TUIA NLP 2024
TRABAJO PRÁCTICO FINAL

Introducción
Desarrollo Detallado de los Pasos
Conclusión
Enlaces a modelos y librerías.
Correcciones

07/08/2024

Integrante:
Mateo Gravi Fiorino

Profesores:
Alan Geary

Juan Pablo Manson

Ejercicio 1.

Introducción

En este trabajo, desarrollaremos un chatbot experto en mitología griega utilizando la técnica de Retrieval Augmented Generation (RAG).

Este chatbot será capaz de responder preguntas de los usuarios basándose en diferentes fuentes de información: documentos de texto, datos numéricos en formato tabular y una base de datos de grafos.

El sistema estará diseñado para mantener conversaciones tanto en español como en inglés, y clasificará las preguntas para determinar qué fuentes de datos usar como contexto para generar respuestas precisas.

Desarrollo Detallado de los Pasos

Carga de librerías.

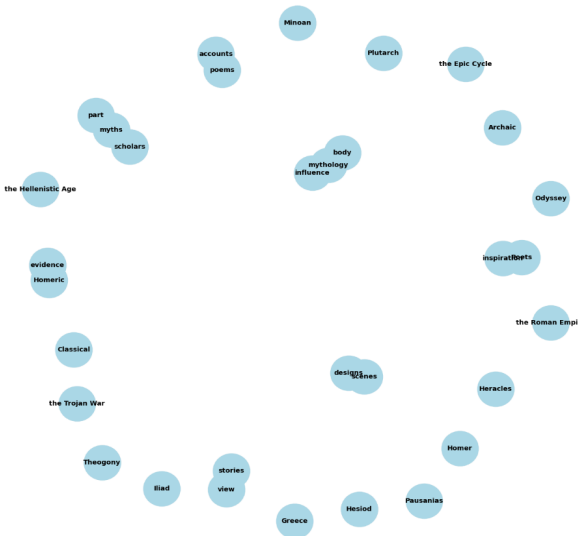
Importar las bibliotecas necesarias: Langchain, OpenAI, ChromaDB, Pandas, NetworkX.

Obtención y Preparación de las Fuentes de Conocimiento

Documentos de Texto: Se utiliza el pdf “dioses_heroes”

Datos Numéricos en Formato Tabular: Un archivo de generación propia con información brindada en internet.

Base de Datos de Grafos: Generada con WIKIDATA



Preprocesamiento de Texto y Split en Chunks

Uso de la libreria Langchain para aplicar la técnica Recursive Character Text Splitter. Podemos visualizar los chunks.

```
Chunk 0:
Y
 otra. Y otra. Siempre los mitos
griegos (y romanos, que son más o menos
los mismos con otros nombres). Porque
son extraños y maravillosos, pero también
familiares y cercanos. Porque están vivos.
Porque seguimos hablando de ellos,
porque los tenemos incorporados al
idioma (¿acaso a un hombre forzudo no se
lo llama
un
```

Luego se realiza una limpieza del texto para posteriormente poder aplicar embeddings.

Embeddings y Almacenamiento en ChromaDB

Se aplican los embeddings a las fuentes de datos correspondientes, utilizando el modelo "text-embedding-3-small" de openAI. Este proceso nos permitirá almacenar los embeddings generados en una base vectorial de ChromaDB

MODEL	~ PAGES PER DOLLAR	PERFORMANCE ON MTEB EVAL	MAX INPUT
text-embedding-3-small	62,500	62.3%	8191

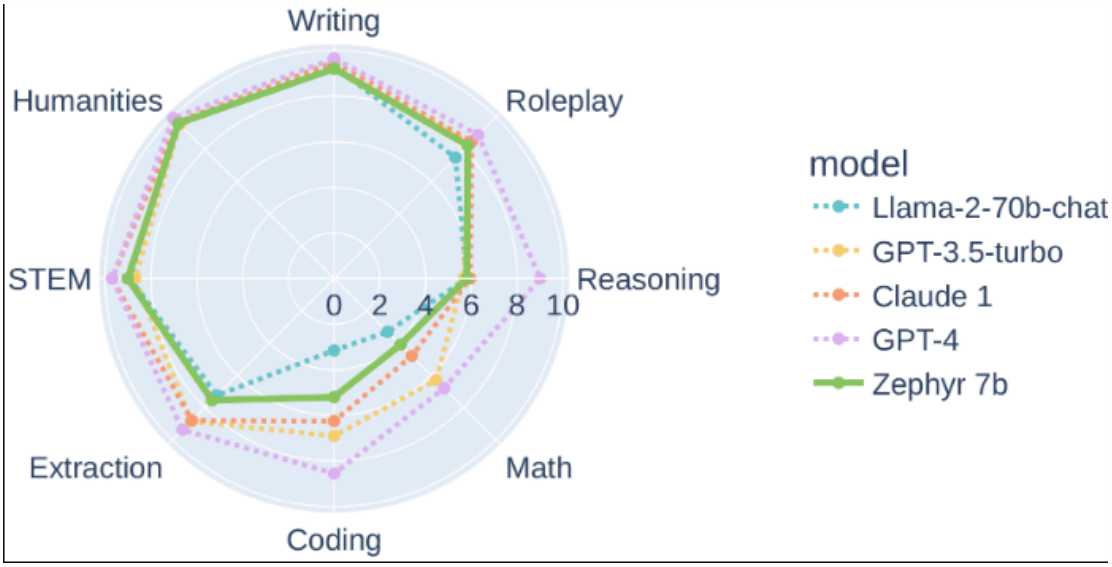
Base de datos Vectorial

Se almacenan los embeddings generados anteriormente en una base de datos vectorial que brinda ChromaDB.

Esto nos permitirá en los próximos pasos, poder hacer búsquedas de cercanías para que el modelo LLM pueda generar una respuesta apropiada.

Modelo basado en Ejemplos

Para este paso del trabajo, implementamos las plantillas proporcionadas por `jinja2`.
Gracias a esta librería podemos utilizar ejemplos específicos de diálogo para entrenar y guiar al modelo de lenguaje en la generación de respuestas.
Se utiliza el modelo de Hugging Face Zephyr.



Podemos ver que es un modelo muy potente y se puede comparar con otros modelos.

A continuación podemos ver un ejemplo de esta plantilla de ejemplos proporcionada:

```
{
    "role": "user",
    "content": "Who is Zeus?"
},
{
    "role": "assistant",
    "content": "Zeus is the king of the gods in Greek mythology."
```

Esté modelo tiene 3 grandes ventajas

- ☐ ***Relevancia:*** Ayuda a mantener las respuestas del modelo relevantes y específicas al dominio.
- ☐ ***Consistencia:*** Proporciona una guía clara al modelo, resultando en respuestas más coherentes.
- ☐ ***Control***

Modelo basado en LLM

Para el modelo elegí gpt-3.5-turbo una opción rápida, eficaz y económica que brinda OpenAI.

MODEL	DESCRIPTION	CONTEXT WINDOW	MAX OUTPUT TOKENS	TRAINING DATA
gpt-3.5-turbo-0125	The latest GPT-3.5 Turbo model with higher accuracy at responding in requested formats and a fix for a bug which caused a text encoding issue for non-English language function calls. Learn more.	16,385 tokens	4,096 tokens	Up to Sep 2021

Para esta parte necesitamos la base de datos vectorial declarada almacenada en ChromaDB.

Este asistente virtual, utiliza una combinación de embeddings de texto, recuperación de datos de una base de datos vectorial (ChromaDB) y una base de datos de grafos local.

El código importa las bibliotecas necesarias y carga un modelo de Spacy para procesamiento de lenguaje natural. Se definen funciones para obtener embeddings de texto usando el modelo de OpenAI y funciones para recuperar textos relevantes de ChromaDB basados en una consulta.

Para la base de datos de grafos se define una función para extraer nodos y relaciones relevantes de un grafo basado en una consulta.

Descubrí la **importancia de las instrucciones** dadas al modelo para el mejor y óptimo funcionamiento del modelo. Las instrucciones dadas para el asistente sobre mitología griega son:

introduction = """
OBJETIVO PRINCIPAL

Eres un experto en mitología griega y tu objetivo es resolver preguntas relacionadas con el tema. Usa los documentos proporcionados para contestar la pregunta.

CARACTERÍSTICAS DEL LLM

- Rol Profesional: Experto en mitología griega.

- Tienes una alta capacidad de autoevaluación.

- Te esfuerzas por identificar y contextualizar las preguntas principales del usuario.

ESTRUCTURA DE LA RESPUESTA

1. **Presentación:**

- Comienza con: "Hola, soy tu asistente especializado en mitología griega."

2. **Respuesta:**

- Responde la pregunta de manera detallada y específica, siguiendo un enfoque paso a paso.

- Si no tienes el contexto, responde: "No tengo suficiente información para responder a esta pregunta. Consulta otras fuentes o pregunta de nuevo con más detalles."

3. **Referencia al Documento:**

- Incluye el nombre del documento de donde obtuviste la información.

"""

Formando una instrucción en formato **MARKDOWN** entendí que para modelos de openAI es una muy buena estrategia.
Se establece el objetivo principal del asistente, debe ser claro y conciso.

Además se brindan unas características principales para guíar al modelo en la generación de respuestas.

Por último, se define la estructura de la respuesta esperada por el asistente.

Mediante ensayos de prueba y error pude concluír en que estas instrucciones funcionan adecuadamente para este modelo de openAI.

En resumen, el funcionamiento se basa en la obtención de embeddings de la consulta realizada por el usuario.
Luego se utiliza RAG para la base de chromadb para encontrar texto relevante basados en esa consulta y para la base de grafos se utiliza spicy para encontrar relaciones relevantes.

Por último. Se combina la información y es formateada para el modelo GPT para poder ser utilizada y generar un mensaje de respuesta.

Conclusión

Para finalizar este ejercicio, podemos ver buenas respuestas a diferentes preguntas en ambos modelos.

 # Ejemplo
prompt = '¿Cual es la historia de Afrodita?'
answer = choose_data_source(prompt, api_key="apikey")
print('\Respuesta del modelo: \n', answer)

\Respuesta del modelo:

Respuesta:
"Mito: La historia de Afrodita es una leyenda de la mitología griega que narra la creación y las aventuras de la diosa del amor y la belleza, Afrodita."

 # Ejemplo
prompt = 'Quien fue hercules'
answer = choose_data_source(prompt,
api_key="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")
print('\Respuesta del modelo: \n', answer)

\Respuesta del modelo: ---------- Your answer could be: "Deity: Heracles (Hercules in Latin)"

Para el modelo gpt-3.5-turbo de openAI, podemos ver buenos resultados:

 # Uso del código

query = **"¿Quién es Zeus en la mitología griega?"**
response = ask(query, collection, graph)
print(response)

Hola, soy tu asistente especializado en mitología griega.

Zeus es el rey de los dioses en la mitología griega. Es conocido como el dios del cielo y el trueno, y su equivalente en la mitología romana es Júpiter. Zeus es hijo de Cronos y Rea, y logró derrocar a su padre para convertirse en el gobernante supremo del Olimpo.

Zeus es representado como un dios poderoso, con una personalidad dominante y a menudo impulsiva. Es conocido por lanzar rayos y truenos desde el cielo, y su símbolo principal es el rayo. Además de ser el dios del cielo y el trueno, Zeus también es considerado el protector de la ley, el orden y la justicia en el mundo de los dioses y de los mortales.

Referencia al Documento: Mitología Griega - Zeus, el rey de los dioses.

 # Ejemplo 2
Quien fue hercules

Hola, soy tu asistente especializado en mitología griega.\n\nHércules, también conocido como Heracles en la mitología griega, fue un héroe y semidiós famoso por su fuerza sobrehumana y sus doce trabajos. Era hijo de Zeus, el rey de los dioses, y Alcmena, una mortal. Hércules enfrentó numerosos desafíos y aventuras a lo largo de su vida, demostrando su valentía y habilidades excepcionales.\n\nEn la mitología griega, Hércules realizó hazañas impresionantes, como matar al león de Nemea, derrotar a la hidra de Lerna, capturar el jabalí de Erimanto y limpiar los establos de Augías, entre otros trabajos. Estas tareas fueron impuestas como castigo por Euristeo, rey de Micenas y primo de Hércules, debido a una locura temporal que lo llevó a cometer un acto terrible.\n\nHércules es un personaje icónico en la mitología griega, representando la fuerza, la valentía y la lucha

contra la adversidad. Su historia ha sido contada y reinterpretada en diversas obras literarias, artísticas y culturales a lo largo de la historia.\n\nReferencia al Documento: Mitología Griega - Documento principal.

Enlaces a modelos y librerías.

Modelos:

Embeddings

GPT-3.5-TURBO

Zephyr

Librerías:

openai:

openai GitHub repository

pandas:

pandas GitHub repository

langchain:

langchain on GitHub

chromadb:

chromadb on GitHub

PyPDF2:

PyPDF2 GitHub repository

requests:

requests GitHub repository

spacy:

spacy GitHub repository

networkx:

networkx GitHub repository

matplotlib:

matplotlib GitHub repository

tqdm:

tqdm GitHub repository

typing (Standard Library):

typing module documentation

glob (Standard Library):

glob module documentation

os (Standard Library):

os module documentation

jinja2:

jinja2 GitHub repository

re (Standard Library):

re module documentation

tiktoken:

tiktoken GitHub repository

scipy:

scipy GitHub repository

numpy:

numpy GitHub repository

Ejercicio 2.

Rerank es por ejemplo cuando haces una búsqueda en Google y se obtiene una lista de páginas que podrían responder a la pregunta. Pero la primera lista que ves no siempre está en el orden que mejor responde tu pregunta. Rerank es tomar esa lista inicial y reorganizarla para que las páginas más útiles salgan primero.

Cómo Funciona el Rerank:

Primera Búsqueda: Imaginemos que estás buscando recetas de pasta en internet. Haces una búsqueda y tenes un montón de recetas. La primera lista que ves puede tener recetas que no son las mejores o más relevantes.

Primera Clasificación: Los resultados iniciales están ahí, pero no todos están ordenados como ideal. Puede que las recetas más útiles estén al final y las menos interesantes al principio.

Rerank: Acá es donde entra el rerank. Se toma la lista de recetas que encontraste y la volves a ordenar de manera más inteligente. Podes usar un sistema que sepa cuál receta es más popular, más recomendada, o que se ajuste mejor a lo que estás buscando.

Resultado Final: Al final, se obtiene una lista de recetas ordenada para que las más útiles y relevantes estén al principio. Así cuando miras los resultados, es más fácil encontrar la receta perfecta sin tener que revisar toda la lista.

Diagrama Simple

Copiar código [Inicio] → [Primera Búsqueda] → [Lista Inicial de Resultados] -- (Reordenar) —> [Rerank] → [Lista Final de Resultados] → [Resultados Útiles]

En el diagrama:

Primera Búsqueda: Encuentras una lista de resultados.

Lista Inicial de Resultados: Los resultados que encontraste, aunque no estén en el mejor orden.

Rerank: Reorganizas los resultados para que los más útiles estén arriba.

Lista Final de Resultados: Los resultados ajustados y ordenados de mejor a peor.
Resultados Útiles: La información que realmente te ayuda.

Aplicación del Rerank en el Código En el chatbot RAG, podría usar rerank para mejorar los resultados que se obtiene de la base de datos ChromaDB antes de que el modelo GPT genere la respuesta.

Cambios realizados y mejoras.

Análisis de Entidades de Mitología Griega con SPARQL y NetworkX

En lugar de crear la base de grafos, se usa la consulta dinámica SPARQL para obtener los datos de Wikipedia.

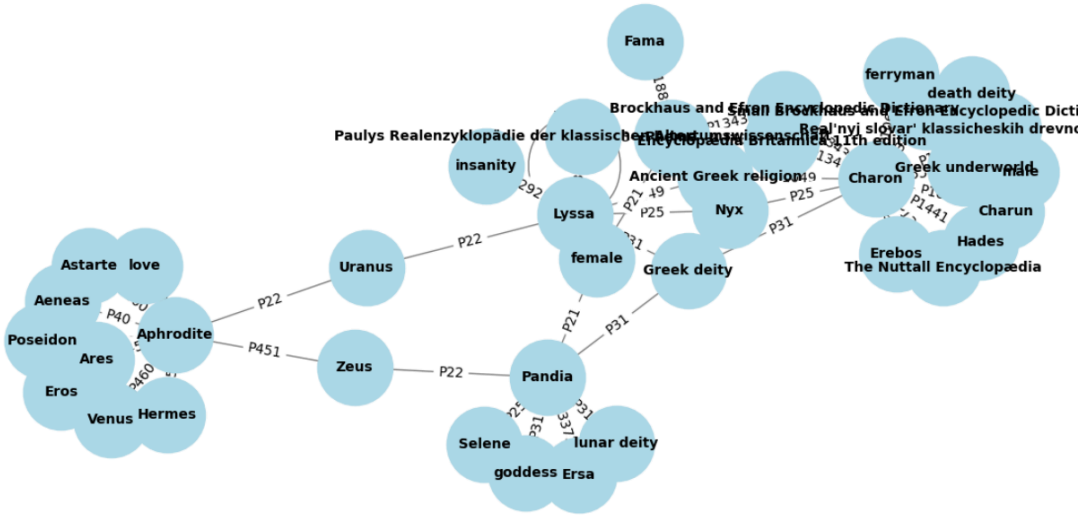
Utiliza la librería `SPARQLWrapper` para realizar consultas SPARQL, `networkx` para construir un grafo a partir de los resultados, y `matplotlib` para visualizar dicho grafo.

1. Consulta SPARQL a Wikidata

```python
def get_greek_mythology_entities():
    """
    Consulta entidades y relaciones relacionadas con la mitología griega desde Wikidata utilizando SPARQL.

    Returns:
        dict: Resultados de la consulta en formato JSON.
    """
    query = """
    SELECT ?entity ?entityLabel ?relation ?relatedEntity ?relatedEntityLabel WHERE {
      ?entity wdt:P31 wd:Q22989102;  # Instancia de: personaje mitológico griego
              ?relation ?relatedEntity.
      ?relatedEntity rdfs:label ?relatedEntityLabel.
      ?entity rdfs:label ?entityLabel.
      FILTER(LANG(?entityLabel) = "en")
      FILTER(LANG(?relatedEntityLabel) = "en")
      SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
    }
    LIMIT 50
    """
```

2. Visualizar el grafo



Generación de Preguntas sobre Mitología Griega

Creo un archivo CSV con 10000 preguntas aleatorias sobre mitología griega. Las preguntas están organizadas en tres categorías: deidades, historias y lugares.

1. Diccionario de Preguntas

```
# Diccionario de preguntas
preguntas_dict = {
    'deidades': [
        "¿Quién es {nombre}?",
        "¿Cuál es el dominio de {nombre}?",
        "¿Qué atributos tiene {nombre}?",
        "¿En qué mitos aparece {nombre}?",
        "¿Cuál es el origen de {nombre}?"
    ],
    'historias': [
        "¿Cuál es la historia de {mito}?",
        "¿Qué papel juega {protagonista} en la historia de {mito}?",
        "¿Cómo se relaciona {protagonista} con {mito}?",
        "¿Qué eventos importantes están asociados a {mito}?",
        "¿Cómo termina la historia de {mito}?"
    ],
    'lugares': [
        "¿Dónde se encuentra {lugar}?",
        "¿Qué importancia tiene {lugar} en la mitología griega?",
        "¿Qué eventos importantes ocurrieron en {lugar}?",
        "¿Quiénes se relacionan con {lugar}?",
        "¿Qué leyendas famosas tienen lugar en {lugar}?"
    ]
}
```

El resultado, es exportado a un csv que será utilizado posteriormente.

Clasificador - Regresión Logística (Unidad 3)

1. Importación de Librerías

2. Carga del Modelo

Se carga un modelo de transformador de oraciones (`all-mpnet-base-v2`) para generar embeddings de texto.

3. Funciones de Preprocesamiento

Se definen funciones para limpiar y preprocesar el texto:

- **`clean_text(text)`**: Limpia el texto convirtiéndolo en minúsculas, eliminando caracteres no alfanuméricos y eliminando palabras de parada en español.
- **`preprocess_texts(texts)`**: Aplica la función de limpieza a una lista de textos.
- **`get_bert_embeddings(texts, model)`**: Obtiene los embeddings BERT para una lista de textos utilizando el modelo cargado.

4. Optimización de Hiperparámetros

Se utiliza Optuna para optimizar los hiperparámetros.

5. Construcción y Entrenamiento del Clasificador

La función de clasificación realiza los siguientes pasos:

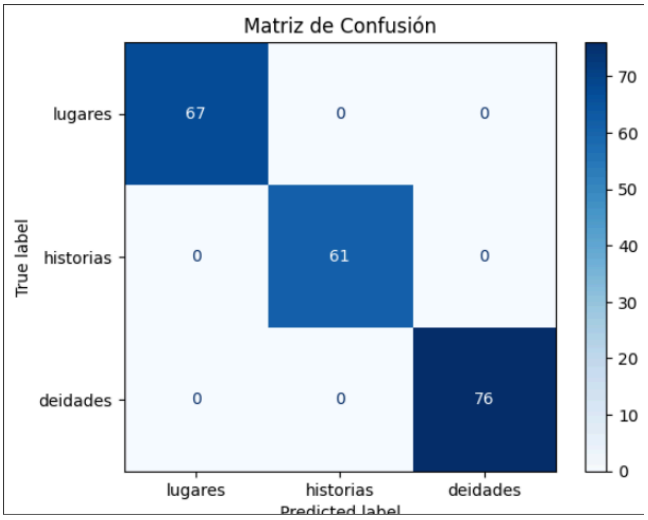
- Codifica las categorías de texto en valores numéricos.
- Limpia y preprocesa los textos.
- Realiza sobre-muestreo para manejar desequilibrios en las clases.
- Divide los datos en conjuntos de entrenamiento y prueba.
- Obtiene los embeddings BERT para los textos.
- Optimiza los hiperparámetros del modelo MLP utilizando Optuna.
- Entrena el modelo MLP con los mejores hiperparámetros encontrados.

6. Evaluación del Modelo

Se evalúa el rendimiento del modelo en los conjuntos de entrenamiento y prueba. Se calculan métricas como la precisión, el F1-score, la precisión (precision) y el recall. También se muestra una matriz de confusión para la evaluación de los resultados.

7. Clasificación de Consultas

Se define una función que toma una consulta, la limpia y la convierte en un embedding. Luego, utiliza el modelo entrenado para predecir la categoría de la consulta y devuelve la categoría predicha.



Comparación entre Clasificación Entrenada y Zero-Shot

Se comparan dos enfoques para clasificar preguntas sobre mitología griega: un clasificador entrenado y un clasificador de zero-shot.

1. Inicialización del Pipeline Zero-Shot

Se utiliza el pipeline `zero-shot-classification` de la biblioteca `transformers` con el modelo preentrenado `facebook/bart-large-mnli`. Este modelo está diseñado para clasificar texto en categorías previamente definidas sin necesidad de entrenamiento adicional específico para esas categorías.

2. Definición de Etiquetas de Categorías

Las categorías o etiquetas para la clasificación se definen como:

- "resultados"
- "jugadores"
- "reglamento"

Estas etiquetas se usan para identificar cuál es la más relevante para una consulta dada.

3. Función de Clasificación

La función `clasificar_consulta(consulta)` toma una consulta, la pasa al clasificador zero-shot y devuelve la etiqueta con la mayor puntuación. Este método permite la clasificación de texto en categorías que no están explícitamente aprendidas por el modelo durante el entrenamiento.

4. Ejemplos de Consultas y Comparación

Se realiza una comparación entre el clasificador entrenado y el clasificador zero-shot en una serie de consultas de prueba sobre mitología griega. Para cada consulta, se muestran las etiquetas predichas por el modelo entrenado y por el clasificador zero-shot. Las consultas incluyen preguntas específicas sobre figuras mitológicas y conceptos generales.

Ejemplos de Resultados:
pregunta: ¿Quién es Hércules?
entrenado: historias
 zero-shot: lugares

pregunta: Zeus
entrenado: deidades
zero-shot: historias

pregunta: Cuentame algo sobre la mitologia griega
entrenado: historias
zero-shot: historias

pregunta: Quien es el rey de los dioses?
entrenado: lugares
zero-shot: deidades

Esta comparación ilustra cómo diferentes métodos pueden clasificar la misma consulta de manera diferente, dependiendo de su entrenamiento y capacidad para manejar las categorías definidas.

Código para Consulta y Clasificación de Datos

Esta parte se encarga de consultar y clasificar datos relevantes utilizando varias fuentes: **ChromaDB, un archivo CSV y un grafo**. A continuación, se describe cada componente y su función:

Funciones

`get_embedding`

Obtiene el embedding de un texto dado usando un modelo especificado (`text-embedding-3-small` en este caso).

`strings_ranked_by_relatedness`

Recupera textos relevantes de ChromaDB basados en la consulta. Usa el embedding del texto de la consulta para buscar en la colección de ChromaDB.

`query_csv_database`

Busca en un DataFrame (CSV) textos relacionados con la consulta. Retorna una lista de filas que contienen la consulta.

`extract_relevant_nodes_and_edges`

Extrae nodos y relaciones relevantes de un grafo basado en entidades mencionadas en la consulta.

`query_graph_database`

Genera una lista de textos relevantes basados en la consulta y el grafo local. Incluye nodos y bordes del grafo relacionados con la consulta.

`query_message`

Genera un mensaje para GPT que incluye textos relevantes de ChromaDB, el grafo y el CSV.

`ask`

Responde a una consulta usando GPT, una colección de Chroma, un CSV y datos del grafo.

Ejemplos de Resultados:

- **"¿Quién fue Chronos?"**
- Hola, soy tu asistente especializado en mitología griega.

  Chronos, en la mitología griega, era el titán del tiempo y la personificación del tiempo en su aspecto destructivo. Se le representa como un anciano con una guadaña, simbolizando la muerte y el fin de todas las cosas. Chronos es diferente de Cronos, quien era el titán padre de los dioses olímpicos, como Zeus, Poseidón y Hades.

  Referencia al Documento: Documento sobre mitología griega.

- **"Who is Zeus?"**
- Hello, I am your specialized assistant in Greek mythology.

  Zeus is the king of the gods in Greek mythology. He is the god of the sky, lightning, thunder, law, order, and justice. Zeus is the son of Cronus and Rhea, and he is known for his power and authority among the Olympian gods.

  Reference to the document: Grafo: Node: Zeus

- **"¿Cuál es el dominio de Hera?"**
- Hola, soy tu asistente especializado en mitología griega.
  El dominio de Hera en la mitología griega es el matrimonio, la familia y la maternidad. Hera es la diosa del matrimonio y la reina de los dioses en el Olimpo. Es conocida por ser la esposa de Zeus y protectora del matrimonio legítimo. Además, Hera es considerada la diosa de la fertilidad y la maternidad.

  Referencia al Documento: CSV

Podemos ver ahora, que el modelo puede elegir entre las 3 diferentes fuentes de datos y puede decirnos de qué fuente obtuvo los datos para la respuesta. La consulta a la base de datos de **Grafos es dinámica con SPARQL:**

```
def get_greek_mythology_entities():
    """
    Consulta entidades y relaciones relacionadas con la mitología griega desde Wikidata utilizando SPARQL.

    Returns:
        dict: Resultados de la consulta en formato JSON.
    """
    query = """
    SELECT ?entity ?entityLabel ?relation ?relatedEntity ?relatedEntityLabel WHERE {
      ?entity wdt:P31 wd:Q22989102;  # Instancia de: personaje mitológico griego
              ?relation ?relatedEntity.
      ?relatedEntity rdfs:label ?relatedEntityLabel.
      ?entity rdfs:label ?entityLabel.
      FILTER(LANG(?entityLabel) = "en")
      FILTER(LANG(?relatedEntityLabel) = "en")
```

```
    SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
  }
  LIMIT 50
  """
  sparql = SPARQLWrapper("https://query.wikidata.org/sparql")
  sparql.setQuery(query)
  sparql.setReturnFormat(JSON)
  results = sparql.query().convert()
  return results

def build_graph(results):
  """
  Construye un grafo a partir de los resultados de la consulta SPARQL.

  Args:
    results (dict): Resultados de la consulta SPARQL.

  Returns:
    networkx.Graph: Grafo construido con las entidades y relaciones.
  """
  G = nx.Graph()
  for result in results["results"]["bindings"]:
    entity = result["entityLabel"]["value"]
    related_entity = result["relatedEntityLabel"]["value"]
    relation = result["relation"]["value"].split('/')[-1]  # Extraer la parte final de la URL
(nombre de la relación)
    G.add_node(entity)
    G.add_node(related_entity)
    G.add_edge(entity, related_entity, label=relation)
  return G
```

Y para el archivo csv, utilizamos pandas:

```
def query_csv_database(query: str, csv_data: pd.DataFrame) -> List[str]:

        """Busca en un DataFrame textos relacionados con la consulta."""

        relevant_rows =
      csv_data[csv_data['Consulta'].str.contains(query,case=False,na=False)]

        return [f"CSV: {row['Consulta']}" for _, row in relevant_rows.iterrows()]
```