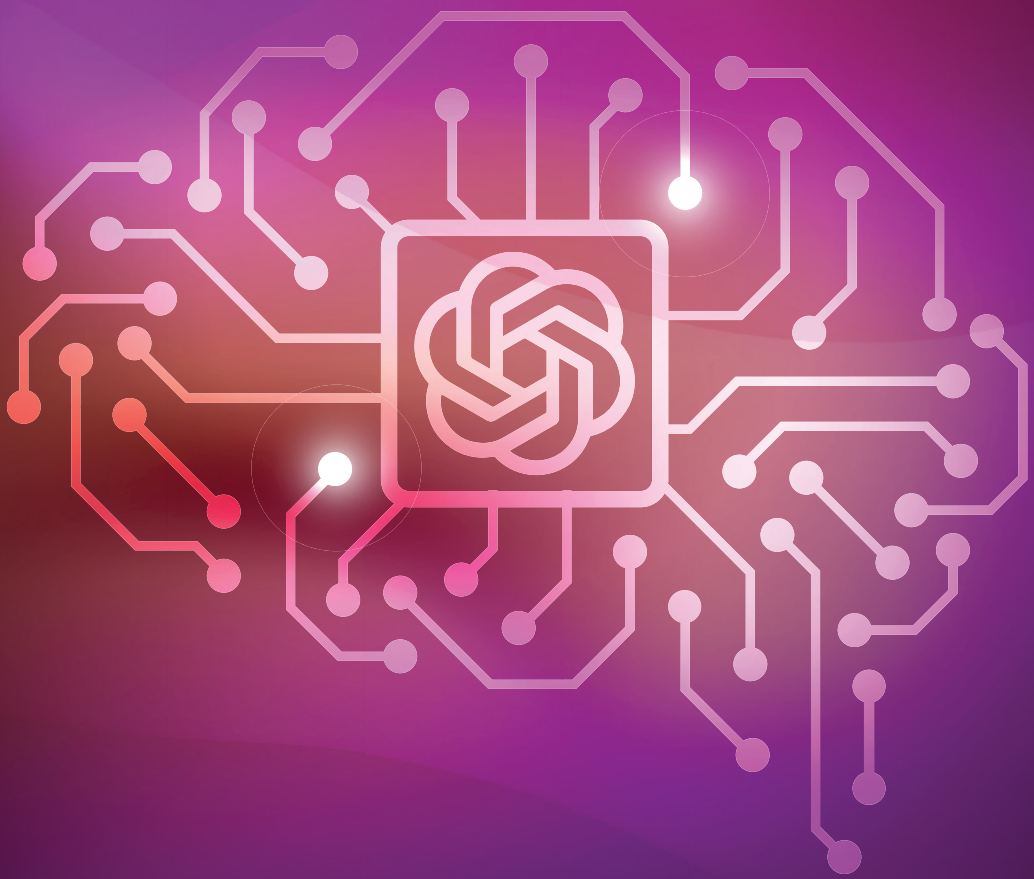
ANAYA MULTIMEDIA

# Inteligencia artificial generativa con modelos de ChatGPT y OpenAI

Aproveche las capacidades de los LLM de OpenAI para mejorar la productividad y fomentar la innovación con GPT-3 y GPT-4



VALENTINA ALTO

# ÍNDICE DE CONTENIDOS

# PARTE 1

## FUNDAMENTOS DE LOS MODELOS DE INTELIGENCIA ARTIFICIAL GENERATIVA Y GPT

La primera parte de este libro presenta los fundamentos de los modelos de inteligencia artificial generativa y GPT, incluyendo una breve historia del desarrollo de OpenAI y su serie de modelos estrella, la familia GPT.

Esta parte comienza con una visión general del dominio de la IAG, ofreciendo conocimientos básicos sobre esta área de investigación de la inteligencia artificial, así como sobre su historia y sus avances de vanguardia. De igual modo, el lector se familiarizará con las aplicaciones de la IAG, desde generación de textos hasta composición de música.

A continuación, hablamos sobre la compañía que puso el poder de la IAG a disposición del público en general: OpenAI. Conoceremos la tecnología del lanzamiento más conocido de OpenAI, ChatGPT, y lograremos entender el viaje de investigación que, partiendo de las redes neuronales artificiales (RNA), dio lugar a los grandes modelos de lenguaje o LLM (*Large Language Models*).

Esta parte contiene los siguientes capítulos:

- Capítulo 1: Introducción a la inteligencia artificial generativa (IAG).
- Capítulo 2: OpenAI y ChatGPT: Más allá del despliegue publicitario.

misma pregunta. Por ejemplo, si se le pregunta al modelo: "¿Qué es OpenAI?" varias veces con la temperatura fijada en 0, siempre dará la misma respuesta. Por otro lado, si se repite lo mismo con un modelo con una temperatura de 1, intentará modificar sus respuestas cada vez tanto en palabras como en estilo.

- **Longitud máxima (*Maximum length*):** Varía de 0 a 2.048, y controla la longitud (en términos de *tokens*) de la respuesta del modelo al *prompt* del usuario.
- **Detener secuencias (*Stop sequences*):** Con una información introducida por el usuario, logra que las respuestas finalicen en el momento deseado, como por ejemplo al final de una frase o de una lista.
- **Máximas probabilidades (*Top probabilities* o *Top P*):** Con un valor entre 0 y 1, controla los *tokens* que el modelo tendrá en cuenta al generar una respuesta. Si se configura en 0.9 considerará el 90 % más probable de todos los posibles *tokens*. Quizá se pregunte lo siguiente: "¿Por qué no configurar las máximas probabilidades en 1, de forma que se elijan todos los *tokens* más probables?". La respuesta es que los usuarios podrían seguir deseando conservar la variedad cuando el modelo tiene una confianza baja, incluso en el caso de los *tokens* con la mayor puntuación.
- **Penalización por frecuencia (*Frequency penalty*):** Con un valor entre 0 y 2, controla la repetición de los mismos *tokens* en la respuesta generada. Cuanto mayor sea la penalización, menores son las probabilidades de ver los mismos *tokens* más de una vez en la misma respuesta. La penalización reduce la posibilidad de manera proporcional, según la frecuencia con la que un *token* ha aparecido en el texto hasta el momento (que es la diferencia clave con respecto al siguiente parámetro).
- **Penalización por presencia (*Presence penalty*):** Con un valor similar, es también parecido al anterior parámetro, pero más estricto. Reduce la posibilidad de repetir cualquier *token* que haya aparecido en el texto hasta el momento. Como es más estricto que la penalización por frecuencia, este parámetro aumenta además la probabilidad de introducir nuevos temas en una respuesta.
- **Mejor de todos (*Best of*):** Con un valor entre 0 y 20, genera varias respuestas, mostrando solamente la que tiene la mejor probabilidad total de todos sus *tokens*.
- **Texto previo y posterior a la pregunta (*Pre- and post-response text*):** Teniendo como valor una información introducida por el usuario, inserta texto antes y después de la respuesta del modelo, lo que le ayuda en la preparación de la misma.

Además de probar los modelos de OpenAI en el Playground, el usuario siempre puede llamar a las API de los modelos en su código personalizado e integrar los modelos en sus aplicaciones. De hecho, en la esquina derecha del Playground es posible hacer clic en View code (Ver código) y exportar la configuración, como se muestra en la figura 2.8.



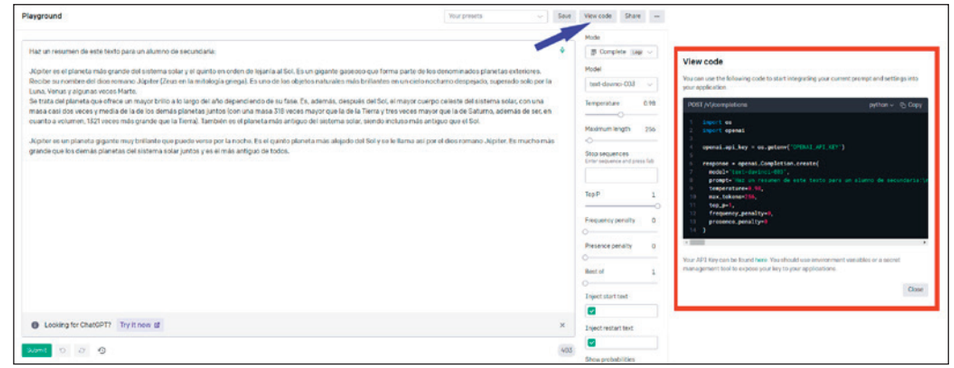



**Figura 2.8.** Código de Python para llamar a un modelo de GPT-3 con un *prompt* en lenguaje natural.

Como puede verse en la imagen anterior, el código exporta la configuración de parámetros establecida en el Playground.

A continuación, ya es posible empezar a utilizar la librería de OpenAI en Python instalándola mediante el comando `pip install openai`. Para usar los modelos será necesario generar una clave API. El usuario puede encontrar sus claves API (`https://platform.openai.com/account/api-keys`) en la configuración de su cuenta, como se muestra en la figura 2.9.

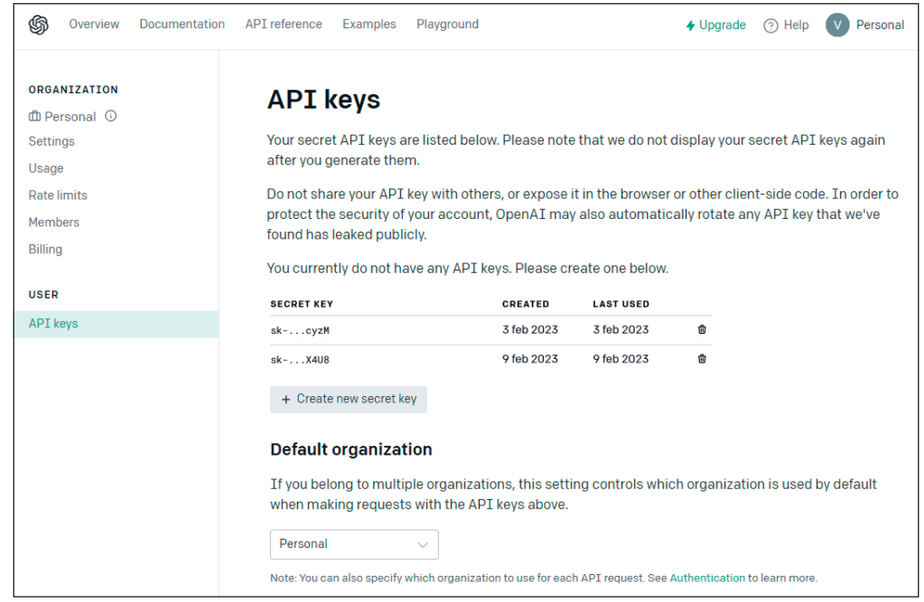



**Figura 2.9.** Claves API de la página de configuración de la cuenta de un perfil de OpenAI.

Con las API de OpenAI, es posible probar las siguientes familias de modelos adicionales no disponibles en el Playground:

- ***Moderation* (moderación):** Se trata de un modelo de ajuste fino desarrollado por OpenAI que puede detectar contenido de texto sensible o no seguro. *Moderation* usa algoritmos de aprendizaje automático para clasificar texto como seguro o no seguro según su contexto y uso del lenguaje. Este modelo se emplea para automatizar la moderación del contenido en plataformas de redes sociales, comunidades en línea y en muchos otros dominios. Hay muchas categorías, como odio, odio/amenazas, autolesiones, sexo, sexo/menores, violencia, violencia/violencia gráfica, etc.
  Este es un código de ejemplo para la API de *Moderation*:

```
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
openai.Moderation.create(
  input="I want to kill him",
)
```

El resultado de esto es el siguiente:

```
<OpenAIObject id=modr-6sHusuY9frxJdfqTBXHsOAfWhckrh at
0x218bd8482c0> JSON: {
  "id": "modr-6sHusuY9frxJdfqTBXHsOAfWhckrh",
  "model": "text-moderation-004",
  "results": [
    {
      "categories": {
        "hate": false,
        "hate/threatening": false,
        "self-harm": false,
        "sexual": false,
        "sexual/minors": false,
        "violence": true,
        "violence/graphic": false
      },
      "category_scores": {
        "hate": 1.7164344171760604e-05,
        "hate/threatening": 2.614225103059198e-08,
        "self-harm": 2.5988580176772302e-08,
        "sexual": 2.8184256279928377e-06,
        "sexual/minors": 9.1383149936064e-09,
        "violence": 0.9910049438476562,
        "violence/graphic": 5.316753117767803e-07
      },
      "flagged": true
    }
  ]
}
```

En este caso, la API detectó evidencias de contenido violento.



- ***Embeddings* (incrustaciones):** Algunos modelos pueden utilizar incrustaciones, que implican la representación de palabras o frases en un espacio de varias dimensiones. Las distancias matemáticas entre distintas instancias en este espacio representan su parecido en términos de significado. Como ejemplo, imaginemos las palabras reina, mujer, rey y hombre. En nuestro espacio multidimensional, donde las palabras son vectores, si la representación es correcta, en condiciones ideales querríamos conseguir el ejemplo de la figura 2.10.

$$\vec{rey} + (\vec{mujer} - \vec{hombre}) \approx \vec{reina}$$

**Figura 2.10.** Ejemplo de ecuaciones vectoriales entre palabras.

Esto significa que la distancia entre mujer y hombre debe ser igual a la distancia entre reina y rey. Este es un ejemplo de una incrustación:

```
import openai
embedding = openai.Embedding.create(
    input="El gato está sobre la mesa",
    model="text-embedding-ada-002")["data"][0][ "embedding"]
```

El método anterior crea una representación vectorial del texto de entrada. Podemos ver los primeros 10 vectores del resultado a continuación:

```
embedding[1:10]

[-0.01369840931147337,
 -0.007505378685891628,
 -0.002576263388618827,
 -0.014773285016417503,
 0.019935185089707375,
 -0.01802290789783001,
 -0.01594814844429493,
 -0.0010944041423499584,
 -0.014323337003588676]
```

Las incrustaciones pueden ser de gran utilidad en situaciones de búsqueda inteligentes. De hecho, disponiendo de la incrustación de la entrada del usuario y de los documentos en los que el usuario desea buscar, es posible calcular la métrica de la distancia (es decir, la similitud del coseno) entre la entrada y los documentos. Haciendo esto, existe la posibilidad de recuperar los documentos más próximos, en términos de distancia matemática, a la entrada del usuario.

- ***Whisper* (susurro):** Se trata de un modelo de reconocimiento de voz que puede transcribir audio en texto. *Whisper* puede reconocer y transcribir varios idiomas y dialectos con una elevada precisión, lo que le convierte en una valiosa herramienta para sistemas de reconocimiento de voz automatizados. Aquí tenemos un ejemplo:

```
# Nota: es necesario utilizar OpenAI Python v 0.27.0 para que
el siguiente código funcione
import openai
```

```
openai.api_key = os.getenv("OPENAI_API_KEY")
audio_file= open("/path/to/file/audio.mp3", "rb")
transcript = openai.Audio.transcribe("whisper-1", audio_file)
```

El resultado tiene este aspecto:

```
{"text": Sí, hola, acabo de observar una abolladura en el lateral de mi coche
y no tengo ni idea de cómo se ha producido. No hubo testigos
y me siento muy frustrado.
....
}
```

Todos los modelos anteriores vienen como prefabricados, es decir, que ya han sido preentrenados con una enorme base de conocimiento.

No obstante, hay formas de personalizar y ajustar aún más los modelos para situaciones de uso particulares.

El primer método está integrado en el modo en que el modelo está diseñado, e implica proporcionarle el contexto en el enfoque de aprendizaje *few-shot* (trataremos esta técnica más adelante en el libro).

En otras palabras, podríamos pedirle al modelo que generase un artículo cuya plantilla y diccionario recuerdan a otro que ya hemos escrito. Para ello, introduciríamos en el modelo la consulta de generar un artículo y añadiríamos el anterior escrito a modo de referencia o contexto, de manera que el modelo esté mejor preparado para dar respuesta a la petición. La figura 2.11 muestra un ejemplo.

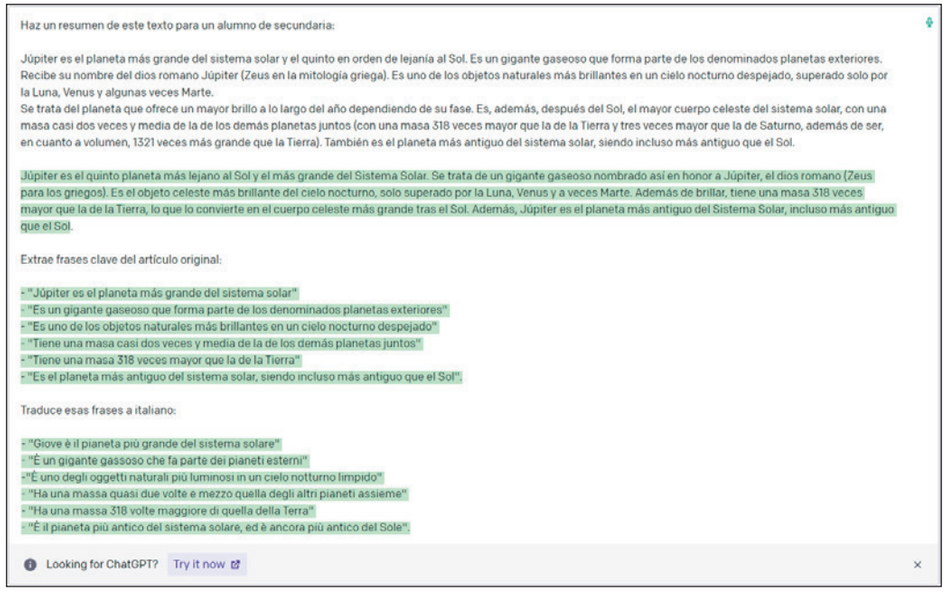



**Figura 2.11.** Un ejemplo de una conversación dentro del Playground de OpenAI con el enfoque de aprendizaje *few-shot*.



El segundo método es más sofisticado y se llama *fine-tuning* o ajuste fino; es el proceso de adaptar un modelo preentrenado a una nueva tarea.

En el ajuste fino, los parámetros del modelo preentrenado son alterados, o bien ajustando los existentes, o añadiendo otros nuevos, para refinar mejor los datos para la nueva tarea. Esto se lleva a cabo entrenando al modelo con un conjunto de datos etiquetado más pequeño y específico para la nueva tarea. La idea de este método es aprovechar el conocimiento aprendido del modelo preentrenado y ajustarlo al máximo a la tarea nueva, en lugar de entrenar un modelo desde cero. Veamos la figura 2.12.

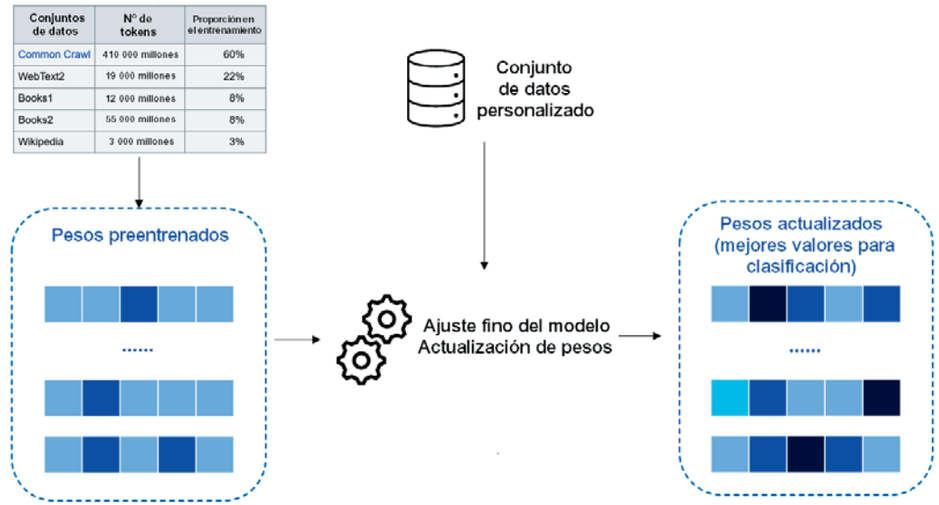



**Figura 2.12.** Ajuste fino de un modelo.

En esta figura vemos un esquema del funcionamiento del ajuste fino en modelos prefabricados de OpenAI. La idea es que disponemos de un modelo preentrenado con pesos o parámetros de uso general, al que alimentamos con datos personalizados, normalmente en forma de *prompts* de clave-valor y mensajes de completación, como vemos aquí:

```
{"prompt": "<texto del prompt>", "completación": "<texto generado ideal>"}
{"prompt": "<texto del prompt>", "completación": "<texto generado ideal>"}
{"prompt": "<texto del prompt>", "completación": "<texto generado ideal>"}
...
```

Una vez realizado el entrenamiento, dispondremos de un modelo personalizado que funciona con especial eficacia para una determinada tarea, por ejemplo, la clasificación de la documentación de una empresa.

Lo bonito del ajuste fino es que permite crear modelos prefabricados, ajustados a determinados casos prácticos, sin la necesidad de empezar desde cero, con la ayuda de conjuntos de datos de menor tamaño y, por tanto, empleando menos tiempo

# 4

# COMPRENDA EL DISEÑO DE *PROMPTS*

En capítulos anteriores, hemos mencionado el término *prompt* varias veces para hacer referencia a los textos o mensajes introducidos por el usuario en ChatGPT y en los modelos de OpenAI en general.

Este capítulo se centra con más detalle en la importancia del diseño y la ingeniería de *prompts* como técnica para aumentar la precisión del modelo. Los *prompts* tienen un enorme impacto en el resultado generado por el modelo, de ahí que un *prompt* bien diseñado logra guiar al modelo hacia la generación de resultados pertinentes y precisos, mientras que otro mal diseñado producirá probablemente una respuesta irrelevante o confusa. Asimismo, es también importante incorporar al mensaje consideraciones éticas para evitar que el modelo genere contenido dañino.

En este capítulo hablaremos de los siguientes temas:

- ¿Qué es un *prompt* y por qué es importante?
- Aprendizajes *zero-shot*, *one-shot* y *few-shot*, típicos de los modelos de transformadores.
- Principios de unos *prompts* bien definidos para obtener resultados significativos y consistentes.
- Evitar el riesgo de parcialidad oculta y tener en cuenta consideraciones éticas en ChatGPT.

Una vez finalizado el capítulo, el lector podrá obtener resultados de elevada calidad de sus interacciones con ChatGPT y con los modelos de OpenAI gracias a un diseño adecuado de *prompts*.

Empecemos con una sugerencia general sobre cómo podemos lograr un día más productivo (véase la figura 5.1).

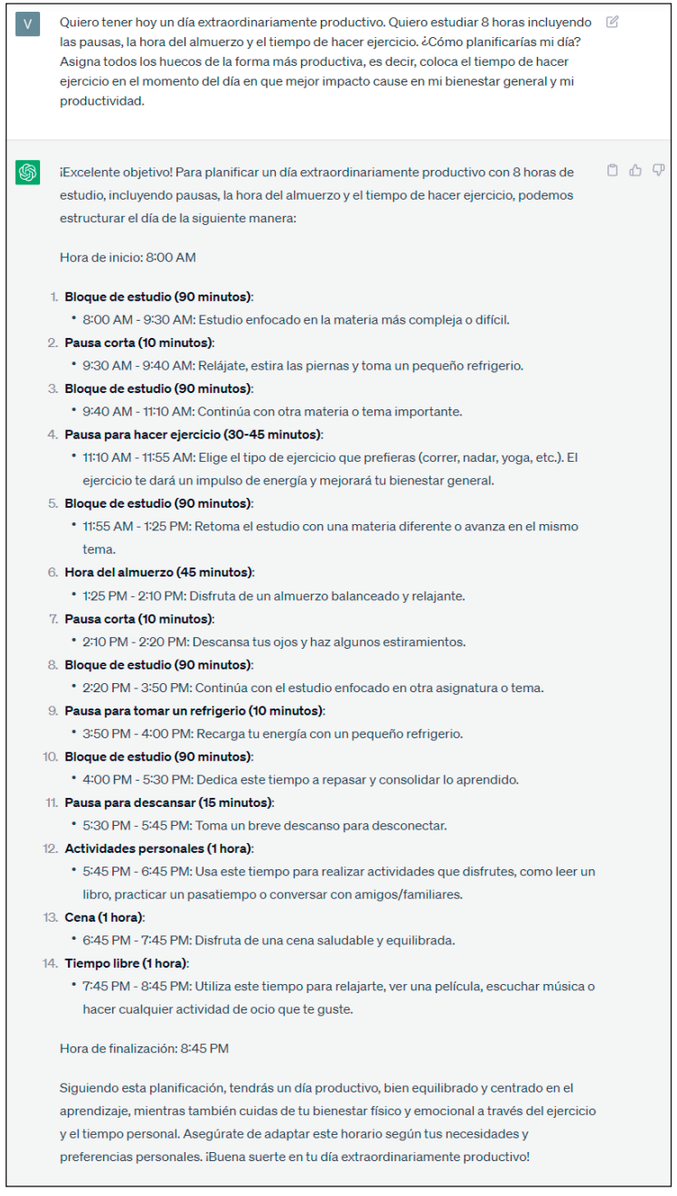



**Figura 5.1.** Un ejemplo de ChatGPT generando una rutina productiva.



La rutina anterior es bastante genérica, puesto que no se centra en una lista de actividades específica (excepto estudiar). Desafiemos a ChatGPT a crear una rutina más orientada a las tareas, incluyendo una lista de actividades que tenemos que llevar a cabo a lo largo del día (véase la figura 5.2).

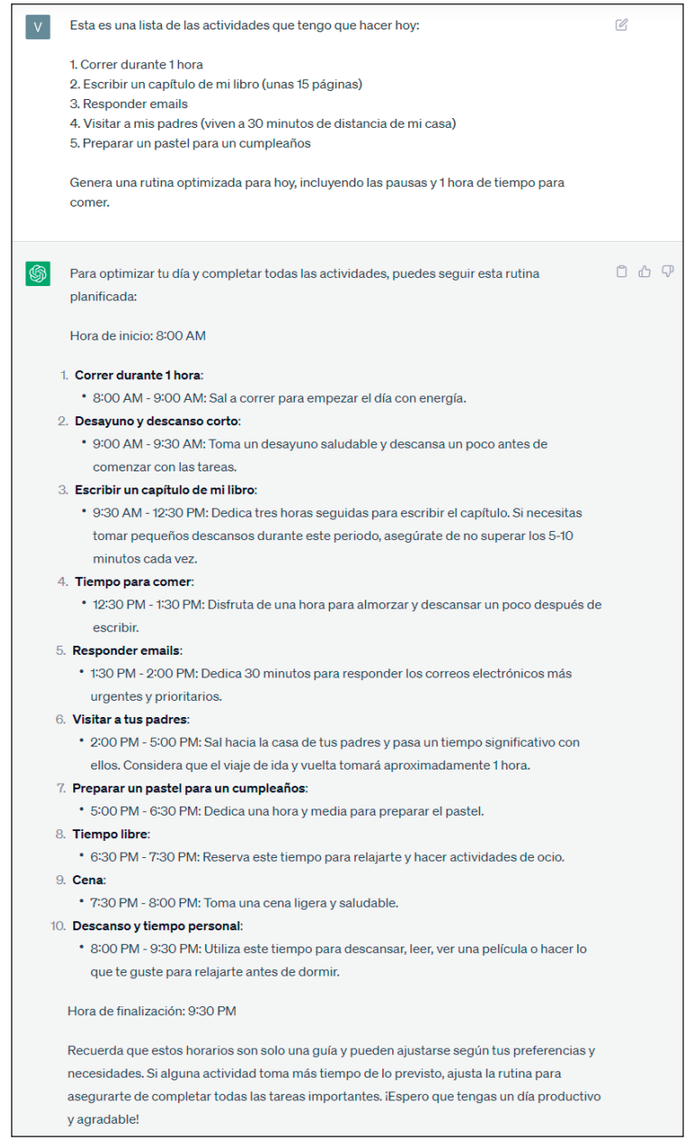



**Figura 5.2.** Rutina optimizada generada por ChatGPT según una lista de actividades.

ChatGPT para desarrolladores van mucho más allá de este ejemplo. Puede hacer las veces de asistente diario para la generación, explicación y depuración de código. Entre los lenguajes más conocidos tenemos Python, JavaScript, SQL y C#. Pero ChatGPT trata una gama mucho más amplia de lenguajes, tal y como revela por sí mismo en la figura 6.1.

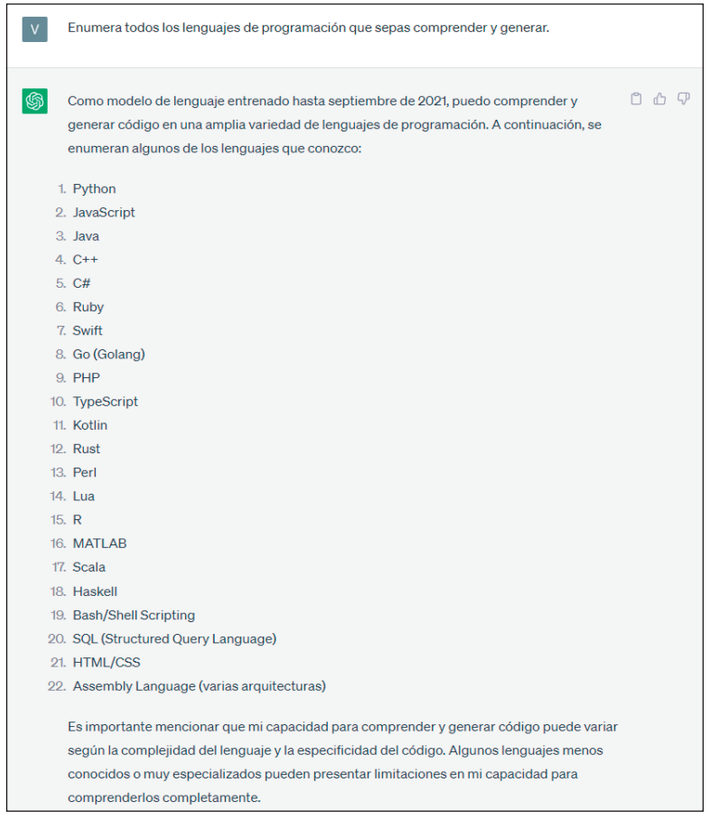



**Figura 6.1.** ChatGPT enumera los lenguajes de programación que es capaz de comprender y generar.

No importa si hablamos de desarrolladores de la parte visual o del servidor, de científicos o ingenieros de datos, o de alguien que trabaje con un lenguaje de programación; en todos los casos, ChatGPT puede cambiar las reglas del juego; en los distintos ejemplos de las próximas secciones veremos cómo.

A partir de la siguiente sección, nos adentraremos en ejemplos concretos de lo que ChatGPT puede conseguir cuando trabaja con código. Veremos situaciones de uso completas que tratan distintos ámbitos, de manera que podamos familiarizarnos con el uso de ChatGPT como asistente de código.



## Generar, optimizar y depurar código

La principal habilidad de ChatGPT que hay que aprovechar es la generación de código. ¿Cuántas veces hemos buscado un fragmento de código ya escrito para empezar? ¿Que hay que generar funciones `utils`, conjuntos de datos de muestra, esquemas en SQL, etc.? Ningún problema; ChatGPT es capaz de generar código a partir de una entrada de texto en lenguaje natural (véase la figura 6.2).

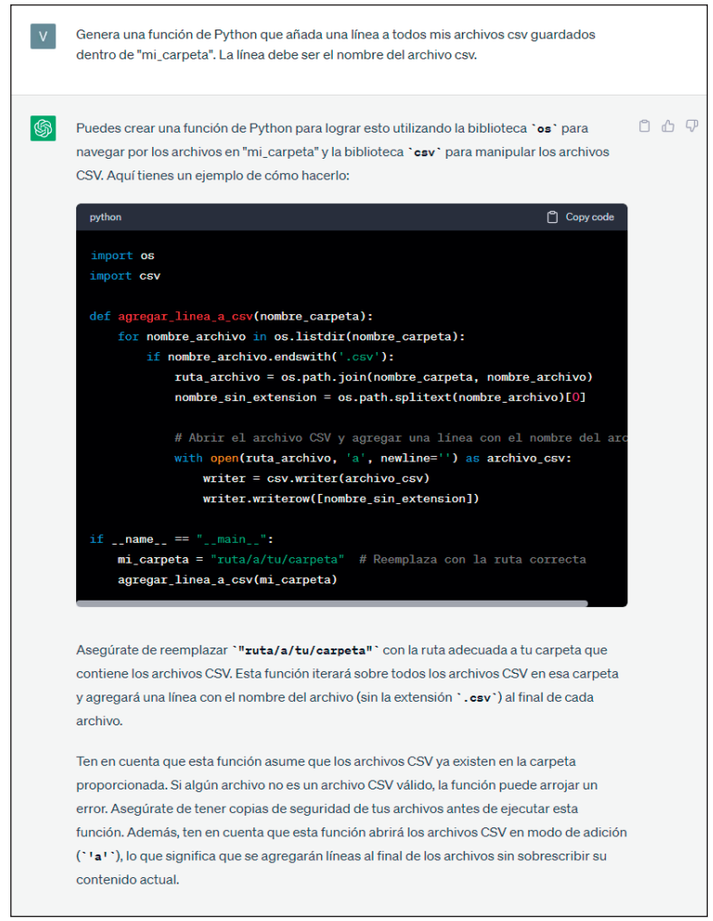



**Figura 6.2.** Ejemplo de ChatGPT generando una función de Python que escribe en archivos CSV.

Como se puede observar, ChatGPT no solo generó la función, sino que además fue capaz de explicar lo que hace, cómo utilizarla y con qué hay que sustituir elementos genéricos como `mi_carpeta`.

ChatGPT proporcionó un ejemplo para lograr los mismos resultados con un tiempo de ejecución inferior. Asimismo, elaboró la alternativa con una explicación clara de la razón por la que el método de las comprensiones de listas es más rápido que un bucle `for`. En la figura 6.7 comprobamos el rendimiento de este nuevo código comparado con el anterior.

```
from timeit import default_timer as timer
from datetime import timedelta

start = timer()

#mi código
elements = list(range(1_000_000))
data = []
for el in elements:
      if el % 2 != 0:
        data.append(el)

end = timer()
print(timedelta(seconds=end-start))

start = timer()

#script generado por ChatGPT
elements = list(range(1_000_000))
data = [el for el in elements if el % 2 != 0]

end = timer()
print(timedelta(seconds=end-start))
```

```
0:00:00.141185
0:00:00.073695
```

**Figura 6.7.** Comparación de los tiempos de ejecución de los códigos del usuario y de ChatGPT.

Como se puede comprobar, el segundo método (el generado por ChatGPT) ofrece una reducción en el tiempo de ejecución de más o menos un 47,8 %.

Además de la generación y optimización de código, ChatGPT se puede aprovechar también para explicar y depurar errores. A veces los errores son difíciles de interpretar; de aquí que una explicación en lenguaje natural pueda ser de utilidad para identificar el problema y encontrar la solución. Por ejemplo, al ejecutar un archivo `.py` desde la línea de comandos, aparece el siguiente error:

```
2023-03-25 11:27:10.270 Uncaught app exception
Traceback (most recent call last):
  File "C:\Users\vaalt\Anaconda3\lib\site-packages\streamlit\runtime\ scriptrunner\script_runner.
py", line 565, in _run_script
    exec(code, module.__dict__)
  File "C:\Users\vaalt\OneDrive\Desktop\medium articles\llm.py", line 129, in <module>
    user_input = get_text()
  File "C:\Users\vaalt\OneDrive\Desktop\medium articles\llm.py", line 50, in get_text
    input_text = st.text_input("You: ", st.session_state['input'], key='input', placeholder =
'Your AI assistant here! Ask me anything...', label_visibility = 'hidden')
  File "C:\Users\vaalt\Anaconda3\lib\site-packages\streamlit\runtime\ metrics_util.py", line 311,
in wrapped_func
    result = non_optional_func(*args, **kwargs)
  File "C:\Users\vaalt\Anaconda3\lib\site-packages\streamlit\elements\ text_widgets.py", line
174, in text_input
    return self._text_input(
```



```
  File "C:\Users\vaalt\Anaconda3\lib\site-packages\streamlit\elements\ text_widgets.py", line
266, in _text_input
    text_input_proto.value = widget_state.value
TypeError: [] has type list, but expected one of: bytes, Unicode
```

Veamos si ChatGPT es capaz de lograr que comprendamos la naturaleza del error. Para ello, basta con darle el texto del error y pedirle que lo explique (véase la figura 6.8).

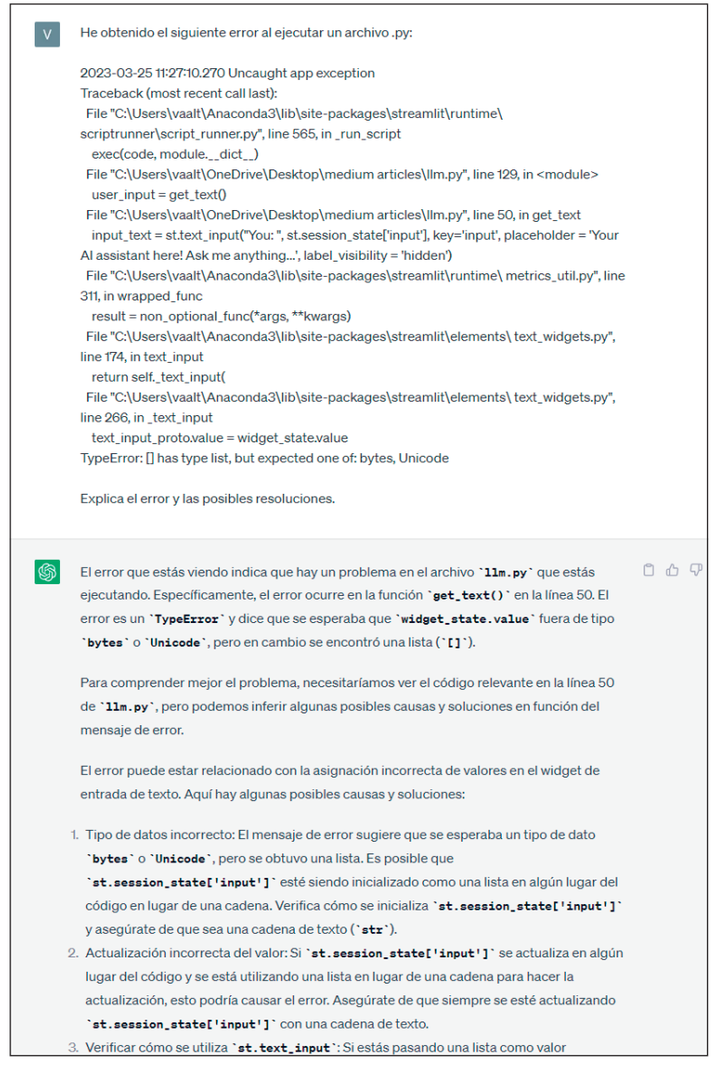



**Figura 6.8.** ChatGPT explicando un error de Python en lenguaje natural.

## Requisitos técnicos

Estos son los requisitos técnicos para este capítulo:

- Una cuenta gratuita de Azure, que se puede crear de manera gratuita aquí: `https://azure.microsoft.com/es-es/free/cognitive-services/`.
- Acceso garantizado a Azure OpenAI dentro de la cuenta de Azure. En el momento de escribir esto, el acceso a este servicio solo estaba garantizado con una solicitud expresa, que se puede llevar a cabo rellenando el formulario que se puede encontrar en `https://aka.ms/oai/access`.

## OpenAI y Microsoft para IA empresarial: presentación de Azure OpenAI

Microsoft tiene una larga historia de inversión en investigación y desarrollo IA, estando especialmente enfocada en la creación de herramientas de IA destinadas a empresas y usuarios individuales, para resolver problemas complejos y mejorar la productividad.

También cuenta con una serie de hitos en el intento de lograr la paridad humana en ámbitos de inteligencia artificial como el reconocimiento de voz (2017), la traducción automática (2018), las conversaciones de preguntas y respuestas (2019), la subtitulación de imágenes (2020) y la comprensión del lenguaje natural (2021).

**DEFINICIÓN**

La paridad humana en inteligencia artificial se refiere al punto en el que un sistema de IA realiza una o varias tareas igual que un ser humano o de una forma indistinguible. Este concepto se utiliza para medir el rendimiento de los sistemas de IA, especialmente en áreas como la comprensión del lenguaje natural, el reconocimiento de voz y de imágenes. Lograr la paridad humana en IA se considera un hito importante, ya que demuestra la capacidad de la inteligencia artificial de igualar de manera efectiva las habilidades humanas en un determinado ámbito.

En las siguientes secciones exploraremos los antecedentes y la historia de investigación de Microsoft en el campo de la inteligencia artificial, para entender plenamente su travesía hacia su asociación con OpenAI y, finalmente, el desarrollo del servicio AOAI.

### Antecedentes de Microsoft en inteligencia artificial

Las primeras investigaciones en el campo de la inteligencia artificial se remontan a finales de los años 90, cuando Microsoft establece su aprendizaje automático (AA) y sus grupos estadísticos aplicados. A partir de aquí, la empresa empezó a investigar



y experimentar con agentes inteligentes y asistentes virtuales. En este caso, el prototipo era Clippy o Clipo, un asistente digital personal para Microsoft Office (véase la figura 9.1).

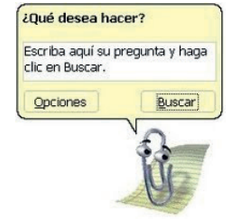



**Figura 9.1.** Clipo, el asistente de Office lanzado en el año 2000.

Clipo fue el precursor de herramientas más sofisticadas como Cortana. Lanzado en 2014, Cortana es un asistente digital que usa procesamiento del lenguaje natural (PLN) y AA para proporcionar a los usuarios asistencia personalizada.

Más tarde, en 2016, como expansión de su Proyecto Oxford, Microsoft lanzó Microsoft Cognitive Services en la nube Azure, una serie de API que ofrecían capacidades de IA a los desarrolladores sin que fuera necesario que tuvieran experiencia previa en AA y ciencia de datos.

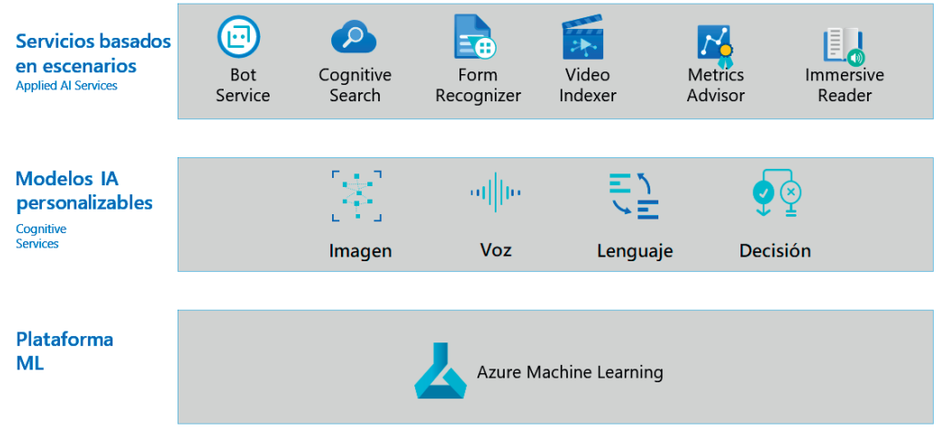



**Figura 9.2.** Servicios de IA de Microsoft Azure.

Con Cognitive Services, por fin la inteligencia artificial podía ser consumida por una amplia variedad de usuarios, desde grandes empresas a desarrolladores individuales. A partir de aquí, asistimos a lo que ahora denominamos la democratización de la inteligencia artificial, es decir, la IA ya no es un privilegio solo para los que tienen profundos conocimientos técnicos y un hardware potente y experto para entrenar modelos. Cognitive Services ha sido desarrollado por las siguientes razones:

Por ejemplo, veamos la siguiente situación. Somos una clínica privada y cada día luchamos por encontrar información entre la gran cantidad de documentación disponible. Para producir un diagnóstico, los doctores tienen que consultar muchos documentos, y todo ello consume mucho tiempo.

Estamos buscando un asistente de IA que pueda ayudarnos en el proceso de investigación. Con este objetivo, usaremos un despliegue de Azure OpenAI denominado `embedding` asociado al modelo `text-embedding-ada-002`.

La idea es la siguiente:

1. Obtener la incrustación del texto disponible con el modelo correspondiente.
2. Obtener la incrustación de la consulta del usuario con el modelo correspondiente.
3. Calcular la distancia entre la consulta incrustada y la base de conocimiento de incrustación.
4. Devolver los fragmentos de texto más similares y usarlos como contexto para el modelo GPT.
5. Usar el modelo GPT para generar una respuesta.

En la figura 10.28 se representan estos pasos.

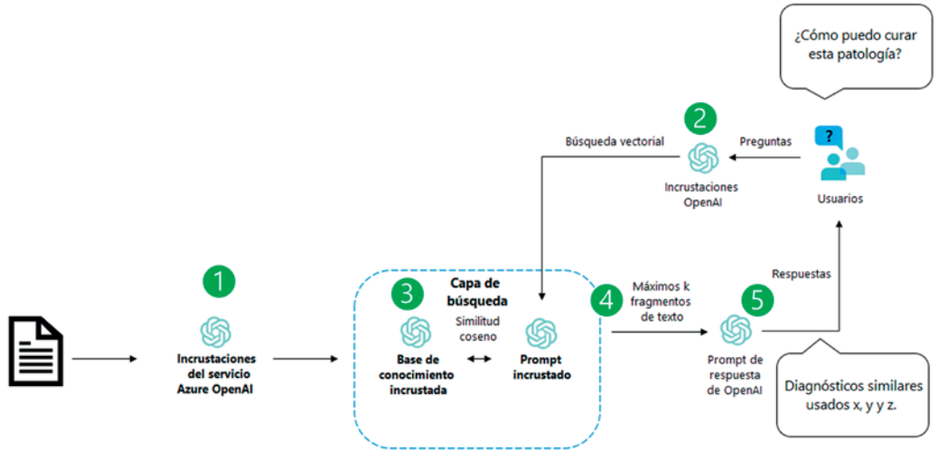



**Figura 10.28.** Una arquitectura de referencia de un proyecto de búsqueda semántica.

Para simular la base de conocimiento usaremos un documento sobre tratamientos alternativos para la ciática (que se puede encontrar en `https://doi.org/10.1136/bmj-2022-070730`).

Para la gestión de la incrustación y de las preguntas y respuestas, emplearemos módulos de LangChain.



## Incrustación de documentos utilizando módulos de LangChain

El primer paso de nuestro proyecto es inicializar un modelo de incrustación, de modo que podamos vectorizar nuestros documentos personalizados. Para ello podemos usar el módulo `OpenAIEmbeddings` de LangChain, que envuelve los modelos de incrustaciones directamente desde Azure OpenAI:

```
from langchain.embeddings import OpenAIEmbeddings from langchain.chat_models import AzureOpenAI
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores.faiss import FAISS
from pypdf import PdfReader
from langchain.document_loaders import PyPDFLoader
  embeddings = OpenAIEmbeddings(document_model_name="text-embedding-ada-002") embeddings.embed_
query('this is a test')
```

La figura 10.29 muestra el resultado.

```
[7]
from langchain.embeddings import OpenAIEmbeddings
#from langchain.chat_models import AzureChatOpenAI

embeddings = OpenAIEmbeddings(document_model_name="text-embedding-ada-002")
embeddings.embed_query('this is a test')
```

```
... Output exceeds the size limit. Open the full output data in a text editor
[-0.012838087975978851,
 -0.007421397138386965,
 -0.017617521807551384,
 -0.02827831171452999,
 -0.0186663419008255,
 0.0173785500228405,
 -0.01821495033800602,
 -0.006950092036277056,
 -0.009937237948179245,
 -0.03858064487576485,
 0.010674066841602325,
 0.02412286028265953,
 -0.013647936284542084,
 0.013189907185733318,
 0.0021125758066773415,
 0.012406611815094948,
 0.02079053409397602,
 0.0007459566695615649,
 0.008397198282182217,
 -0.005350309889763594,
 0.008968074806034565,
 0.014351575635373592,
 -0.014086050912737846,
 0.015055214054882526,
 -0.022211087867617607,
...
 0.014271917752921581,
 0.00510801887139678,
 -0.01090640015900135,
 0.014391403645277023,
 ...]
```

**Figura 10.29.** Ejemplo de incrustaciones de texto.

Como podemos ver, el resultado es un vector numérico, calculado con el modelo de incrustación `text-embedding-ada-002`.

Los modelos de inteligencia artificial generativa (IAG) y de lenguajes de inteligencia artificial son cada vez más conocidos por sus incomparables capacidades. Este volumen ofrece información sobre el funcionamiento interno de los LLM o grandes modelos de lenguaje y una guía para la creación de modelos de lenguaje propios. Comienza con una introducción al campo de la IAG que le permitirá comprender cómo se entrenan estos modelos para generar nuevos datos.

Tendrá además la oportunidad de explorar casos prácticos en los que ChatGPT ha logrado mejorar la productividad y fomentar la creatividad. Aprenderá cómo sacar el máximo partido de sus interacciones con ChatGPT enriqueciendo el diseño de *prompts* y aprovechando las capacidades de aprendizaje *zero-shot, one-shot y few-shot*, es decir, con cero, uno y pocos ejemplos por tarea. Los casos prácticos están agrupados por los ámbitos de técnicos de marketing, investigadores y desarrolladores o científicos de datos, lo que le permitirá aplicar rápidamente a sus propios retos lo aprendido en este libro.

También descubrirá situaciones producidas en empresas utilizando en su beneficio las API de modelos de OpenAI disponibles en la infraestructura de Azure, tanto modelos generativos (como GPT-3) como modelos integrados. En cada situación dispondrá de una implementación integral con Python, utilizando Streamlit como parte visible y LangChain para facilitar la integración de los modelos en sus aplicaciones.

Cuando llegue al final de este libro, habrá obtenido el conocimiento necesario para manejarse perfectamente en el campo de la IAG y empezar a utilizar las API de los modelos de ChatGPT y OpenAI en sus propios proyectos.

Con este volumen logrará:

- Comprender los conceptos de IAG, desde el nivel básico al intermedio.
- Centrarse en la arquitectura GPT para modelos de IAG.
- Maximizar el valor de ChatGPT con un eficaz diseño de *prompts*.
- Explorar aplicaciones y casos prácticos de ChatGPT.
- Utilizar modelos y funciones de OpenAI mediante llamadas a API.
- Construir y desplegar sistemas de IAG con Python.
- Aprovechar la infraestructura de Azure para situaciones de uso en empresas.
- Garantizar una IA responsable y ética en sistemas de IAG.

ISBN 978-84-415-4896-1
2315249
9 788441 548961

ANAYA MULTIMEDIA

www.anayamultimedia.es