¿Qué abarca la clase?

- Definiciones generales
  - Chatbot comercial
  - Vector Store
  - Búsqueda Semántica
  - Prompt Engineering
- Flujo del chatbot
- Laboratorio en VS Code
- Lecciones aprendidas

Preguntas retóricas

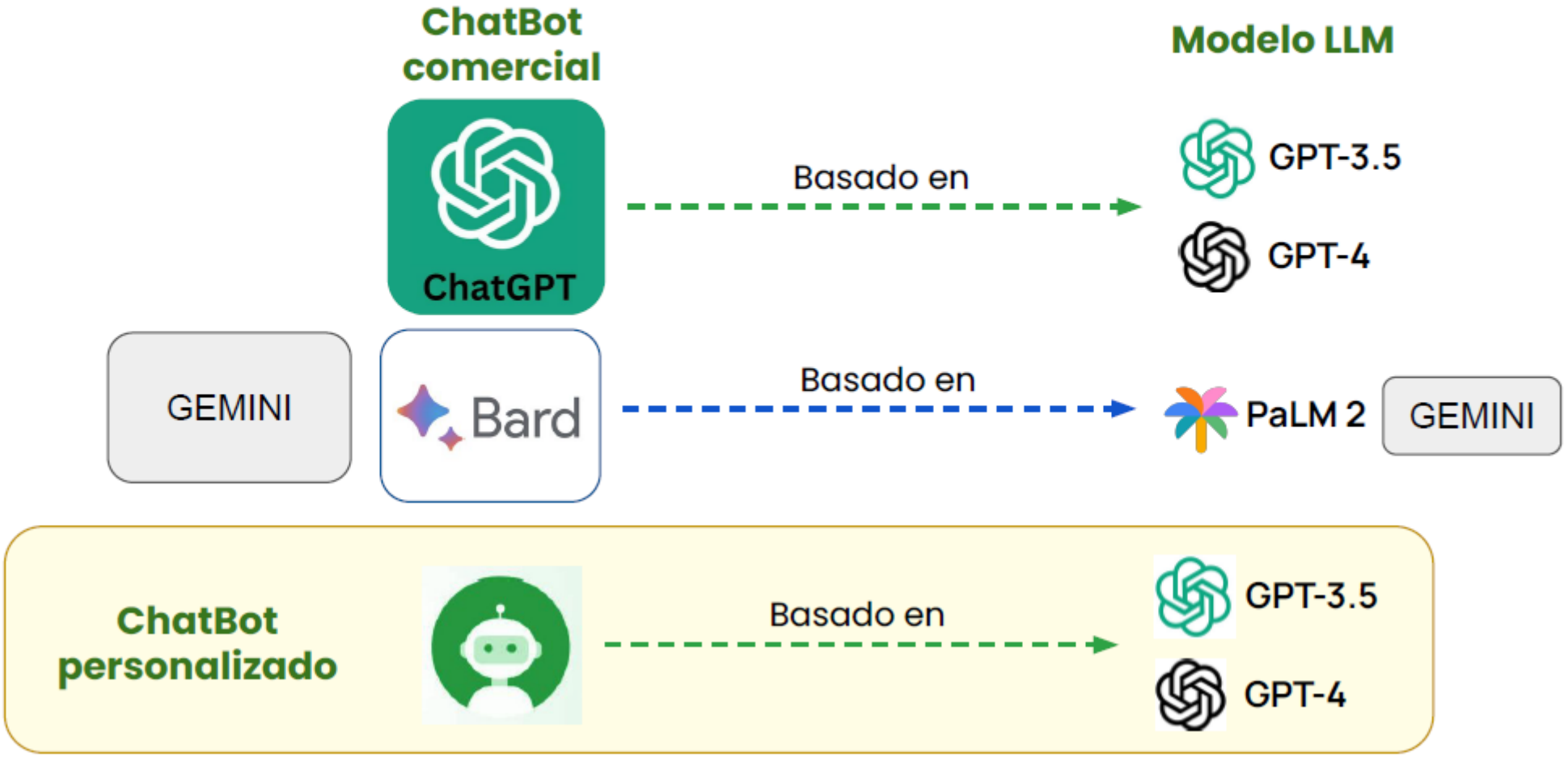
¿Qué tan difícil es crear un chatbot personalizado?

¿Se puede crear un chatbot con 100 líneas de código?

¿Puedo crear mi propio Vector Store?

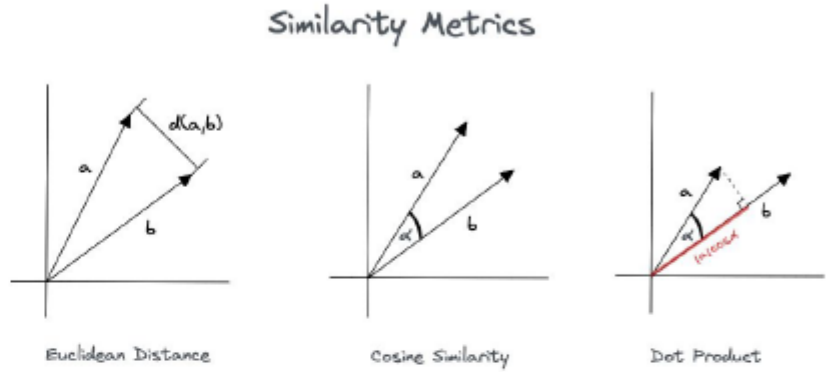
¿Cómo construir un prompt adecuadamente?

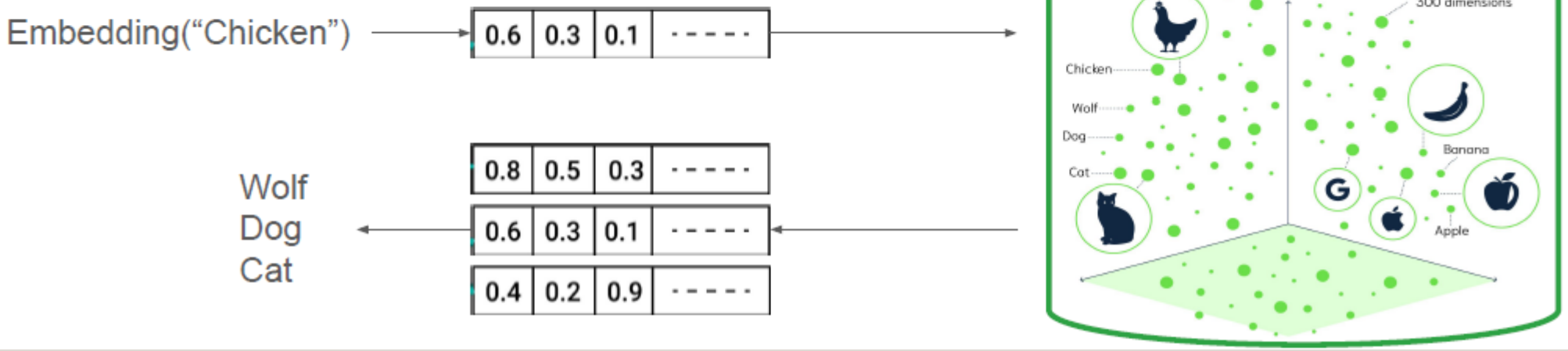
Chatbot comercial



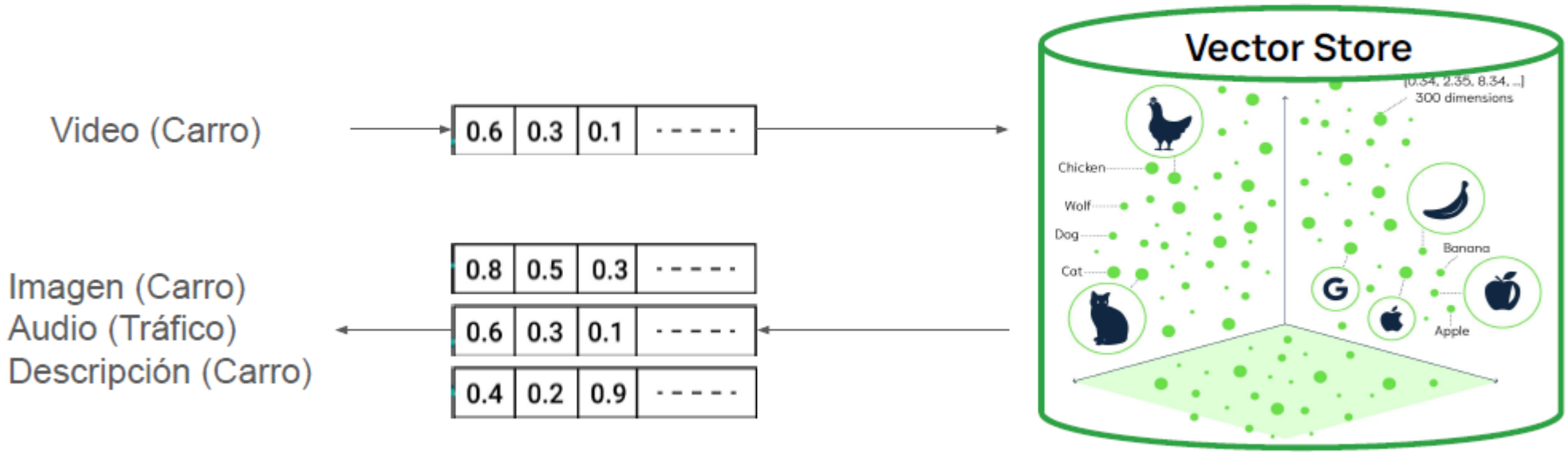
Búsqueda semántica

Cuando **realiza una consulta de búsqueda**, también se convierte en un vector y se **compara con todos los vectores** del Vector Store.
Los vectores con la **mayor métrica de similitud similitud**, es decir, sus vectores más cercanos en el espacio de incrustación, se devuelven como los **resultados más relevantes.**





Búsqueda semántica

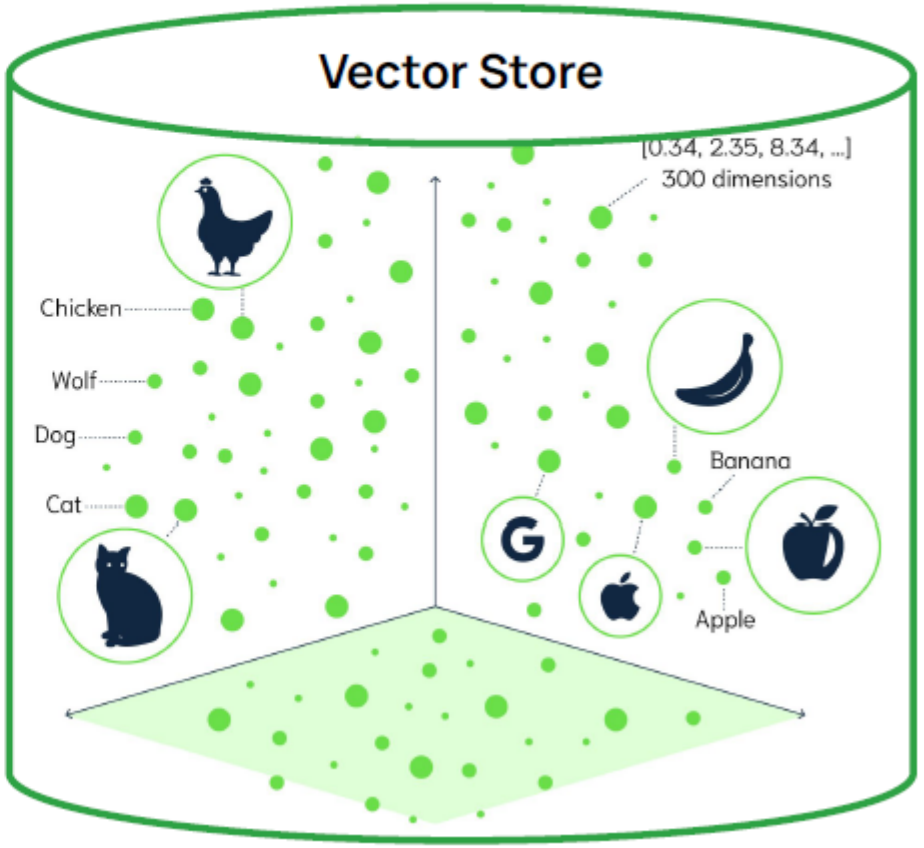


Vector Store

Un Vector Store es un tipo especializado de sistema de **almacenamiento de datos** diseñado para manejar un tipo específico de datos: **vectores de alta dimensión.**

Permite:

- Almacenar y recuperar vectores de manera eficiente
- Realizar búsqueda por similitud
- Administrar metadatos



Prompt Engineering

'Prompt Engineering' es el **arte de optimizar preguntas o elaborar las instrucciones** perfectas para un LLM.

Se trata de diseñar y perfeccionar las indicaciones de texto que enviamos al LLM, para que pueda **comprender y responder las preguntas** de la mejor manera posible.

Contexto

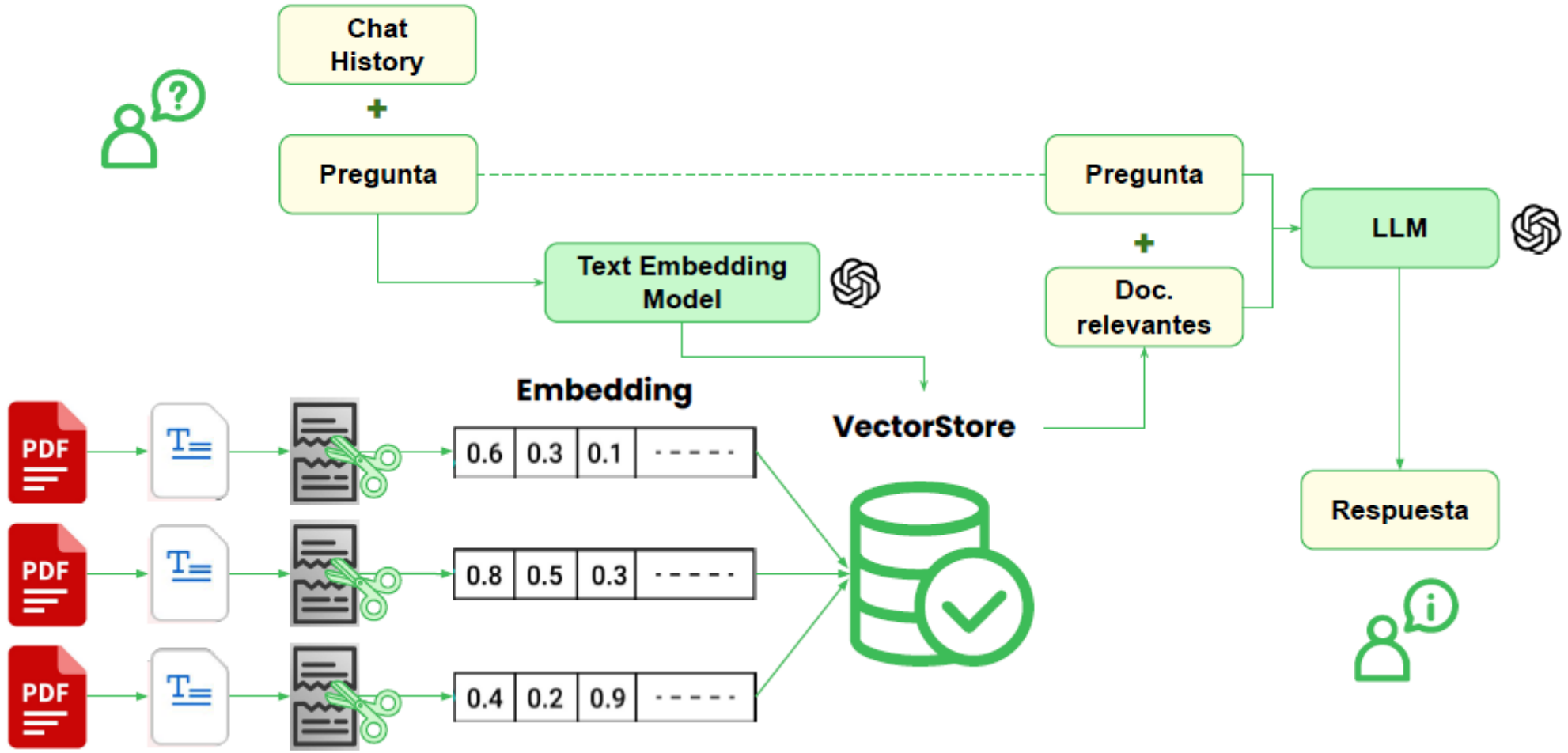
Tarea

Instrucciones

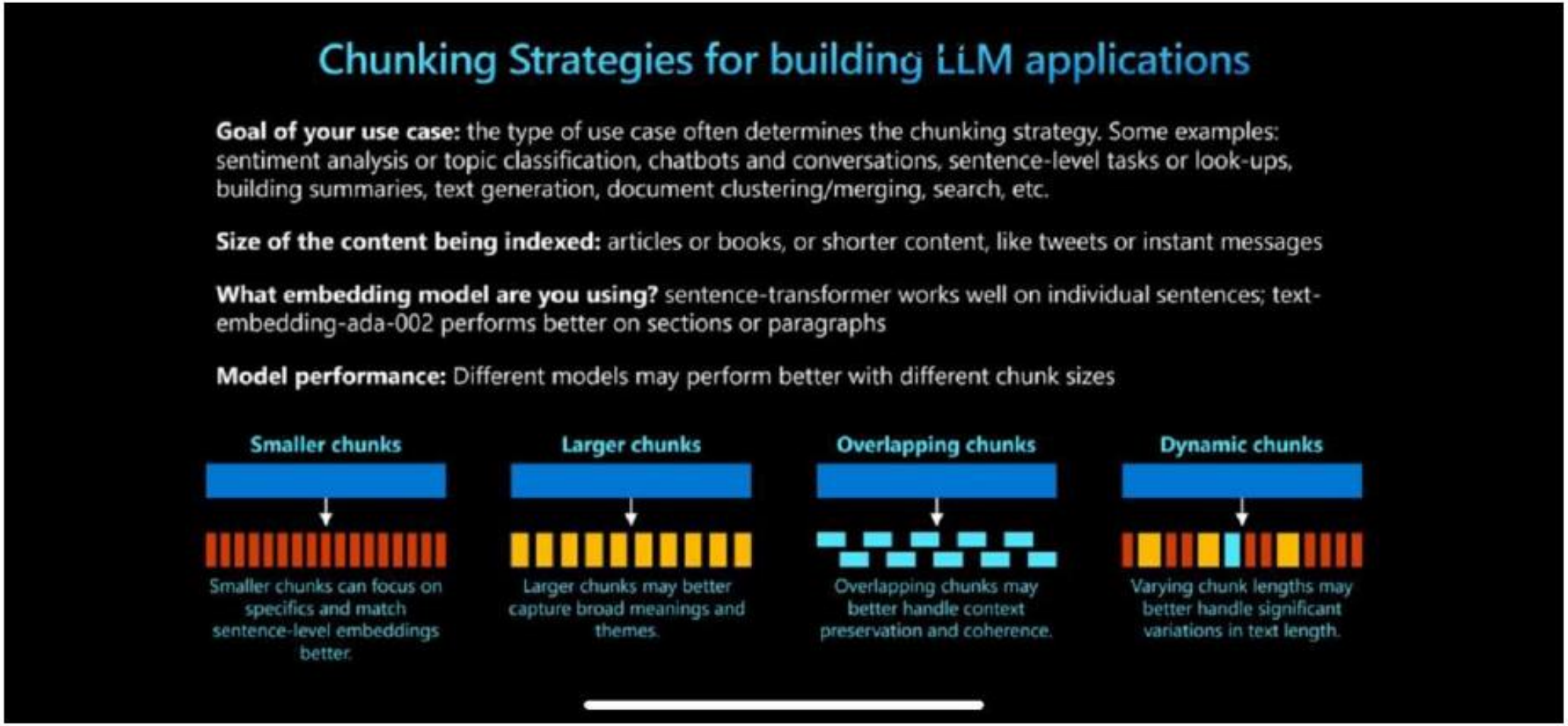
Aclarar y refinar

Eres instructor de yoga y escribes para una revista de bienestar. Escribe un artículo sobre la meditación. El objetivo es educar a los lectores sobre los beneficios de la meditación para aliviar el estrés y mejorar la calidad del sueño. Finalice el artículo enumerando algunos consejos prácticos para principiantes que quieran empezar. Mantenga el tono accesible y amigable. Apunta a un mínimo de 800 palabras. ¿Lo entiendes?

Flujo de chatbot



Generación de chunks



Crear Vector Store como dataframe

Crear una lista de diccionarios a partir de los fragmentos de documento divididos previamente.
Cada diccionario contiene dos claves: 'Chunks', que almacena el contenido del fragmento de texto, y 'Metadata', que almacena los metadatos asociados a dicho fragmento.

Crear un DataFrame de pandas a partir de la lista de diccionarios 'data'.

```
import pandas as pd
data = [{'Chunks': doc.page_content,
         'Metadata': doc.metadata}
        for doc in doc_splits]
df_vector_store = pd.DataFrame(data)
df_vector_store.head()
```

	Chunks	Metadata
0	Building Pipelines and Environments for \nLar...	{'source': 'llm_doc.pdf', 'page': 0}
1	Contents\nIntroduction to LLMOps 1\nWhy LLMOps...	{'source': 'llm_doc.pdf', 'page': 1}
2	1.\nIntroduction to LLMOps\nGenerative AI mode...	{'source': 'llm_doc.pdf', 'page': 2}
3	translation and even creative writing.\nUsing ...	{'source': 'llm_doc.pdf', 'page': 2}
4	with LLMs. Together, these allow data scientis...	{'source': 'llm_doc.pdf', 'page': 2}

Crear Vector Store como dataframe

Abrir el archivo 'credentials.json' en modo de lectura.

Cargar el contenido del archivo JSON en una variable llamada 'config_env'.

Extraer el valor asociado a la clave 'openai_key' del diccionario 'config_env'.

Crear una instancia de la clase OpenAI, proporcionando la clave API extraída del archivo de configuración.

Llamar al método 'embeddings.create' del cliente de OpenAI para crear embeddings del texto.
Se especifica el modelo 'text-embedding-ada-002', el texto de entrada y el formato de codificación.
La función retorna la representación vectorial del texto.

Añadir una nueva columna 'Embedding' al DataFrame 'df_vector_store'.

Guardar el DataFrame 'df_vector_store' en un archivo con formato pickle.

```
file_name = open('credentials.json')
config_env = json.load(file_name)
api_key = config_env["openai_key"]
client = OpenAI(api_key=api_key)

def text_embedding(text=[]):
    embeddings = client.embeddings.create(model="text-embedding-ada-002",
                                          input=text,
                                          encoding_format="float")
    return embeddings.data[0].embedding

df_vector_store["Embedding"] = df_vector_store["Chunks"].apply(lambda x: text_embedding([x]))
df_vector_store["Embedding"] = df_vector_store["Embedding"].apply(np.array)

df_vector_store.to_pickle('df_vector_store.pkl')
df_vector_store.head()
```

	Chunks	Metadata	Embedding
0	Building Pipelines and Environments for \nLar...	{'source': 'llm_doc.pdf', 'page': 0}	[-0.0004000346, -0.012532205, 0.017549334, -0....
1	Contents\nIntroduction to LLMOps 1\nWhy LLMOps...	{'source': 'llm_doc.pdf', 'page': 1}	[0.0031110481, -0.0044141663, -0.0021332903, -...
2	1.\nIntroduction to LLMOps\nGenerative AI mode...	{'source': 'llm_doc.pdf', 'page': 2}	[-0.020828877, -0.008388099, -0.013470776, -0....
3	translation and even creative writing.\nUsing ...	{'source': 'llm_doc.pdf', 'page': 2}	[-0.006481511, -0.005626922, -0.0033462602, -0...
4	with LLMs. Together, these allow data scientis...	{'source': 'llm_doc.pdf', 'page': 2}	[0.0069767754, -0.0102603715, 0.0035248334, -0...

Formulación de pregunta

Definir una cadena de texto que contiene la consulta de interés.

Generar la representación vectorial (embedding) de la consulta de texto utilizando la función 'text_embedding'.

Esta función fue definida previamente y se encarga de convertir el texto de entrada en un vector numérico utilizando un modelo de embeddings específico de OpenAI. La variable 'query_embedding' almacenará el vector resultante.

```
query = '¿Cómo se selecciona un modelo llm?'
query_embedding = text_embedding(query)
query_embedding
```

✓ 0.7s

```
[-0.02419639,
 0.011456056,
 0.0020118116,
 -0.02696989,
 -0.041670803,
 0.028281495,
 -0.026204787,
 0.024128078,
 -0.010930046,
 -0.023649888,
 0.017911615,
 0.025945198,
 -0.012064039,
 0.0051200436,
 -0.02553532,
 -0.0005460754,
 -0.0007160035,
 -0.0006946557,
 0.026723964,
 -0.009092432,
 -0.0063974927,
 -0.012945274,
 0.028008245,
 -0.031314585,
 -0.0047614016,
```

Búsqueda semántica

Definir una función para calcular el producto punto entre dos vectores.

Definir una función para calcular la similitud de coseno entre dos vectores.

Definir una función para obtener los fragmentos de texto más relevantes desde un almacenamiento vectorial dada una consulta.

Convertir el embedding de la consulta en un array de numpy.

Calcular la similitud de coseno para cada vector en el almacenamiento y ordenar los resultados de mayor a menor similitud, seleccionando los índices de los 'n_chunks' más altos.

Seleccionar los fragmentos de texto correspondientes a los índices de mayor similitud.

Devolver una lista con los fragmentos de texto seleccionados.

```python
def get_dot_product(row):
    return np.dot(row, query_vector)

def cosine_similarity(row):
    denominator1 = np.linalg.norm(row)
    denominator2 = np.linalg.norm(query_vector.ravel())
    dot_prod = np.dot(row, query_vector)
    return dot_prod/(denominator1*denominator2)

def get_context_from_query(query, vector_store, n_chunks = 5):
    global query_vector
    query_vector = np.array(query_embedding)
    top_matched = (
        vector_store["Embedding"]
        .apply(cosine_similarity)
        .sort_values(ascending=False)[:n_chunks]
        .index)
    top_matched_df = vector_store[vector_store.index.isin(top_matched)][["Chunks"]]
    return list(top_matched_df['Chunks'])
```