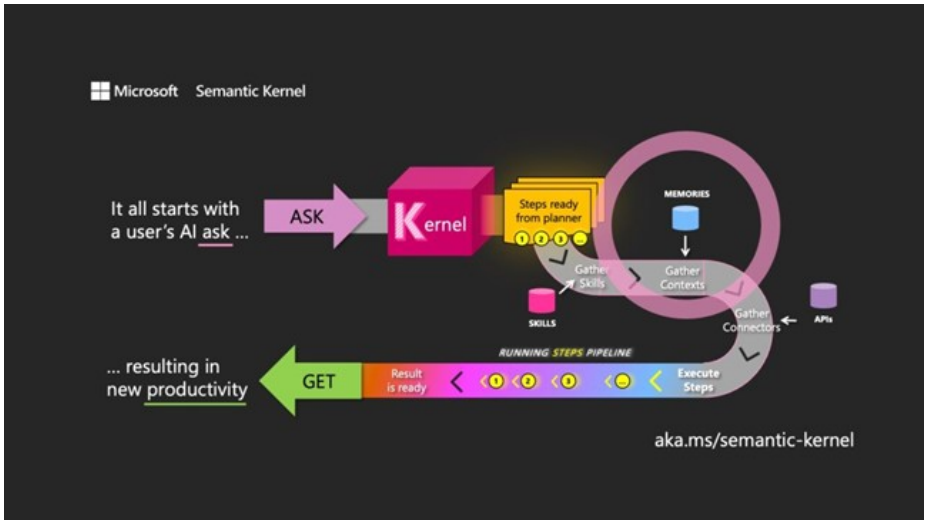


Figura 7. Pasos de Semantic Kernel para completar una petición. Fuente: Ramel, 2023.

A continuación, se describe cada **paso** del proceso:

- ***Ask.*** Se refiere a la petición o pregunta lanzada por el usuario.
- ***Kernel.*** Este componente interpreta la petición o pregunta y orquesta los siguientes pasos del proceso, coordinando al resto de componentes que participan en la generación de la respuesta.
- ***Planner.*** A partir de la interpretación realizada por el *kernel,* este componente genera una estrategia compuesta por un conjunto de acciones o *pipeline* para dar respuesta a la petición o pregunta original.
- ***Gather.*** En este paso se recopila la información necesaria de todas las fuentes relevantes. Se utilizan conectores para recuperar datos de documentos, buscadores o el contenido de memorias.

- ***Skills.*** También denominados *plugins.* Cada *plugin* contiene un conjunto de funciones relacionadas que realizan tareas útiles en el proceso de generación de la respuesta. Por ejemplo, el WriterPlugin contiene las siguientes funciones: generar una lista de ideas sobre un tema, escribir un *e-mail,* escribir un poema, traducir un texto o resumir un libro.

- **Conectores.** Son los componentes especializados en recuperar información de un tipo de fuente determinada.

- ***Inform.*** Se refiere a la presentación de la respuesta generada, que puede ser de tipo texto, imagen o audio.

- ***Pipelines.*** Secuencias de acciones *(steps)* predefinidas o generadas de forma dinámica por el *planner.*

- ***Get.*** En este punto se puede consultar información de bajo nivel generada durante el proceso. Por ejemplo, el contenido de las memorias del *kernel.*

Bot Framework

Bot Framework es la plataforma de creación de chatbots de Microsoft, anterior a la llegada de ChatGPT. Todavía se sigue utilizando para escenarios de negocio que no requieren la versatilidad de ChatGPT o para integrar ChatGPT en entornos como Microsoft Teams.

Los componentes de este *framework* son los siguientes:

- **Software Development Kit (SDK).** Permite diseñar e implementar bots a partir de código fuente en C#, Python, Java o Javascript.

- **Bot Framework Composer.** Interfaz gráfica para la generación de bots. Permite diseñar diálogos, respuestas y habilidades del bot.

- **Azure Bot Service.** Servicio utilizado para hospedar bots y conectarlo a servicios como Skype, Teams o Facebook Messenger. Proporciona funcionalidades relacionadas con analítica, depuración y seguridad.

- **Bot Framework Emulator.** Aplicación de escritorio utilizada para probar el bot antes de realizar su despliegue.

Power Platform

Power Platform es la plataforma *low code/no code* de Microsoft. Permite diseñar e implementar aplicaciones multiplataforma (web, móvil, escritorio) sin necesidad de utilizar código fuente. Existen **tres enfoques** para incorporar a una aplicación de Power Platform las capacidades de Azure OpenAI:

- Utilizar **Power Virtual Agents,** el componente de la plataforma especializado en la creación de chatbots, con el conector Azure OpenAI.

- Utilizar la funcionalidad de **Generative answers de Power Virtual Agents** para complementar las temáticas incorporadas al agente con respuestas construidas a partir de fuentes externas, incluyendo OpenAI.

- Utilizar la integración con OpenAI de **AI Builder,** el componente de Power Platform que permite utilizar modelos preentrenados o construir modelos propios para optimizar procesos de negocio.

5.5. Uso de bases de datos de vectores para búsqueda de datos relacionados

Como se ha indicado en secciones previas, la generación de *embeddings* es un mecanismo eficaz para establecer **similitudes entre textos o palabras** mediante la conversión a vectores numéricos. Por otro lado, el almacenamiento y consulta de *embeddings* en una base de datos permite contestar a preguntas sobre un tema o buscar información relacionada con él sin necesidad de que exista un *match* exacto entre la pregunta y la información almacenada en la base de datos. Esto es posible gracias al cálculo de similitud semántica que puede realizarse con *embeddings,* pero no con las palabras o textos originales.

En Azure OpenAI, existen **tres sistemas/servicios** soportados para la gestión de *embeddings:* Azure Cognitive Search (AI Search), CosmosDB y Redis. A continuación, se describen sus principales características.

Azure Cognitive Search (AI Search)

Este servicio implementa un buscador a partir de información propia de una organización. Esta información es indexada para que pueda ser recuperada rápidamente en respuesta a una búsqueda de un usuario.

La capacidad de *vector search* en el servicio Azure Cognitive Search es la que permite trabajar con *embeddings.*

Vector search proporciona las siguientes funcionalidades:

- **Búsqueda de información similar a la proporcionada como entrada.** Puede utilizarse como buscador interno de una organización o para la implementación de sistemas de recomendación.

- **Soporte de búsquedas multimodales,** que permiten recuperar texto e imágenes con la misma consulta.
- Combinación de búsquedas **basadas en *embeddings* con *match*** de palabras clave para maximizar la calidad y precisión de los resultados.
- Mediante el uso de modelos de *embeddings* entrenados en **múltiples idiomas,** es posible responder a búsquedas independientemente del idioma en el que sean escritas por el usuario.
- Posibilidad de definir filtros de texto y numéricos a partir de metadatos definidos sobre los *embeddings.* De esta forma, se pueden recuperar datos más precisos que cumplan todas las condiciones especificadas por el usuario.

Azure CosmosDB

Azure CosmosDB es el **servicio de bases de datos NoSQL** de Microsoft. Proporciona diferentes **interfaces** de acceso a la base de datos, incluyendo MongoDB, PostgreSQL y Cassandra. De esa forma, tanto usuarios como aplicaciones pueden utilizar CosmosDB como si se tratase de una de estas bases de datos, sin necesidad de realizar cambios en el código fuente para migrar a CosmosDB o aprender una sintaxis nueva para generar consultas. También implementa la capacidad *vector search,* de forma muy similar a Azure Cognitive Search.

Azure Cache for Redis

Este servicio está basado en la tecnología Redis, una base de datos NoSQL, que permite almacenar los datos en memoria. De esta forma, Redis proporciona latencias mínimas en el proceso de recuperación de información. Ha incorporado también la capacidad de *vector search.* Es la alternativa idónea para organizaciones que ya están usando el servicio y quieren explotar las ventajas de un *vector store.*

5.6. Cuaderno de ejercicios

1. A partir del SDK de GPT-4 Turbo with Vision, escribir el código fuente en Python que permite generar con OpenAI la descripción de una imagen.

Para ello, se debe inicializar el cliente de OpenAI con los valores de API *key* y *endpoint,* y crear una petición que especifique el *system message,* el *prompt* y la URL desde la que puede descargarse la imagen.

```
from openai import AzureOpenAI


api_base = os.getenv("AZURE_OPENAI_ENDPOINT")

api_key= os.getenv("AZURE_OPENAI_API_KEY")

deployment_name = 'deployment1'

api_version = '2023-12-01-preview' # this might change in the future


client = AzureOpenAI(

    api_key=api_key,

    api_version=api_version,

    base_url=f"{api_base}/openai/deployments/{deployment_name}"

)


response = client.chat.completions.create(
```

```
    model=deployment_name,
    messages=[
        { "role": "system", "content": "You are a helpful assistant." },
        { "role": "user", "content": [
            {
                "type": "text",
                "text": "Describe this picture:"
            },
            {
                "type": "image_url",
                "image_url": {
                    "url": "<image URL>"
                }
            }
        ] }
    ],
    max_tokens=2000
)
print(response)
```

2. Utilizando el SDK de OpenAI para Python, escribir el código fuente que permite lanzar el mismo *prompt* en tres ocasiones y obtener respuestas iguales o similares en todas ellos.

Para este ejercicio, es importante entender el concepto de *seed,* un parámetro de la llamada a client.chat.completions.create() , y el requisito de reproducible output .

```
import os

from openai import AzureOpenAI


client = AzureOpenAI(

  azure_endpoint = os.getenv("AZURE_OPENAI_ENDPOINT"),

  api_key=os.getenv("AZURE_OPENAI_API_KEY"),

  api_version="2024-02-01"

)


for i in range(3):

  print(f'Story Version {i + 1}\n---')


  response = client.chat.completions.create(

    model="gpt-35-turbo-0125", # Model = should match the deployment name
you chose for your 0125-preview model deployment

    seed=42,

    temperature=0.7,
```

```
    max_tokens =50,
    messages=[
      {"role": "system", "content": "You are a helpful assistant."},
      {"role": "user", "content": "Tell me a story about how the universe
began?"}
    ]
  )

  print(response.choices[0].message.content)
  print("---\n")

  del response
```

3. Utilizando el SDK de OpenAI para Python, escribir el código fuente que permite recuperar el identificador de un *deployment* actualmente operativo para un modelo especificado como parámetro (text-search-davinci-doc-001).

Para ello, se deben listar todos los *deployments* y comprobar cuál de los que se encuentran actualmente operativos (status == 'succeeded') se basa en el modelo especificado. En caso de que no exista ninguno, se debe crear un nuevo *deployment* para ese modelo con escalado (scale_type) estándar.

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

```
load_dotenv()

openai.api_type = "azure"

openai.api_base = "https://tutorial-openai-01-2023.openai.azure.com/"

openai.api_version = "2022-12-01"

openai.api_key = os.getenv("OPENAI_API_KEY")



desired_model = "text-search-davinci-doc-001"

deployment_id = None

result = openai.Deployment.list()



for deployment in result.data:

    if deployment["status"] != "succeeded":

        continue

    model = openai.Model.retrieve(deployment["model"])

    if model["id"] == desired_model:

        deployment_id = deployment["id"]



if not deployment_id:

    result = openai.Deployment.create(model=desired_model, scale_settings=
{"scale_type":"standard"})
```

```
    deployment_id = result["id"]
```

4. Utilizando LangChain, escribir el código fuente en Python que permite generar un *prompt* para OpenAI con la pregunta "¿En qué año se firmó la constitución española?", y obtener la respuesta generada. Para ello, se deben fijar los valores de las variables de entorno OPENAI_API_KEY y OPENAI_ORGANIZATION, definir el *template* del *prompt,* inicializar una LLMChain y enviar el *prompt* con la función run().

```
import os

os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY

os.environ["OPENAI_ORGANIZATION"] = OPENAI_ORGANIZATION


from langchain.chains import LLMChain

from langchain_core.prompts import PromptTemplate

from langchain_openai import OpenAI


template = """Question: {question}"""

prompt = PromptTemplate.from_template(template)

llm = OpenAI()

llm_chain = LLMChain(prompt=prompt, llm=llm)

question = "¿En qué año se firmó la constitución española?"

llm_chain.run(question)
```

5. Utilizando LangChain, escribir el código fuente en Python para enviar un *prompt* a OpenAI con la clase ChatOpenAI y el método `invoke()`. Extraer de la respuesta e imprimir los valores de *logprobs,* es decir, la probabilidad de los tókenes de la secuencia dado el contexto.

Los *logprobs* son una indicación de la confianza del modelo en la calidad de la respuesta. Para más información sobre logprobs: Hills, J. y Anadkat, S. (2023, diciembre 19). *Using logprobs.* OpenAI. https://cookbook.openai.com/examples/using_logprobs

---

```
from langchain_openai import ChatOpenAI

import getpass

import os


os.environ["OPENAI_API_KEY"] = getpass.getpass()


llm = ChatOpenAI(model="gpt-3.5-turbo-0125").bind(logprobs=True)

msg = llm.invoke(("human", "how are you today"))

msg.response_metadata["logprobs"]["content"][:5]
```

5.7. Referencias bibliográficas

Aahill, mrbullwinkle (2023, agosto 7). Use the Azure OpenAI web app. Microsoft. https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/use-web-app

Ayala, F. (2023, julio 24). *Azure OpenAI Landing Zone reference architecture.* Microsoft. https://techcommunity.microsoft.com/t5/azure-architecture-blog/azure-openai-landing-zone-reference-architecture/ba-p/3882102?WT.mc_id=academic-0000-abartolo

Castillo, F. (2023, febrero 2). Embeddings: Meaning, Examples and How To Compute. Arize.https://arize.com/blog-course/embeddings-meaning-examples-and-how-to-compute/

Hills, J. y Anadkat, S. (2023, diciembre 19). *Using logprobs.* OpenAI. https://cookbook.openai.com/examples/using_logprobs

mrbullwinkle, dbradish-microsoft, Pookam90, erc-urban, varun-dhawan, mgreenegit, jimmystridh, HeidiSteen, nitinme, MSFTeegardem, learn-build-servce-prod[bot], gahl-levy (2024, marzo 20). https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/embeddings?tabs=python-new

mrbullwinkle, KarlErickson, nitinme, mgreenegit, dbradish-microsoft, eric-urban, v-alju (2023, noviembre 12). https://learn.microsoft.com/en-us/azure/ai-services/openai/chatgpt-quickstart?tabs=command-line%2Cpython-new&pivots=programming-language-studio

Sánchez, A. (2024). Azure Openai for cloud native applications: Designing, Planning, and Implementing Generative AI Solutions. O'Reilly Media.

Fundamentals of Azure OpenAI Service

Microsoft Learn (s. f.). *Fundamentals of Azure OpenAI Service.* https://learn.microsoft.com/en-us/training/modules/explore-azure-openai/

Este curso oficial de Microsoft permite profundizar en varios de los conceptos descritos en este tema. Además, proporciona una visión integral de las capacidades de OpenAI disponibles en Azure e incluye referencias a documentación técnica detallada para el alumno que quiera adquirir más experiencia práctica.

Artificial intelligence on Microsoft Azure

Mcrosoft (2024, marzo 21). *Artificial intelligence on Microsoft Azure.* Coursera. https://www.coursera.org/learn/artificial-intelligence-microsoft-azure

Este curso creado por Microsoft en Coursera presenta una visión de conjunto acerca de la inteligencia artificial bajo la plataforma de computación en la nube Azure. Describe conceptos menos técnicos y más relacionados con el negocio, así como con las estrategias óptimas para adoptar la IA generativa en una organización.

Test

1. ¿Cuáles son los beneficios para la IA generativa de una arquitectura nativa de nube?

A. Clasificación de textos, chatbots, generación de imágenes y traducción.

B. Capacidad de computación, agilidad, innovación e interoperabilidad.

C. Menor coste de adquisición de infraestructura y más potencia de computación.

D. Capacidad de computación, agilidad e innovación.

2. ¿Cuál es la diferencia entre los modelos de OpenAI Codex y DALL-E?

A. El primero se utiliza para generar imágenes a partir de una descripción, el segundo para generar código fuente a partir de lenguaje natural.

B. El primero se utiliza para generar código fuente a partir de lenguaje natural, el segundo para generar imágenes a partir de una descripción.

C. Ambos se utilizan para implementar chatbots avanzados.

D. Son diferentes versiones del mismo modelo que procesa lenguaje natural.

3. ¿Qué es un *embedding* y qué aplicaciones tiene?

A. Es una representación matemática de una palabra o texto. Se utiliza para agilizar las respuestas de un buscador, dado que es más rápido procesar números que texto.

B. Es una representación matemática de una palabra o texto. Se utiliza para detectar textos similares, búsqueda de información sobre un tema o generación de código fuente a partir de lenguaje natural.

C. Es el valor numérico asociado al significado de una palabra o texto. Se utiliza para detectar textos similares, búsqueda de información sobre un tema o generación de código fuente a partir de lenguaje natural.

D. Es una forma de codificación de palabras y texto que permite almacenar información sobre un tema en menos espacio que su equivalente en texto.

4. ¿Para qué se utiliza un Playground en Azure OpenAI Studio?

A. Es el entorno utilizado por usuarios avanzados para desarrollar soluciones de IA generativa en Azure.

B. Para realizar experimentos y pruebas en el proceso de incorporar IA generativa a una aplicación, a través de una interfaz gráfica.

C. Es un simulador de IA generativa que sirve para aprender los conceptos esenciales, pero no permite construir soluciones reales.

D. Para realizar experimentos y pruebas en el proceso de incorporar IA generativa a una aplicación, mediante código fuente.

5. ¿Qué significado tiene el valor de temperatura en un LLM?

A. Representa el número de tókenes utilizado en la respuesta.

B. Representa el grado de determinismo o creatividad en la respuesta a un *prompt*.

C. Representa en qué medida aparece en la respuesta contenido violento o inadecuado.

D. Representa el tiempo invertido por el chatbot en generar la respuesta.

6. ¿Qué procedimiento se debe seguir para desplegar en una aplicación web un chatbot basado en IA generativa construido en un *playground*?

A. Escribir un *template* de despliegue basado en Azure Resource Manager, y utilizar un agente de despliegue para alojarlo en una URL.

B. Pulsar el botón «Deploy to» en Azure OpenAI Studio, y proporcionar un nombre y perfil de *hardware* para la aplicación.

C. El procedimiento de despliegue implica utilizar el SDK de Python para dar las órdenes de despliegue a través de la API de Azure.

D. Pulsar el botón «Deploy to» en Azure OpenAI Studio y especificar un JSON con las características del despliegue.

7. ¿En qué consiste el *fine-tuning* de un modelo?

A. El proceso de *fine-tuning* se utiliza para incorporar bases de conocimiento nuevas sobre temáticas especializadas, codificando reglas que determinan en qué casos se debe utilizar el nuevo contenido.

B. El proceso de *fine-tuning* se utiliza para incorporar bases de conocimiento nuevas sobre temáticas especializadas, realizando un reentrenamiento de uno de los modelos proporcionados por OpenAI.

C. El proceso de *fine-tuning* se utiliza para corregir respuestas inapropiadas a determinados *prompts.*

D. El proceso de *fine-tuning* se utiliza para reducir la complejidad de las respuestas y las posibles alucinaciones del chatbot.

8. ¿Cuál de las siguientes es una característica de LangChain?

A. La posibilidad de actuar como *vector store* a partir de una base de datos PostgreSQL.

B. La disponibilidad de una interfaz común para trabajar con cualquiera de los principales LLMs.

C. La posibilidad de escribir código fuente en Python, C#, Java y GoLang.

D. La disponibilidad de un entorno de *dashboarding* para medir la evolución de la precisión en las respuestas de los modelos LLM utilizados.

9. ¿A qué se refiere el término *data augmented generation* (DAG)?

A. Se refiere a los mecanismos que permiten realizar actividades de orquestación entre diferentes LLMs.

B. Se refiere a los mecanismos que permiten incorporar información de fuentes externas en la respuesta.

C. Se refiere a los mecanismos que permiten generar *embeddings* para almacenarlos en un *vector store.*

D. Se refiere a la posibilidad de mejorar un modelo, incorporando nuevos ejemplos.

10. ¿En qué se diferencia un *agent* de un *chain* de LangChain?

A. No existe ninguna diferencia, son términos intercambiables.

B. Los agentes son capaces de ejecutar diferentes secuencias de acciones en función de los resultados obtenidos en acciones previas, mientras que las *chains* ejecutan una secuencia estática de acciones.

C. Los agentes ejecutan una secuencia estática de acciones, mientras que las *chains* son capaces de ejecutar diferentes secuencias de acciones en función de los resultados obtenidos en acciones previas.

D. Una *chain* es una versión especializada de un agente para un determinado LLM.