UNIVERSIDAD NACIONAL DE ROSARIO
FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

Trabajo Práctico N° 2

PROCESAMIENTO DEL LENGUAJE NATURAL

INFORME

RAG – Retrieval Augmented Generation
Agentes – Sistemas Multiagentes

Carrera: Tecnicatura Universitaria en Inteligencia Artificial

Alumno: *Mussi, Miguel*

Ciclo: *2023*

TABLA DE CONTENIDOS

PARTE I: CONSIGNAS

Pautas Generales

- El trabajo deberá ser realizado individualmente.
- Deberá informar cuál es la url del repositorio con el que van a trabajar y las definiciones de en qué problematicas quisieran solucionar con un sistema multiagente en el siguiente formulario: https://docs.google.com/forms/d/e/1FAIpQLSdOZNGOzQ1gbf43caA4ygAbLx5tm5bU-s8RdfdftOzd_aXzhA/viewform?usp=pp_url
- Se debe entregar un informe en el cual se incluya las justificaciones y un vínculo a los archivos que permitan reproducir el proyecto.
- Temas deseables a cubrir en el tp
    - Recuperación de datos de bases de datos de grafos
    - Extracción de conocimiento de texto y posterior inserción en una base de datos de grafos
    - Agentes (estará cubierto en el Ejercicio 2)

Ejercicio 1 – RAG

Crear un chatbot experto en un tema a elección, usando la técnica RAG (Retrieval Augmented Generation). Como fuentes de conocimiento se utilizarán al menos las siguientes fuentes:

- Documentos de texto
- Datos numéricos en formato tabular (por ej., Dataframes, CSV, sqlite, etc.)
- Base de datos de grafos (Online o local)

El sistema debe poder llevar a cabo una conversación en lenguaje español. El usuario podrá hacer preguntas, que el chatbot intentará responder a partir de datos de algunas de sus fuentes. El asistente debe poder clasificar las preguntas, para saber qué fuentes de datos utilizar como contexto para generar una respuesta.

Requerimientos generales

- Realizar todo el proyecto en un entorno Google Colab
- El conjunto de datos debe tener al menos 100 páginas de texto y un mínimo de 3 documentos.
- Realizar split de textos usando Langchain (RecursiveTextSearch, u otros métodos disponibles). Limpiar el texto según sea conveniente.
- Realizar los embeddings que permitan vectorizar el texto y almacenarlo en una base de datos ChromaDB
- Los modelos de embeddings y LLM para generación de texto son a elección

Ejercicio 2 – Agentes

Realice una investigación respecto al estado del arte de las aplicaciones actuales de agentes inteligentes usando modelos LLM libres.

Plantee una problemática a solucionar con un sistema multiagente. Defina cada uno de los agentes involucrados en la tarea.

Realice un informe con los resultados de la investigación y con el esquema del sistema multiagente, no olvide incluir fuentes de información.

Opcional: Resolución con código de dicho escenario.

PARTE II: RESOLUCIÓN EJERCICIO 1

Elección del tema

El chatbot será especialista en temas relacionados al **Marco Legal de la Educación Argentina**, en particular en la provincia de Buenos Aires.

Será denominado "**EduArDo**" como una forma de darle un nombre propio que relacione las ideas de "**Edu**cación", "**Ar**gentina" y "Hacer" ('***Do***' *en inglés*).

[Imagen generada con Copilot Designer con tecnología DALL.E 3]

Para su contexto, se lo proveerá con los siguientes archivos pdf:

- Marco Legal de la Educación en la Argentina *(21 páginas)*
- Reglamento General de Escuelas Públicas de la Provincia de Buenos Aires *(88 páginas)*
- Ley de Educación Nacional Nº 26206 *(30 páginas)*
- Estatuto Docente de la Provincia de Buenos Aires *(271 páginas)*

Además, se cargará un dataframe con el Padrón Oficial de Establecimientos Educativos *(datos.gob.ar)* para consultas sobre datos específicos sobre las instituciones.

Estructura del código

Bases de datos

El chatbot será capaz de identificar el objetivo de cada consulta realizada por el usuario y redireccionará la búsqueda de datos para su contexto hacia tres tipos distintos de fuentes de datos.

- **Base de datos vectorial:** contiene información correspondiente al marco legal de la educación argentina y en particular de la provincia de Buenos Aires. Corresponde a leyes, normas y regulaciones que afecten a las instituciones educativas o a cualquiera de los agentes y personas que participan en ellas.
  Para obtener esta información, se le aportan una serie de archivos pdf de fuentes oficiales que corresponden al Marco Legal de la Educación Argentina, la Ley de Educación Nacional, el Reglamento General de Escuelas de Buenos Aires y el Estatuto Docente de la Provincia de Buenos Aires.
- **Base de datos de grafos**: contiene información general de algunas personalidades destacadas de la educación argentina. Esta información fue extraída de la web Wikidata y se eligieron sólo algunas personalidades a modo de ejemplo (Domingo Faustino Sarmiento, Carlos Pellegrini, Julio Argentino Roca y Arturo Jauretche).
  Entre los datos utilizados para almacenar en la base de datos de grafos se encuentran nombre, nacionalidad, fechas de nacimiento y fallecimiento, ocupaciones, etc.
- **Base de datos tabular**: corresponde a un DataFrame creado con Pandas a partir de un dataset oficial denominado “Padrón Oficial de Establecimientos Educativos” y contiene toda la información general y de contacto de cada institución educativa del país.

Por lo que, a partir de la información suministrada como contexto al modelo de lenguaje, el usuario podrá consultar aspectos formales de la reglamentación y legislación educativa, datos específicos sobre personalidades destacadas de la historia de la educación argentina, o bien, solicitar información de contacto de cualquier institución educativa del país.

Base de datos vectorial

Lectura. En primer lugar, se extraen de los pdfs los textos con **pdfplumber**.

```
[ ]
# Importa los archivos de una carpeta específica de drive
drive_path = "/content/drive/MyDrive/UNR/4 - Proc Lenguaje Natural (IA42)/TP2/EduArDo/Data"
```

```
import os
import pdfplumber as pp

# Nombre de los textos cargados en Drive
archivos = ['Marco_Legal_Educacion_Argentina',
            'Ley_Educacion_Nacional_26206',
            'Reglamento_Gral_Escuelas_BsAs',
            'Estatuto_Docente_BsAs']

# Aca se almacenaran los textos de los pdf
textos = {}

# Extracción de los pdf:
for archivo in archivos:
    pdf_path = os.path.join(drive_path, f'{archivo}.pdf')

    # Verificamos si el archivo existe antes de intentar abrirlo
    if os.path.exists(pdf_path):
        # Abrimos el pdf
        with pp.open(pdf_path) as pdf:
            texto = ""
            # Por cada página
            for pagina in pdf.pages:
                # Extraemos el texto
                texto += pagina.extract_text() + "\n"
            # Una vez que extraemos cada pagina guardamos el texto entero con la clave del nombre correspondi
            textos[archivo] = texto
    else:
        print(f"El archivo {archivo}.pdf no se encuentra en la ruta especificada: {pdf_path}")

```

Limpieza. En este apartado, se destaca que algunos documentos tienen numeración y/o texto en los pies de páginas o caracteres especiales. Por este motivo se realizaron varias pruebas para 'limpiar' los textos y eliminar estos junto a las stopwords. Algunas de ellas fueron satisfactorias (por ejemplo, la eliminación del texto ubicado en el pie de página o la numeración). Otras pruebas, en cambio (como la eliminación de stopwors), dejan el texto demasiado falto de coherencia y, al tratarse de reglamentaciones donde la interpretación debe ser literal y rigurosa, se creyó conveniente que debían quedar como estaban, aún en perjuicio de la cantidad de texto almacenado y la eficiencia en las búsquedas.

```
for pagina in pdf.pages:
    # Extraemos el texto
    texto_pagina = pagina.extract_text()

    # Eliminamos números de página y pie de página usando expresiones regulares
    texto_pagina = re.sub(r'\b\d+\b', '', texto_pagina)  # Eliminar números
    texto_pagina = re.sub(r'Pie de Página.*', '', texto_pagina, flags=re.IGNORECASE)  # Eliminar pie de página
```

```
textos['Marco_Legal_Educacion_Argentina']
```

```
'Ministerio de Educación del Gobierno\nde la Ciudad de Buenos Aires\nDirección General de Planeamiento Educativo
\nProyecto de Recopilación y Reformulación de Normativa Educativa\nMARCO LEGAL DE LA EDUCACIÓN EN LA ARGENTINA\n
Informe realizado por: Verónica Consoli\n\nCoordinación: Susana Xifra\nIntegrantes: Gisela Rotstein, Verónica Co
nsoli, Susana Lungarete, Mariela Arroyo,\nValentina Tenti, Tatiana Corvalán, Patricia Pérez.\n\nMARCO LEGAL DE L
A EDUCACIÓN EN LA ARGENTINA1.\nEn el presente documento se presentan leyes y algunas Resoluciones del CFE que\ni
ntegran del marco legal del Sistema educativo argentino y de la Ciudad de Buenos Aires\nactualizadas a la fecha
y con una breve descripción, a excepción del nuevo Código Civil\ny Comercial sobre el que se describen algunas c
uestiones particulares que se consideran\nde interés.\nA modo de introducción para la lectura del documento se d
estaca para asegurar una\nintegración normativa real e impedir interpretaciones sesgadas que la interpret…'
```

Se adjunta una muestra de los resultados obtenidos luego de varias etapas de limpieza.

```
1   textos_cortados['Estatuto_Docente_BsAs']
```

```
['ESTATUTO DEL DOCENTE\nPROVINCIA DE BUENOS AIRES\nÍNDICE\nCAPITULO I\nDISPOSICIONES GENERALES\n1. Ambito de
aplicación\n2. Docencia. Definición\n3. Adquisición de Derechos y Obligaciones\n4. Situación de revista :
Clases\n5. Pérdida\nCAPITULO II\nDE LAS OBLIGACIONES Y DERECHOS DEL PERSONAL DOCENTE\n6. Deberes\n7. Derechos
del docente titular\n8. Derechos de docentes suplentes y provisionales\n9. Asociaciones gremiales\nCAPITULO
III\nDE LA CLASIFICACION DE LOS ESTABLECIMIENTOS DE ENSEÑANZA\n10. Criterios de clasificación\nCAPITULO
IV\nDEL ESCALAFON\n11. Escalafón General (grados jerárquicos)\n12. Adecuación por dirección docente\n13.
Ingreso. Tipos de prestación\n14. Previsión\n15. Discapacitados, (aplicación de la Ley de)\n16. Denominación
interpretación en normas posteriores.\nCAPITULO V\nDE LA ESTABILIDAD\n17. Titulares\n18. Perdida del derecho.
Casos\n19. Traslado\n20. Disminución de jerarquía\nCAPITULO VI\nDE LA DISPONIBILIDAD\n21. Casos\n22.
Duración\n23. Reubicación transitoria\n24. Destino definitivo\n25. Requisitos',
 'CAPITULO V\nDE LA ESTABILIDAD\n17. Titulares\n18. Perdida del derecho. Casos\n19. Traslado\n20. Disminución
de jerarquía\nCAPITULO VI\nDE LA DISPONIBILIDAD\n21. Casos\n22. Duración\n23. Reubicación transitoria\n24.
Destino definitivo\n25. Requisitos\n26. Principio de unidad familiar\n27. Cese. Cómputo de períodos no
trabajados\nCAPITULO VII\nDE LAS INCOMPATIBILIDADES\n28. Acumulación de cargos y horas\n29. Superposición
horaria. Distancia\n30. Falta grave\nCAPITULO VIII\nDE LAS REMUNERACIONES\n31. Integración\n32. Sistema de
índices\n33. Bonificación por antigüedad. Cálculo\nDGCyE / SSE / Dirección Legal y Técnica / 1\n34.
Acumulación de servicios. Reajuste\n35. Servicio activo. Continuidad\n36. Medios desfavorables.
Bonificación\n37. Función especializada. Ídem\n38. Funciones transitorias. Retribución\n39. Retribución
especial\nCAPITULO IX\nDE LOS TRIBUNALES DE CLASIFICACION\n40. Tribunales de clasificación centrales y
descentralizados\n41. Integración\n42. Representantes docentes. Duración del mandato',
```

```
1   textos_cortados['Estatuto_Docente_BsAs']
```

```
['ESTATUTO DOCENTE PROVINCIA BUENOS AIRES ÍNDICE CAPITULO I DISPOSICIONES GENERALES . Ambito aplicación .
Docencia . Definición . Adquisición Derechos Obligaciones . Situación revista : Clases . Pérdida CAPITULO II
OBLIGACIONES DERECHOS PERSONAL DOCENTE . Deberes . Derechos docente titular . Derechos docentes suplentes
provisionales . Asociaciones gremiales CAPITULO III CLASIFICACION ESTABLECIMIENTOS ENSEÑANZA . Criterios
clasificación CAPITULO IV ESCALAFON . Escalafón General ( grados jerárquicos ) . Adecuación dirección docente
. Ingreso . Tipos prestación . Previsión . Discapacitados , ( aplicación Ley ) . Denominación interpretación
normas posteriores . CAPITULO V ESTABILIDAD . Titulares . Perdida derecho . Casos . Traslado . Disminución
jerarquía CAPITULO VI DISPONIBILIDAD . Casos . Duración . Reubicación transitoria . Destino definitivo .
Requisitos . Principio unidad familiar . Cese . Cómputo períodos trabajados CAPITULO VII INCOMPATIBILIDADES .
Acumulación cargos horas .',
 'jerarquía CAPITULO VI DISPONIBILIDAD . Casos . Duración . Reubicación transitoria . Destino definitivo .
Requisitos . Principio unidad familiar . Cese . Cómputo períodos trabajados CAPITULO VII INCOMPATIBILIDADES .
Acumulación cargos horas . Superposición horaria . Distancia . Falta grave CAPITULO VIII REMUNERACIONES .
Integración . Sistema índices . Bonificación antigüedad . Cálculo DGCyE / SSE / Dirección Legal Técnica / .
Acumulación servicios . Reajuste . Servicio activo . Continuidad . Medios desfavorables . Bonificación .
Función especializada . Ídem . Funciones transitorias . Retribución . Retribución especial CAPITULO IX
TRIBUNALES CLASIFICACION . Tribunales clasificación centrales descentralizados . Integración . Representantes
docentes . Duración mandato . Elección . Requisitos . Integrantes . Incompatibilidades . Licencia goce haberes
. Recusación . Excusación . . Funciones . Listas . Publicidad . Impugnación dictámenes . Recursos CAPITULO X
CLASIFICACION PERSONAL DOCENTE',
```

Chunks. El siguiente paso es partir cada uno de los textos en chunks.

```
[ ]  1  from langchain.text_splitter import RecursiveCharacterTextSplitter
     2
     3  # Creamos el text splitter
     4  text_splitter = RecursiveCharacterTextSplitter(
     5      # Tamaño del chunk
     6      chunk_size=1000,
     7      # Solapamiento entre chunks
     8      chunk_overlap=250
     9      )
```

La elección de los parámetros **chunk_size** y **chunk_overlap** responde a un análisis de las longitudes promedio aproximada de los artículos y la coherencia en los artículos del texto, con un solapamiento de un 25% del tamaño del chunk.

La siguiente imagen ilustra el resultado de la partición del texto.

```
[ ]  # ---------------------------- CONTROL DE CHUNKS ----------------------------------------------
     for i, txt in enumerate(textos_cortados['Marco_Legal_Educacion_Argentina']):
       # Imprimimos la longitud de la cadena, y luego el trozo de texto (chunk)
       print(f'Split: {i}/{len(textos_cortados["Marco_Legal_Educacion_Argentina"])}')
       print(f'Len: {len(txt)}\nTexto:\n{txt}')
       print('')
       # ---------------------------------------------------------------------------------------------
```

```
Split: 23/74
Len: 956
Texto:
expresar sus propias opiniones. Al contrario, los Estados partes deben dar por supuesto
que el niño tiene capacidad para formarse sus propias opiniones y reconocer que tiene
derecho a expresarlas; no corresponde al niño probar primero que tiene esa capacidad
(…) el artículo 12 no impone ningún límite de edad al derecho del niño a expresar su
opinión y desaconseja a los Estados partes que introduzcan por ley o en la práctica límites
de edad que restrinjan el derecho del niño a ser escuchado en todos los asuntos que lo
afectan…" (párrs. 19 y 21).
El artículo 681 CCyC establece como principio general un límite preciso para el ejercicio
del derecho a trabajar de las personas menores de edad, decidido por sí solos por las
personas menores de edad, fijándolo en la edad de 16 años.
La ley 26.390, impuso la prohibición de desempeño laboral, a las personas menores de
16 años, y estableció condiciones especiales de admisión al empleo a partir de los 16

Split: 24/74
Len: 967
Texto:
personas menores de edad, fijándolo en la edad de 16 años.
La ley 26.390, impuso la prohibición de desempeño laboral, a las personas menores de
16 años, y estableció condiciones especiales de admisión al empleo a partir de los 16
años. Es decir, para la franja de personas de entre 16 a 18 años, el desempeño laboral
requiere cumplimentar requisitos particulares, entre ellos, la autorización parental.
Se trata de una suerte de independencia del hijo adolescente, fundada en la formación de
este para un oficio o profesión y en la habilitación otorgada por el organismo competente,
que, por lo demás, en nada altera la obligación alimentaria de los progenitores que el
CCyC mantuvo en igual sentido que en el CC (ley 26.579): durante la menor edad y con
extensión a los 21 años de edad (art. 658 CCyC).
El art. 30 CCyC, exime de autorización a aquellas personas menores de edad que
hubiesen obtenido título habilitante para ejercer una profesión, autorización que se

Split: 25/74
Len: 945
```

Embeddings. Ahora que ya tenemos todos los textos cortados, es necesario realizar los embeddings para poder llevarlos a Chromadb.

```
[ ]  import chromadb

     # Creamos el objeto de la base de datos
     bd_chroma = chromadb.Client()
```

Como modelo de embedding se utiliza Universal **Sentence Encoder Multilingual**. La elección se basa nuevamente en una cuestión empírica de probar distintos modelos (tanto multilenguaje como no) y hacer consultas para ver con cual se daban mejores resultados (claro está que en el caso de los que no eran multilenguaje se obtuvieron resultados bastante peores). En chromadb se pueden utilizar como embedding modelos de la librería **Sentence Transformers**.

Al usar como embedding un modelo ajeno a Sentence Transformers fue necesario modificar la función de embedding para poder utilizar la función **get_collection()** más tarde.

```
import tensorflow_hub as hub
import numpy as np
import tensorflow_text
from chromadb import Documents, EmbeddingFunction, Embeddings

# Se utiliza universal sentence encoder multilenguaje como embedding
# es necesario modificar la función de embedding de ChromaDB (utilizada mas adelante a la hora de hacer get_collection)
class MyEmbeddingFunction(EmbeddingFunction):

    def __init__(self):
        # Variable de instancia
        self.embed = hub.load("https://www.kaggle.com/models/google/universal-sentence-encoder/frameworks/TensorFlow2/variations/multilingual/versions/2")

    def __call__(self, input: Documents) -> Embeddings:
        embeddings = self.embed(input).numpy().tolist()
        return embeddings

funcion_embedding = MyEmbeddingFunction()
```

Chromadb permite tener distintas colecciones donde almacenar embeddings y al tener distintos textos pertenecientes a distintas fuentes, por una cuestión de hacer la búsqueda un poco más específica enfocada solamente en el documento que el usuario consulte, se crea una colección por archivo. Previamente se generan los embeddings. Por último, se almacena en la colección tanto los embeddings, como los chunks, las ids y el nombre del texto como metadata.

```python
# Diccionario para los embeddings que generemos
embeddings_textos_cortados = {}

# Por cada archivo con split ya hecho
for nombre, archivo in textos_cortados.items():
    # Creamos los embeddings del documento
    embeddings_textos_cortados[nombre] = funcion_embedding(textos_cortados[nombre])
    # Creamos una coleccion perteneciente a ese archivo
    coleccion = bd_chroma.create_collection(name=f"{nombre}", metadata={"hnsw:space": "cosine"})
    # Agregamos:
    coleccion.add(
        embeddings=embeddings_textos_cortados[nombre],
        # Los splits
        documents=archivo,
        # Como metadata a que archivo pertenece dicho split
        metadatas = [ {'Archivo': nombre} for _ in range(len(archivo)) ],
        # Como id que número de chunk es dicho split
        ids=[f'Número de chunk {str(x)}' for x in range(len(archivo))]
    )
```

Búsqueda semántica

Con respecto a esta base de datos solo queda crear la función **busqueda_semantica(archivo, query)** que realiza la búsqueda semántica acorde al texto y a la consulta del usuario.

Simplemente con el nombre del artículo se busca en la colección correspondiente a la consulta y se realiza la consulta en sí, trayendo los chunks más cercanos. Al final se formatea el texto que es lo que se devuelve para ser utilizado en el contexto del LLM.

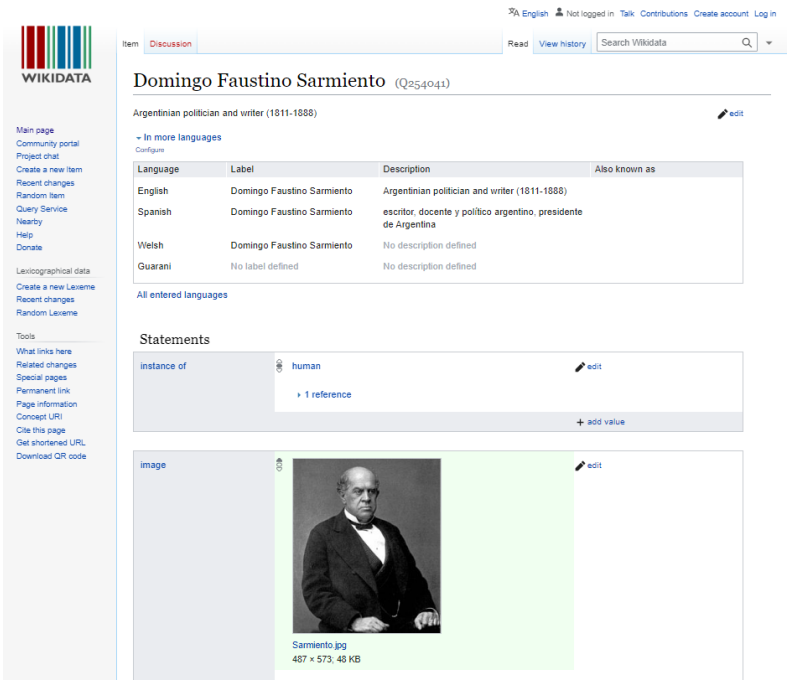
```python
# Función para realizar la busqueda semántica, recibe el string correspondiente al nombre de la colección (nombre del archivo) y la query
def busqueda_semantica(archivo, query):
    """
    Realiza una búsqueda semántica en la colección correspondiente al archivo utilizando una query.

    Parámetros:
    - archivo (str): Nombre del archivo (nombre de la colección) en la que se realizará la búsqueda.
    - query (str): Query semántica a utilizar en la búsqueda.

    Retorno:
    - str: Resultado formateado en un string que incluye información sobre la similitud coseno, el documento más cercano y su id.
    """

    # Cantidad de resultados que traera
    cantidad_resultados=3
    # A partir del nombre del archivo se obtiene la colección correspondiente
    coleccion = bd_chroma.get_collection(name=archivo, embedding_function=funcion_embedding)
    # Búsqueda
    resultado_query = coleccion.query(
        query_texts=query,
        n_results=cantidad_resultados,
        # Devuelve distancia (coseno seteada arriba), metadata (nombre del archivo) y los documentos mas cercanos (chunks correspondientes)
        include=['distances', 'metadatas', 'documents']
    )
    # Formateo de los resultados para luego pasar al contexto del LLM.
    resultado_limpio=f'Archivo: {archivo}\n\n'
    for i in range(cantidad_resultados):
      resultado_limpio += f'{resultado_query["ids"][0][i]}:\n'
      resultado_limpio += f'Similitud coseno: {resultado_query["distances"][0][i]}\n'
      resultado_limpio += f'{resultado_query["documents"][0][i]}:\n\n'

    # Devuelve el resultado formateado en un string
    return resultado_limpio
```

La siguiente imagen ilustra un resultado obtenido al realizar una **busqueda_semantica()** de ejemplo.

```
[ ]  1  # ---------------------------------- CONTROL DE CONSULTA VECTORIAL ----------------------------------------------------
     2  print('Control de Consulta Vectorial:')
     3  print('')
     4  pregunta = '¿Cúantos días le corresponde a una docente de licencia por maternidad?'
     5  print(f'Pregunta: {pregunta}')
     6  print('')
     7  print(busqueda_semantica('Estatuto_Docente_BsAs', pregunta))
     8  # ---------------------------------- FIN CONTROL DE CONSULTA VECTORIAL ------------------------------------------------

Control de Consulta Vectorial:

Pregunta: ¿Cúantos días le corresponde a una docente de licencia por maternidad?

Archivo: Estatuto_Docente_BsAs

Número de chunk 310:
Similitud coseno: 0.3605421781539917
mientras persistan los mismos en el lugar de trabajo de origen.
El inspector de enseñanza correspondiente, arbitrará los medios para que el cambio de destino transitorio
se opere dentro de los tres días de relevada la docente.
d.1.7. Las licencias por maternidad es obligatoria y la docente interesada
deberá solicitarla o en su defecto serán dispuestas de oficio por la autoridad médica o a pedido del
superior jerárquico inmediato. En caso de omisión de la solicitud y siendo ésta de oficio, se le otorgarán
los día que restan hasta completar el período pre-parto y luego los 90 días contados a partir del mismo.
d.1.8. Los haberes de la docente licenciada por maternidad serán liquidados
íntegramente sin interrupción debiéndose presentar la partida de nacimiento dentro de los tres (3) días de
expedida la misma. Los docentes provisionales y suplentes que en uso de licencia cesen su desempeño
continuaran percibiendo sus haberes hasta la finalización del período concedido con todos sus efectos:

Número de chunk 307:
Similitud coseno: 0.5008333325386047
concederán seis (6) días hábiles con goce de haberes a partir de la fecha de celebración del matrimonio.
La justificación de estas inasistencias se realizará con la presentación del certificado
de matrimonio expedido por autoridad competente.
d ) Por Maternidad o Adopción :
d.1. Por Maternidad:
d.1.1. Personal titular, provisional y suplente: El Personal femenino se le
otorgará licencia por embarazo y maternidad con goce íntegro de haberes, por el término de ciento treinta
y cinco (135) días corridos, a partir del 7º y medio mes de embarazo lo que se acreditará mediante la
presentación del certificado médico.
d.1.2. En los casos de nacimientos múltiples, y/ o prematuros (2,300 Kg. o
menos) a partir del séptimo y medio mes de gestación se ampliara hasta completar ciento cincuenta (150)
días corridos de licencia.
DGCyE / SSE / Dirección Legal y Técnica / 80
d.1.3. Cuando se produzcan nacimientos múltiples y/o prematuros entre el:

Número de chunk 314:
Similitud coseno: 0.5150641202926636
integro de haberes, en caso de adopción de hijos de hasta siete (7) años de edad, al personal femenino
durante noventa (90) días corridos, y al masculino durante cinco (5) días, a partir de la fecha de guarda o
tenencia con fines de adopción, otorgada por autoridad judicial o administrativa competente. Si se trata
de adopción múltiple, el término para el personal femenino se extenderá a ciento cinco (105) días
corridos.
d.2.2. Si durante el transcurso de la licencia ocurriere el fallecimiento del niño,
la misma caducara a la fecha del deceso.
```

Base de datos de grafos

Queries. Como se mencionó, esta base de datos contiene información general de algunas personalidades destacadas de la educación argentina. Esta información fue extraída de la web Wikidata y se eligieron sólo algunas personalidades a modo de ejemplo (Domingo Faustino Sarmiento, Carlos Pellegrini, Julio Argentino Roca y Arturo Jauretche) para evaluar la estrategia y el funcionamiento del algoritmo.



Entre los datos utilizados para almacenar en la base de datos de grafos se encuentran nombre, nacionalidad, fechas de nacimiento y fallecimiento, ocupaciones, etc.

Para su armado, se destinó un tiempo a la lectura y buceo bibliográfico para armar las consultas **sparql** según el esquema con que están desarrollados los contenidos y las propiedades disponibles.

Las consultas se crearon primero en **Wikidata Query Service** y luego se guardaron en variables en forma de string. Se creó una query por personaje, lo que generó que en su mayor parte el código de las consultas sea algo repetitivo, pero a los fines prácticos del presente trabajo, y a modo de ejemplo, es más que satisfactorio y cumple a la perfección con la finalidad para la que fue creado.

En la siguiente imagen se puede ver un ejemplo de la query de una de las personalidades registradas.

```
query_sarmiento = '''

SELECT

  ?nombreCompleto ?nacionalidad ?genero ?lealtad ?fechaNacimiento ?lugarNacimiento ?fechaFallecimiento
  ?lugarFallecimiento ?circunstanciasMuerte ?causaMuerte ?lugarSepultura ?madre
  (GROUP_CONCAT(DISTINCT ?_hermanos; SEPARATOR=", ") AS ?hermanos)
  ?conyuge ?hija
  (GROUP_CONCAT(DISTINCT ?_ocupaciones; SEPARATOR=", ") AS ?ocupaciones)
  (GROUP_CONCAT(DISTINCT ?_cargosOcupados; SEPARATOR=", ") AS ?cargosOcupados)
  ?inicioPeriodoActividad ?partidoPolitico ?candidatoEleccionAño ?rangoMilitarAlcanzado
  ?ramaMilitar
  (GROUP_CONCAT(DISTINCT ?_obrasDestacadas; SEPARATOR=", ") AS ?obrasDestacadas)

WHERE

{

  wd:Q254841 wdt:P1559 ?nombreCompleto;
             wdt:P27 [rdfs:label ?nacionalidad];
             wdt:P21 [rdfs:label ?genero];
             wdt:P945 [rdfs:label ?lealtad];
             wdt:P569 ?fechaNacimiento;
             wdt:P19 [rdfs:label ?lugarNacimiento];
             wdt:P570 ?fechaFallecimiento;
             wdt:P20 [rdfs:label ?lugarFallecimiento];
             wdt:P1196 [rdfs:label ?circunstanciasMuerte];
             wdt:P509 [rdfs:label ?causaMuerte];
             wdt:P119 [rdfs:label ?lugarSepultura];
             wdt:P25 [rdfs:label ?madre];
             wdt:P3373 ?hermanos;
             wdt:P26 [rdfs:label ?conyuge];
             wdt:P40 [rdfs:label ?hija];
             wdt:P106 ?ocupaciones;
             wdt:P39 ?cargosOcupados;
             wdt:P2031 ?inicioPeriodoActividad;
             wdt:P102 [rdfs:label ?partidoPolitico];
             wdt:P3602 [rdfs:label ?candidatoEleccionAño];
             wdt:P410 [rdfs:label ?rangoMilitarAlcanzado];
             wdt:P241 [rdfs:label ?ramaMilitar];
             wdt:P800 ?obrasDestacadas;

  SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],es".
                           ?hermanos rdfs:label ?_hermanos.
                           ?ocupaciones rdfs:label ?_ocupaciones.
                           ?cargosOcupados rdfs:label ?_cargosOcupados.
                           ?obrasDestacadas rdfs:label ?_obrasDestacadas.
                          }

  FILTER(LANG(?nacionalidad) = "es").
  FILTER(LANG(?genero) = "es").
  FILTER(LANG(?lealtad) = "es").
  FILTER(LANG(?lugarNacimiento) = "es").
  FILTER(LANG(?lugarFallecimiento) = "es").
  FILTER(LANG(?circunstanciasMuerte) = "es").
  FILTER(LANG(?causaMuerte) = "es").
  FILTER(LANG(?lugarSepultura) = "es").
  FILTER(LANG(?madre) = "es").
  FILTER(LANG(?conyuge) = "es").
  FILTER(LANG(?hija) = "es").
  FILTER(LANG(?partidoPolitico) = "es").
  FILTER(LANG(?candidatoEleccionAño) = "es").
  FILTER(LANG(?rangoMilitarAlcanzado) = "es").
  FILTER(LANG(?ramaMilitar) = "es").

}

GROUP BY ?nombreCompleto ?nacionalidad ?genero ?lealtad ?fechaNacimiento ?lugarNacimiento ?fechaFallecimiento
         ?lugarFallecimiento ?circunstanciasMuerte ?causaMuerte ?lugarSepultura ?madre ?conyuge ?hija ?inicioPeriodoActividad
         ?partidoPolitico ?candidatoEleccionAño ?rangoMilitarAlcanzado ?ramaMilitar
'''
```

Búsqueda en grafos

Con las querys listas, de manera similar a los casos anteriores, sólo resta crear una función **consulta_sparql(query)** que hace una solicitud GET a la página de consultas con la query pasada por parámetro, recibe un json con la respuesta de la consulta, la formatea y la devuelve (o devuelve un mensaje de error).

```python
import requests

def consulta_sparql(query):
    """
    Realiza una consulta SPARQL a Wikidata y devuelve los resultados formateados en un string.

    Parámetros:
    - query (str): Consulta SPARQL a ejecutar en Wikidata.

    Retorno:
    - str: String que contiene los resultados formateados de la consulta SPARQL.
      Retorna None si hay un error durante la consulta.
    """

    # URL donde se enviara la consulta
    url = "https://query.wikidata.org/sparql"

    # Configuración de la consulta SPARQL
    headers = {
        # Valor generalmente asociado a navegador web Chrome. Indica al servidor que la solicitud proviene de un navegador (aunque sea un script)
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36',
        # Queremos recibir un json
        'Accept': 'application/json'
    }

    params = {
        # Consulta SPARQL a consultar
        'query': query,
        # Recibimos un json.
        'format': 'json'
    }

    # Realización de la solicitud GET.
    response = requests.get(url, headers=headers, params=params)

    # Si la respuesta fue exitosa
    if response.status_code == 200:

        # Verificamos el json
        if response.json():

            # Creamos el string para el contexto con los resultados
            resultado = response.json()['results']['bindings'][0]
            respuesta_wikidata = ''
            for key in resultado.keys():
                respuesta_wikidata += f'{key}: {resultado[key]["value"]}\n'

        return respuesta_wikidata

    # Sino
    else:
        print("Error al realizar la consulta. Código de estado:", response.status_code)
        # Devolvemos nulo
        return None
```

La siguiente imagen muestra un ejemplo del resultado obtenido al ejecutar la búsqueda.

```python
# ---------------------------------- CONTROL DE CONSULTA DE GRAFOS ----------------------------------------------------
print('Control de Consulta de Grafos:')
print('')
print(consulta_sparql(query_sarmiento))
# ------------------------------ FIN DE CONTROL DE CONSULTA DE GRAFOS ----------------------------------------------
```

```
Control de Consulta de Grafos:

nombreCompleto: Domingo Faustino Sarmiento
nacionalidad: Argentina
genero: masculino
lealtad: Argentina
fechaNacimiento: 1811-02-14T00:00:00Z
lugarNacimiento: San Juan
fechaFallecimiento: 1888-09-11T00:00:00Z
lugarFallecimiento: Asunción
circunstanciasMuerte: causas naturales
causaMuerte: insuficiencia cardíaca
lugarSepultura: Cementerio de la Recoleta
madre: Paula Albarracín de Sarmiento
conyuge: Benita Martínez Pastoriza de Sarmiento
hija: Ana Faustina Sarmiento
inicioPeriodoActividad: 1834-01-01T00:00:00Z
partidoPolitico: Partido Unitario
candidatoEleccionAño: elecciones presidenciales de Argentina de 1868
rangoMilitarAlcanzado: soldado
ramaMilitar: Ejército Nacional de Colombia
hermanos: Procesa del Carmen Sarmiento, Bienvenida Sarmiento
ocupaciones: escritor, militar, político, diplomático, historiador, periodista
cargosOcupados: presidente de la Nación Argentina, senador de Argentina, embajador de Argentina en Chile, embajador de Argentina en los Estados Unidos
obrasDestacadas: Facundo o civilización y barbarie en las pampas argentinas, Recuerdos de provincia, Vida de Dominguito, Campaña en el Ejército Grande
```

Base de datos tabular

Importación: La base de datos tabular corresponde a un *DataFrame* creado con Pandas a partir de un dataset oficial denominado "**Padrón Oficial de Establecimientos Educativos**" y contiene toda la información general y de contacto de cada institución educativa del país.

```
[ ]  1  # Ruta y nombre del archivo xls
     2  drive_path = "/content/drive/MyDrive/UNR/4 - Proc Lenguaje Natural (IA42)/TP2/EduArDo/Data"
     3  xls_file = "Padron_Oficial_Establecimientos_Educativos.xls"
```

```
[ ]  1  import pandas as pd
     2
     3  # Ruta completa del archivo xls
     4  xls_path = f"{drive_path}/{xls_file}"
     5
     6  # Lee el archivo xls a partir de la fila 12 como cabecera
     7  padron_completo = pd.read_excel(xls_path, header=11)
```

```
1  padron_completo
```

	Jurisdicción	CUE Anexo	Nombre	Sector	Ámbito	Domicilio	CP	Código de área	Teléfono	Código localidad	...	Primaria.2	EGB3	Secundaria.3	Alfabetización	Formación Profesional	Formación Profesional (INET)	Inicial.2	Primaria.3	Secundaria.4	Servicios complementarios
0	Buenos Aires	60000100	JARDÍN DE INFANTES N°915 JAVIER VILLAFAÑE	Estatal	Urbano	TIRO FEDERAL 712	7300	02281	426487	6049005	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Buenos Aires	60000200	ESCUELA DE EDUCACIÓN PRIMARIA N°2 DOMINGO FAUS...	Estatal	Urbano	COLON Y MITRE 498	7300	02281	42-3361	6049005	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
						epnro2azul@yahoo.com.ar															
2	Buenos Aires	60000300	INSTITUTO PEDRO B. PALACIOS	Privado	Urbano	VICTOR MARTINEZ Y OSVALDO SOSA 1946	1757	011	4457-0054	6427006	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	Buenos Aires	60000400	INSTITUTO JUANA DE IBARBOUROU	Privado	Urbano	AV. ROJO 4415	1757	011	4626-1051	6427006	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Buenos Aires	60000600	ESCUELA DE TEATRO DE MORON	Estatal	Urbano	SAN MARTIN 620	1708	011	4629-3097	6568004	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
63241	Tucumán	909000500	ESCUELA DE AGRICULTURA Y SACAROTECNIA	Estatal	Rural	HORCO MOLLE	T4107	NaN	4253467(INT238)	90119004	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
63242	Tucumán	909000600	ESC. DE BELLAS ARTES Y ARTES DECOR. E INDUS. A...	Estatal	Urbano	LAPRIDA 246 NORTE	T4000	0381	4302967-4526018	90084004	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
63243	Tucumán	909000700	INST. SUPERIOR DE MUSICA	Estatal	Urbano	CHACABUCO 242	T4000	NaN	4220767	90084004	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
63244	Tucumán	909000800	INSTITUTO TECNICO DE AGUILARES	Estatal	Urbano	General Savio Italia	T4152	0381	483797/482728	90077001	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
63245	Tucumán	909000900	ESC. TECNICA N° 3 DE VIALIDAD NACIONAL GRAL. M...	Estatal	Urbano	SAN MARTÍN 1986 DIRECCIÓN DE VIALIDAD NACIONAL	T4000	0381	4514671	90084004	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

63246 rows × 45 columns

Análisis: Una vez realizada la carga se procede a efectuar las comprobaciones de rutina (info del DataFrame, cantidad de registros, presencia de datos duplicados, etc)

```
[ ]  1  filas_duplicadas = padron_completo.duplicated()
     2  print("Hay filas duplicadas.") if filas_duplicadas.any() else print("No hay filas duplicadas.")

     No hay filas duplicadas.
```

Filtrado: a modo opcional se agrega la posibilidad de filtrar el contenido del DataFrame para contener las instituciones de la/s provincia/s a elección.

```
[ ]  1  # Copia del padrón completo con los registros correspondientes distritos en particular
     2  distritos_a_conservar = ['Buenos Aires']
     3  padron = padron_completo[padron_completo['Jurisdicción'].isin(distritos_a_conservar)].copy()
```

```
[ ]  1  # Padrón Completo sin filtrar
     2  # padron = padron_completo.copy()
```

Búsqueda tabular

En el contexto de esta base de datos, se crea una función de **búsqueda_tabular(nombre_escuela)** que utiliza herramientas de Pandas para devolver un diccionario con todos los datos registrados para la institución en cuestión. En este sentido, es importante señalar que los nombres de las instituciones están registrados con todo el string en mayúsculas. Si bien se pudo haber procesado el dataframe para encontrar soluciones más apropiadas

desde la ciencia de datos, se prefirió optar por aplicar una estrategia más acorde a los contenidos de esta materia. Para ello se le pidió al LLM que procesara el prompt ingresado por el usuario, detectara el nombre de la institución, descartara todo el resto y lo devolviera convertido a mayúsculas. Las soluciones son favorables en muchos casos, pero funciona mucho mejor si en el prompt ingresado el nombre figura entre comillas, para evitar ciertas alucinaciones del modelo.

```python
def busqueda_tabular(nombre_escuela):
    """
    Busca una escuela por su nombre en el DataFrame "padron" y devuelve los datos correspondientes,
    omitiendo las columnas con valores NaN.

    Parámetros:
    - nombre_escuela (str): Nombre de la escuela a buscar.

    Retorno:
    - dict: Diccionario que contiene las columnas y valores de la escuela encontrada (sin NaN).
      Retorna None si la escuela no se encuentra.
    """
    # Buscar la escuela por nombre
    resultado_busqueda = padron[padron['Nombre'] == nombre_escuela]

    # Verificar si se encontraron resultados
    if not resultado_busqueda.empty:
        # Obtener la primera fila encontrada (suponiendo que los nombres son únicos)
        escuela_encontrada = resultado_busqueda.iloc[0]

        # Filtrar las columnas con valores no NaN
        escuela_valida = {indice: valor for indice, valor in escuela_encontrada.items() if pd.notna(valor)}
        # escuela_valida = [f"{indice}: {valor}" for indice, valor in escuela_encontrada.items() if pd.notna(valor)]

        return escuela_valida
    else:
        print(f"No se encontró ninguna escuela con el nombre '{nombre_escuela}'.")
        return None
```

A continuación, se muestra un ejemplo de

```python
# ---------------------------------- CONTROL DE CONSULTA TABULAR ----------------------------------------
print('Control de Consulta Tabular:')
print('')
escuela_buscar = 'INSTITUTO COMERCIAL RANCAGUA'
resultado = busqueda_tabular(escuela_buscar)
resultado
# ---------------------------- FIN DE CONTROL DE CONSULTA VECTORIAL ----------------------------------------
```

```
Control de Consulta Tabular:

{'Jurisdicción': 'Buenos Aires',
 'CUE Anexo': 61182300,
 'Nombre': 'INSTITUTO COMERCIAL RANCAGUA',
 'Sector': 'Privado',
 'Ámbito': 'Rural',
 'Domicilio': 'SANTA ANA 380  ',
 'CP': '2701',
 'Código de área': '02477',
 'Teléfono': '49-3017',
 'Código localidad': 6623033,
 'Localidad': 'RANCAGUA',
 'Departamento': 'PERGAMINO',
 'Mail': 'icr@cooprancagua.com.ar',
 'Ed. Común': 'X',
 'Secundaria': 'X'}
```

Chatbot

Definición: Una de las primeras tareas es definir la función correspondiente al **chatbot()** con el mensaje de bienvenida y despedida.

```
[ ]
def chatbot():

  while True:

    print('---------------------------------------------- Nuevo mensaje ----------------------------------------------')
    print('')

    print('''Asistente: Hola, soy "EduArDo", un chatbot especializado en temas relacionados
    al marco legal de la Educación Argentina. Si bien poseo información general de todo el país,
    me especializo en particular en la provincia de Buenos Aires.
    Puedo ayudarte con aspectos técnicos y normativos de leyes, reglamentos y estatutos.
    También puedo brindar información detallada de los personajes más influyentes
    en la historia educativa argentina (Sarmiento, Pellegrini, Roca, Jauretche),
    así como también buscarte información de contacto de una institución educativa en particular.
    En que puedo ayudarte? [Escribe "salir" para finalizar sesión]''')
    print('')

    pregunta = input('Usuario: ')
    print('')

    if (pregunta == 'salir'):
      print('Asistente: Espero haber sido de utilidad. Hasta pronto! ')
      break
```

Modelos de lenguaje: Como base para la resolución de esta parte se toma el último ejercicio de la Unidad 6 de la materia donde se utiliza el modelo Zephyr-7B-β (versión "fine-tuneada" de mistralai/Mistral7B-v0.1) disponible en HuggingFace y el cual se utiliza por medio de la API de la página.

Hubo dos funciones que se mantienen tal cual estaban. Una es **plantilla_instruccion_zephyr(messages)** que utiliza Jinja para formatear la plantilla a partir del mensaje recibido.

```
[ ]
# Funcion que define la plantilla jinja
def plantilla_instruccion_zephyr(messages, add_generation_prompt=True):
  template_str = "{% for message in messages %}"
  template_str += "{% if message['role'] == 'user' %}"
  template_str += "<|user|>{{ message['content'] }}</s>\n"
  template_str += "{% elif message['role'] == 'assistant' %}"
  template_str += "<|assistant|>{{ message['content'] }}</s>\n"
  template_str += "{% elif message['role'] == 'system' %}"
  template_str += "<|system|>{{ message['content'] }}</s>\n"
  template_str += "{% else %}"
  template_str += "<|unknown|>{{ message['content'] }}</s>\n"
  template_str += "{% endif %}"
  template_str += "{% endfor %}"
  template_str += "{% if add_generation_prompt %}"
  template_str += "<|assistant|>\n"
  template_str += "{% endif %}"
  # Crear un objeto de plantilla con la cadena de plantilla
  template = Template(template_str)
  # Renderizar la plantilla con los mensajes proporcionados
  return template.render(messages=messages, add_generation_prompt=add_generation_prompt)
```

Y la otra es **conexion_llm(prompt)** que se conecta al modelo via API. En esta última se mantienen la misma cantidad de max_new_tokens, se configura temperature (va de 0 a 100) en 0.01 con el objetivo de que sea lo más determinista posible y evitar la aleatoriedad de las respuestas dejando de lado las posibilidades de creatividad en las respuestas, top_k y top_p se setearon en None (por defecto) ya que luego de probar y probar no pude notar diferencias con las distintas configuraciones.

```python
# Llamada al modelo
def conexion_llm(prompt: str, max_new_tokens: int = 768) -> None:
  try:
    # Tu clave API de Hugging Face
    api_key = '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'

    # URL de la API de Hugging Face para la generación de texto
    api_url = "https://api-inference.huggingface.co/models/HuggingFaceH4/zephyr-7b-beta"

    # Cabeceras para la solicitud
    headers = {"Authorization": f"Bearer {api_key}"}

    # Datos para enviar en la solicitud POST
    # Sobre los parámetros: https://huggingface.co/docs/transformers/main_classes/text_generation
    data = {
      "inputs": prompt,
      "parameters": {
        "max_new_tokens": max_new_tokens,
        "temperature": 0.01,
      }
    }

    # Realizamos la solicitud POST
    response = requests.post(api_url, headers=headers, json=data)

    # Extraer respuesta
    respuesta = response.json()[0]["generated_text"][len(prompt):]

    return respuesta

  except Exception as e:
    print(f"Error durante la conexion con el LLM: Error {e}")
```

Clasificadores: Como principio básico del funcionamiento del chatbot, la idea primaria es que luego de que el usuario ingrese un prompt, un clasificador determine a qué clase pertenece. Una vez realizada la primera clasificación se puede definir la base de datos en la que debe realizarse la búsqueda y obtener de ella la información de contexto necesaria para que el LLM elabore la respuesta a la consulta.

Para la primera etapa se utiliza el mismo LLM para clasificar en una de las tres categorías.

```python
# Clasificador creado con el mismo LLM para saber a que base de datos ir,
# elige un numero dependiendo la base de datos que se necesite.
clasificacion_bdd_llm = clasificador_base_datos(pregunta)
```

Se le da al modelo como entrada el prompt indicándole que es un clasificador de texto en categorías, se le da información de las tres categorías (bases de datos) y se le agrega una cuarta en caso de que no pertenezca a ninguna de las anteriores.

```python
#@title Clasificador de Bases de Datos

# Primera etapa del clasificador de categorias:
def clasificador_base_datos(prompt_inicial: str):

  PLANTILLA_CLASIFICACION_BASE_DATOS = (
    "Clasifica el texto recibido en, estrictamente, una de las categorías mencionadas a continuación.\n"
    "Categorias:\n"
    "1\n"
    "2\n"
    "3\n"
    "4\n"
    '''La categoria 1 corresponde al marco legal de la educación argentina y en particular de la provincia de Buenos Aires.
    Corresponde a leyes, normas y regulaciones que afecten a las instituciones educativas o a cualquiera de los agentes
    y personas que participan en ellas.
    Por ejemplo: artículos de la Ley de Educación Nacional, del Reglamento General de Escuelas de Buenos Aires, del Estatuto Docente, etc.\n'''
    '''La categoria 2 corresponde a datos generales sobre los personajes más influyentes en la historia educativa argentina.
    Por ejemplo: datos sobre Domingo Faustino Sarmiento, Carlos Pellegrini, Julio Argentino Roca y Arturo Jauretche.\n'''
    '''La categoria 3 corresponde a datos específicos de todas las instituciones educativas del país.
    Por ejemplo: nombre, CUE, jurisdicción, sector público o privado, domicilio, localidad, teléfono, mail, etc.\n'''
    '''La categoria 4 corresponde a consultas que no estén relacionadas con ninguna de las anteriores.\n'''
    "Pregunta: {prompt_inicial}\n"
    "Respuesta: "
  )

  mensaje = [
    {
      "role": "system",
      "content": '''Eres un experto clasificador de textos en categorias.''',
    },
    {"role": "user", "content": PLANTILLA_CLASIFICACION_BASE_DATOS.format(prompt_inicial=prompt_inicial)},
  ]

  prompt_plantilla = plantilla_instruccion_zephyr(mensaje)

  clasificacion_base_datos = conexion_llm(prompt_plantilla)

  return clasificacion_base_datos
```

Idealmente se hubiera querido que el modelo de una respuesta del tipo: “Categoría 1” o “1”, o una salida bastante estándar y parecida siempre que se ingrese un prompt de la misma categoría, para simplificar la toma de decisiones posteriores relacionadas a las búsquedas. Pero a pesar de probar varias estrategias nunca se pudo lograr. Para solucionarlo, se decidió pasar esa misma salida por el modelo de clasificación **zero-shot** de **HuggingFace** “MoritzLaurer/mDeBERTa-v3-base-mnli-xnli”.

```
# Como no se logro que el clasificador anterior devuelva solo un número para las categorias
# Se pasa el resultado anterior a otro clasificador (zero-shot sacado de HugginFace) para obtener la categoria.
clasificacion_bdd_zero = clasificador_zero_shot(clasificacion_bdd_llm, etiquetas_base_datos, multi_label=False)
```

```
# Clasificador zero-shot de HugginFace
# Utilizado como segunda etapa del Clasificador de categorias y clasificador de archivos
clasificador_zero_shot = pipeline("zero-shot-classification", model="MoritzLaurer/mDeBERTa-v3-base-mnli-xnli")

# Se utilizan dos listas de etiquetas distintas porque en un caso el clasificador busca textos,
# y en el otro caso lo que busca es información de personalidades.

# Dos grupos de etiquetas para el clasificador
etiquetas_textos = ['Marco Legal',
                    'Ley Educación Nacional 26206',
                    'Reglamento General',
                    'Estatuto Docente']

etiquetas_personas = ['Domingo Faustino Sarmiento',
                      'Carlos Pellegrini', 'Julio Argentino Roca',
                      'Arturo Jauretche']

# Etiquetas para la segunda etapa del clasificador de categorias
etiquetas_base_datos = ['1', '2', '3', '4']
```

De esta manera, se conforma un clasificador de dos etapas para procesar el prompt inicial y definir en qué base de datos se realizará la búsqueda de datos de contexto para el LLM. Los resultados obtenidos son satisfactorios.

(El clasificador zero-shot por sí solo tampoco funcionó bien clasificando los prompt en las categorías, no hubo forma de expresar las categorías con pocas palabras)

```
---------- Nuevo mensaje ----------

Asistente: Hola, soy "EduArDo", un chatbot especializado en temas relacionados
    al marco legal de la Educación Argentina. Si bien poseo información general de todo el país,
    me especializo en particular en la provincia de Buenos Aires.
    Puedo ayudarte con aspectos técnicos y normativos de leyes, reglamentos y estatutos.
    También puedo brindar información detallada de los personajes más influyentes
    en la historia educativa argentina (Sarmiento, Pellegrini, Roca, Jauretche),
    así como también buscarte información de contacto de una institución educativa en particular.
    En que puedo ayudarte?

Usuario: salir

Asistente: Espero haber sido de utilidad. Hasta pronto!
```

```
Usuario: Según el estatuto docente, cuanto tiempo le corresponde de licencia a una docente por maternidad?

--------------------------------------------------------------------------

Control Clasificador de categorias de 2 etapas:

Etapa 1: Clasificacion de LLM
Este texto se clasifica exclusivamente en la categoría 1, correspondiente al marco legal de la educación ar

Etapa 2: Clasificacion Zero-Shot
Etiquetas: ['1', '3', '2', '4']
Puntajes: [0.9933927655220032, 0.0023087584413588047, 0.0021906509064137936, 0.0021078367717564106]

Categoria: Marco legal

--------------------------------------------------------------------------
```

```
Usuario: Puedes darme información de Julio A. Roca?

--------------------------------------------------------------------------

Control Clasificador de categorias de 2 etapas:

Etapa 1: Clasificacion de LLM
La pregunta se encaja en la categoría 2, que corresponde a datos generales sobre los personajes más influye

Etapa 2: Clasificacion Zero-Shot
Etiquetas: ['2', '1', '3', '4']
Puntajes: [0.9932313561439514, 0.003067502053454518 3, 0.0019678131211549044, 0.001733302604407072]

Categoria: Informacion de personalidades
```

```
Usuario: necesito los datos del Instituto Comercial Rancagua. Puedes darmelos?

--------------------------------------------------------------------------

Control Clasificador de categorias de 2 etapas:

Etapa 1: Clasificacion de LLM
La categoría adecuada para la pregunta es la categoría 3, que corresponde a datos específicos de todas las

Etapa 2: Clasificacion Zero-Shot
Etiquetas: ['3', '2', '4', '1']
Puntajes: [0.9872374534606934, 0.00487759942188859, 0.004056375473737717, 0.0038285120390355587]

Categoria: Datos de instituciones
```

Finalmente, si la base de datos con mayor probabilidad del clasificador zero-shot no supera cierto umbral mínimo para asegurar eficacia o simplemente la categoría es la 4 (ninguna de las bases de datos), se imprime un mensaje en pantalla que simula ser una respuesta del chatbot orientando al usuario y mencionando qué tipo de información dispone.

```
Usuario: Quien escribió "El origen de las especies?"

--------------------------------------------------------------------------

Control Clasificador de categorias de 2 etapas:

Etapa 1: Clasificacion de LLM
La pregunta no se encaja en ninguna de las categorías mencionadas. Por lo tanto, se clasifica en la categoría 4 de consultas que no estár

Etapa 2: Clasificacion Zero-Shot
Etiquetas: ['4', '3', '2', '1']
Puntajes: [0.9818061590194702, 0.007497689686715603, 0.006160993594676256, 0.0045350999571383]

Categoria: No corresponde a una categoria

--------------------------------------------------------------------------

Asistente: Disculpa pero no estoy seguro de comprender qué tipo de información necesitas.
      Recuerda que sólo puedo ayudarte con el marco legal de la educación argentina, personalidades destacadas y datos institucionales.
      ¿Puedes reformular la pregunta?
```

Búsqueda de contexto: Si el resultado de la clasificación no es la categoría 4 y el modelo está “bastante seguro” de que se trata de una de las tres bases de datos, el algoritmo continúa con la búsqueda de la información de contexto con ejecuciones diferentes según la categoría resultante. Se detallan a continuación los comportamientos en cada uno de los casos.

Categoría 1. Si la categoría identificada en el clasificador inicial es la 1 (Marco Legal), se usa nuevamente el clasificador zero-shot pero esta vez para definir a qué texto se refiere la consulta.

```python
# CATEGORÍA = 1
# --> preguntas del marco legal / búsqueda semántica / base de datos vectorial:
if clasificacion_bdd_zero['labels'][0] == '1':

  # Usamos nuevamente el clasificador zero-shot de HF para saber de que texto se trata la solicitud del usuario
  clasificacion_zero_textos = clasificador_zero_shot(pregunta, etiquetas_textos, multi_label=False)
```

Recordemos que la idea es ir a buscar los chunks de textos más cercanos a nuestro prompt en la colección de chromadb correspondiente, la función de la búsqueda vectorial necesita un nombre de colección (ver Base de datos vectorial).

```
Usuario: Según el Estatuto Docente, cuál es la licencia por matrimonio?

--------------------------------------------------------------------------

Control Clasificador de categorias de 2 etapas:

Etapa 1: Clasificacion de LLM
La categoría adecuada para clasificar el texto recibido es la categoría 1, correspondiente al marco legal de la educación

Etapa 2: Clasificacion Zero-Shot
Etiquetas: ['1', '3', '4', '2']
Puntajes: [0.9935643076896667, 0.0022175482008606195, 0.002117132768034935, 0.0021010227501392365]

Categoria: Marco legal

--------------------------------------------------------------------------

--------------------------------------------------------------------------

Control de Clasificador de Textos (Para BD vectorial):

Etiquetas: ['Estatuto Docente', 'Marco Legal', 'Reglamento General', 'Ley Educación Nacional 26206']
Puntajes: [0.8200957775115967, 0.16478832066059113, 0.009640488773584366, 0.0054754517041146755]

Texto: Estatuto Docente

--------------------------------------------------------------------------
```

Con la etiqueta definida, se verifica si el valor de la probabilidad supera un umbral mínimo que asegure eficacia. En caso negativo se muestra un mensaje de error y en caso positivo se procede a realizar la búsqueda semántica con el prompt **limpio**.

```python
# Si la etiqueta más probable es menor a umbral mínimo
if clasificacion_zero_textos['scores'][0] < 0.50:

  # No entedemos bien a que texto se refiere
  print('''Asistente: Disculpa pero no entiendo a qué tipo de documentación hace referencia tu consulta.
  Recuerda que por el momento sólo poseo conocimientos en "Marco Legal de la Educación Argentina", "Ley de Educación Nacional 26206",
  "Reglamento General de Escuelas de Buenos Aires" y "Estatuto Docente de Buenos Aires".
  ¿Podrías reformular tu pregunta?\n''')

# Si la etiqueta más probable supera ese umbral
else:

  prompt_limpio = limpiar_prompt_texto(pregunta)
```

La función **impiar_prompt_texto(prompt_inicial)** recibe como parámetro la consulta ingresada por el usuario y se le pide al modelo que quite el nombre del texto con 3-shots de ejemplo, se confecciona la plantilla jinja, se establece la conexión con el modelo y se realiza una limpieza final de la salida dado que el modelo utiliza la misma frase mostrada en el ejemplo.

```python
# Funcion que utiliza el LLM para limpiar el prompt de consulta a la base de datos semantica:
def limpiar_prompt_texto(prompt_inicial: str):

  PLANTILLA_LIMPIAR_PROMPT_TEXTO = (
      "Cadena de texto: {prompt_inicial}.\n"
  )

  mensaje = [
    {
      "role": "system",
      "content": '''Eres un asistente especialista en sintaxis que recibe una cadena de texto ingresada por el usuario.
      El prompt ingresado tratará acerca de algún documento sobre la normativa y el marco legal de la Educación Argentina
      Tu tarea es devolver la misma cadena pero quitándole el nombre del documento en cuestión.\n
      Los documentos a los que se puede hacer referencia son "Marco Legal de la Educación Argentina",
      "Ley de Educación Nacional 26206", "Reglamento General de Escuelas de Buenos Aires" y
      "Estatuto Docente de Buenos Aires".\n'''
      "-------------------------------------------------------------------------\n"
      '''Ejemplo 1:\n
      Cadena de texto: ¿Cuánto tiempo es la licencia por matrimonio según el Estatuto Docente?\n
      Cadena de texto sin el nombre del documento: ¿Cuánto tiempo es la licencia por matrimonio?\n'''
      '''Ejemplo 2:\n
      Cadena de texto: ¿Según la Ley de Educación Nacional, un docente tiene derecho a capacitación?\n
      Cadena de texto sin el nombre del documento: ¿Un docente tiene derecho a capacitación?\n'''
      '''Ejemplo 3:\n
      Cadena de texto: ¿Según el Reglamento General de Escuelas, quién debe reemplazar al director en caso de ausencia?\n
      Cadena de texto sin el nombre del documento: ¿quién debe reemplazar al director en caso de ausencia?\n'''
      "-------------------------------------------------------------------------\n"
      '''Tu salida debe ser únicamente el texto formateado.\n''',
    },
    {"role": "user", "content": PLANTILLA_LIMPIAR_PROMPT_TEXTO.format(prompt_inicial=prompt_inicial)},
  ]

  prompt_plantilla = plantilla_instruccion_zephyr(mensaje)

  prompt_limpio_llm = conexion_llm(prompt_plantilla)

  # Últimos retoques que dejan el prompt limpio
  prompt_limpio = prompt_limpio_llm.split('Cadena de texto sin el nombre del documento: ')[1].split('\n')[0]

  return prompt_limpio
```

Con el prompt limpio se realiza la búsqueda semántica en la base de datos vectorial (en el texto correspondiente según el resultado de la salida del zero-shot de textos) invocando a la función **respuesta_chatbot()** y le pasaremos el resultado al LLM como información de contexto junto con la consulta para que responda.

```python
# Dependiendo el texto que nombre la etiqueta, buscamos en la base de datos vectorial y generamos la respuesta
if clasificacion_zero_textos['labels'][0] == 'Marco Legal': print(f'Asistente: {respuesta_chatbot("Marco_Legal_Educacion_Argentina", prompt_limpio,"1")}')
elif clasificacion_zero_textos['labels'][0] == 'Ley Educación Nacional 26206': print(f'Asistente: {respuesta_chatbot("Ley_Educacion_Nacional_26206", prompt_limpio,"1")}')
elif clasificacion_zero_textos['labels'][0] == 'Reglamento General': print(f'Asistente: {respuesta_chatbot("Reglamento_Gral_Escuelas_BsAs", prompt_limpio,"1")}')
elif clasificacion_zero_textos['labels'][0] == 'Estatuto Docente': print(f'Asistente: {respuesta_chatbot("Estatuto_Docente_BsAs", prompt_limpio,"1")}')
```

Categoría 2. Si el usuario ingresa una consulta relacionada con información de personalidades históricas, el clasificador inicial identificará que corresponde a la categoría 2. Si eso sucede, se utiliza nuevamente el clasificador zero-shot con las etiquetas de las personas (*ver Clasificadores*)

```python
# CATEGORÍA = 2
# --> preguntas de información de personalidades / búsqueda sparql / base de datos grafos:
elif clasificacion_bdd_zero['labels'][0] == '2':

  # Usamos nuevamente el clasificador zero-shot de HF para saber a qué personalidad refiere la pregunta
  clasificacion_zero_pers = clasificador_zero_shot(pregunta, etiquetas_personas, multi_label=False)
```

Con la etiqueta de la persona definida, se repite la verificación si el valor de probabilidad supera el umbral. En caso negativo se muestra un mensaje de error y en caso positivo se procede a realizar la búsqueda de los datos de contexto mediante la consulta sparql correspondiente a esa etiqueta.

```python
# Dependiendo el apellido que nombre la etiqueta, buscamos en la base de datos de grafos
# con la query correspondiente generada arriba y generamos la respuesta
if clasificacion_zero_pers['labels'][0] == 'Domingo Faustino Sarmiento': print(f'Asistente: {respuesta_chatbot(query_sarmiento, pregunta, "2")}')
elif clasificacion_zero_pers['labels'][0] == 'Carlos Pellegrini': print(f'Asistente: {respuesta_chatbot(query_pellegrini, pregunta, "2")}')
elif clasificacion_zero_pers['labels'][0] == 'Julio Argentino Roca': print(f'Asistente: {respuesta_chatbot(query_roca, pregunta, "2")}')
elif clasificacion_zero_pers['labels'][0] == 'Arturo Jauretche': print(f'Asistente: {respuesta_chatbot(query_jauretche, pregunta, "2")}')
```

A continuación, se muestra un ejemplo de la ejecución del clasificador zero-shot de personas.

```
Control Clasificador de categorias de 2 etapas:

Etapa 1: Clasificacion de LLM
La pregunta se encaja en la categoría 2, que corresponde a datos generales sobre los personajes más influyentes en la historia educativa

Etapa 2: Clasificacion Zero-Shot
Etiquetas: ['2', '1', '3', '4']
Puntajes: [0.9945664405822754, 0.0034897627774626017, 0.0011310124536976218, 0.0008128167828544974]

Categoria: Informacion de personalidades

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

Control Clasificador de Personalidades (Para BD de grafos)

Etiquetas: ['Julio Argentino Roca', 'Domingo Faustino Sarmiento', 'Carlos Pellegrini', 'Arturo Jauretche']
Puntajes: [0.9482253789901733, 0.018571943044662476, 0.017763923853635788, 0.015438742004334927]

Personalidad: Julio Argentino Roca

--------------------------------------------------------------------------------
```

Categoría 3. Si el usuario ingresa una consulta relacionada con información de instituciones educativas, el clasificador inicial identificará que corresponde a la categoría 3.

```
Usuario: Necesito información sobre el 'Jardín San Jorge'

--------------------------------------------------------------------------------

Control Clasificador de categorias de 2 etapas:

Etapa 1: Clasificacion de LLM
La categoría adecuada para el texto recibido es la categoría 3, que corresponde a datos específicos de todas las instituciones educativas del país

Etapa 2: Clasificacion Zero-Shot
Etiquetas: ['3', '2', '1', '4']
Puntajes: [0.9969838857650757, 0.001204505329951644, 0.0009202930959872901, 0.0008912768680602312]

Categoria: Datos de instituciones

--------------------------------------------------------------------------------
```

Si eso sucede, simplemente hay que realizar la búsqueda tabular con el nombre de la institución dentro del *DataFrame*. Para ello, se necesita el prompt limpio y en mayúsculas.

```python
# CATEGORÍA = 3
# --> preguntas de información de instituciones / búsqueda tabular / base de datos tabular:
elif clasificacion_bdd_zero['labels'][0] == '3':

  prompt_mayus = limpiar_prompt_mayus(pregunta)

  # ------------------------------------ Control de Filtro de Institución ----------------------------------------------------------
  # Descomentar para controlar como se formateó el prompt
  print('--------------------------------------------------------------------------------')
  print('')
  print('Control de Filtro de Institución (Para BD Tabular)')
  print('')
  print(f'Prompt original: {pregunta}')
  print('')
  print(f'Prompt convertido a mayúsculas: {prompt_mayus}')
  print('')
  print('--------------------------------------------------------------------------------')
  print('')
  # ---------------------------------- Fin control de Filtro de Institución  ----------------------------------------------------------

  print(f'Asistente: {respuesta_chatbot(prompt_mayus, pregunta, "3")}')
```

La función **impiar_prompt_mayus(prompt_inicial)** recibe como parámetro la consulta ingresada por el usuario y se le pide al modelo que extraiga el nombre de la institución mencionada con 3-shots de ejemplo, se confecciona la plantilla jinja, se establece la conexión con el modelo y se realiza una limpieza final de la salida dado que el modelo utiliza la misma frase mostrada en el ejemplo.

```python
# Funcion que utiliza el LLM para limpiar el prompt de consulta a la base de datos semantica:
def limpiar_prompt_mayus(prompt_inicial: str):

  PLANTILLA_LIMPIAR_PROMPT_MAYUS = (
      "Cadena de texto: {prompt_inicial}.\n"
  )

  mensaje = [
    {
      "role": "system",
      "content": '''Eres un asistente especialista en gramática que recibe una cadena de texto ingresada por el usuario.
      El prompt ingresado tratará acerca de alguna institución educativa de Argentina.
      Tu tarea es filtrar el prompt y devolver únicamente el nombre de la institución mencionado en el mismo, todo en mayúsculas, quitándole el resto del texto.
      No deberás modificar el nombre de la institución ni agregar texto.\n'''
      "-------------------------------------------------------------------------\n"
      "Ejemplo 1:\n"
      "Cadena de texto: ¿Puedes darme los datos de contacto del 'Instituto Comercial Rancagua'?\n"
      "Cadena de texto de salida en mayúsculas: INSTITUTO COMERCIAL RANCAGUA\n"
      "Ejemplo 2:\n"
      "Cadena de texto: Necesito información sobre el 'Instituto Politecnico Nº 6 General San Martin'\n"
      "Cadena de texto de salida en mayúsculas: INSTITUTO POLITECNICO Nº 6 GENERAL SAN MARTIN\n"
      "Ejemplo 3:\n"
      "Cadena de texto: ¿Tienes datos de contacto del 'Colegio Nuestra Señora del Huerto'?\n"
      "Cadena de texto de salida en mayúsculas: COLEGIO NUESTRA SEÑORA DEL HUERTO\n"
      "-------------------------------------------------------------------------\n"
      '''Tu salida debe ser únicamente el texto formateado.\n''',
    },
    {"role": "user", "content": PLANTILLA_LIMPIAR_PROMPT_MAYUS.format(prompt_inicial=prompt_inicial)},
  ]

  prompt_plantilla = plantilla_instruccion_zephyr(mensaje)

  prompt_mayus_llm = conexion_llm(prompt_plantilla)

  # Últimos retoques que dejan el prompt limpio
  prompt_mayus = prompt_mayus_llm.split('Cadena de texto de salida en mayúsculas: ')[1].split('\n')[0]

  return prompt_mayus
```

A continuación, se muestra un ejemplo del funcionamiento de la función.

```
Usuario: Necesito el teléfono del 'Instituto Comercial Rancagua'. Lo tienes?

--------------------------------------------------------------------------------

Control Clasificador de categorias de 2 etapas:

Etapa 1: Clasificacion de LLM
La pregunta corresponde a la categoría 3 de clasificación de textos.

Etapa 2: Clasificacion Zero-Shot
Etiquetas: ['3', '2', '4', '1']
Puntajes: [0.9811760783195496, 0.007929276674985886, 0.005647573620080948, 0.005247019696980715]

Categoria: Datos de instituciones

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

Control de Filtro de Institución (Para BD Tabular)

Prompt original: Necesito el teléfono del 'Instituto Comercial Rancagua'. Lo tienes?

Prompt convertido a mayúsculas: INSTITUTO COMERCIAL RANCAGUA

--------------------------------------------------------------------------------
```

Respuesta ChatBot. La función **respuesta_chatbot(nombre, pregunta, categoria)** busca el contexto en la base de datos que corresponda con el tipo de búsqueda pertinente, según la categoría que el chatbot le mande como parámetro:

```python
# Función que deriva la consulta a la base de datos correspondiente para buscar el contexto y luego genera la respuesta
def respuesta_chatbot(nombre, pregunta: str, categoria: str):

  if categoria == '1': resultado_bdd = busqueda_semantica(nombre, pregunta)
  elif categoria == '2': resultado_bdd = consulta_sparql(nombre)
  elif categoria == '3': resultado_bdd = busqueda_tabular(nombre)

  respuesta = generar_respuesta_final(pregunta, resultado_bdd)

  return respuesta
```

- Si la categoría es “1” realiza una búsqueda semántica en la base de datos vectorial con el nombre del texto y la pregunta como parámetros.
- Si la categoría es “2” realiza una consulta sparql en la base de datos de grafos con el nombre de la persona a buscar.
- Si la categoría es “3” realiza una búsqueda tabular en la base de datos correspondiente con el nombre de la institución.

El resultado de la búsqueda (cualquiera sea) se almacena en la variable **resultado_bdd** que será la información de contexto que se le provea al LLM.

Respuesta Final. La función **generar_respuesta_final(query, contexto)** genera la respuesta final y es compartida por todas las categorías. Recibe la consulta y el contexto, nuevamente se le dan las instrucciones al LLM, se genera la plantilla con Jinja, se hace la conexión con el LLM y se devuelve la respuesta final que es la que recibirá el usuario.

```python
# Función para generar la respuesta final (compartida para todas las categorías)
def generar_respuesta_final(query_str: str, contexto: str):

  TEXT_QA_PROMPT_TMPL = (
    "\n\n---------------------------------------------------------------------\n"
    "Informacion de contexto:\n"
    "{context_str}\n"
    "---------------------------------------------------------------------\n"
    "Pregunta: {query_str}\n"
    "Respuesta: \n\n"
  )


  messages = [
    {

    "role": "system",
    "content": '''Responde en ESPAÑOL.\n
Eres un asistente especializado en educación argentina que responde amablemente preguntas guiándose únicamente por la
informacion de contexto pero respondes como si esta no existiera y fueras tu quien lo sabía de antemano.\n
'''
    "Responde lo más corto posible.\n"
    "Responde sin utilizar conocimiento previo.\n"
    '''Si la respuesta no esta en la informacion de contexto escribe la siguiente respuesta: 'Disculpa no puedo
responder a la pregunta en este momento'.\n'''

    },

    {"role": "user",
    "content": TEXT_QA_PROMPT_TMPL.format(context_str=contexto, query_str=query_str)},
  ]

  prompt_final = plantilla_instruccion_zephyr(messages)

  # # ------------------------------------ CONTROL DE PROMPT QUE RECIBE EL LLM  ------------------------------------------------
  # print('-------------------------------------------------------------------------------------')
  # print('')
  # print('Control Prompt que recibe el LLM:')
  # print('')
  # print(prompt_final)
  # print('-------------------------------------------------------------------------------------')
  # print('')
  # # ------------------------------------ FIN CONTROL DE PROMPT QUE RECIBE EL LLM  ---------------------------------------------

  respuesta_final = conexion_llm(prompt_final)

  return respuesta_final
```

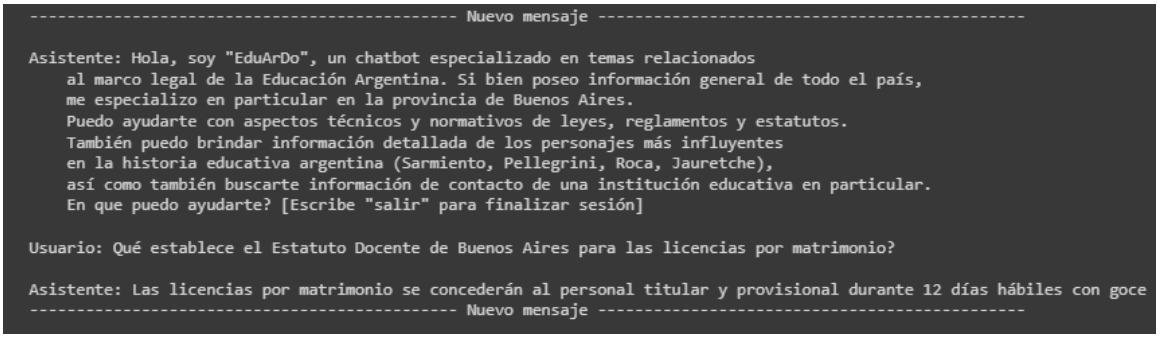
Repositorio GitHub

Todos los archivos correspondientes a la resolución de este ejercicio se encuentran alojados en un repositorio público en GitHub.

https://github.com/MiguelMussi/NLP_TP2

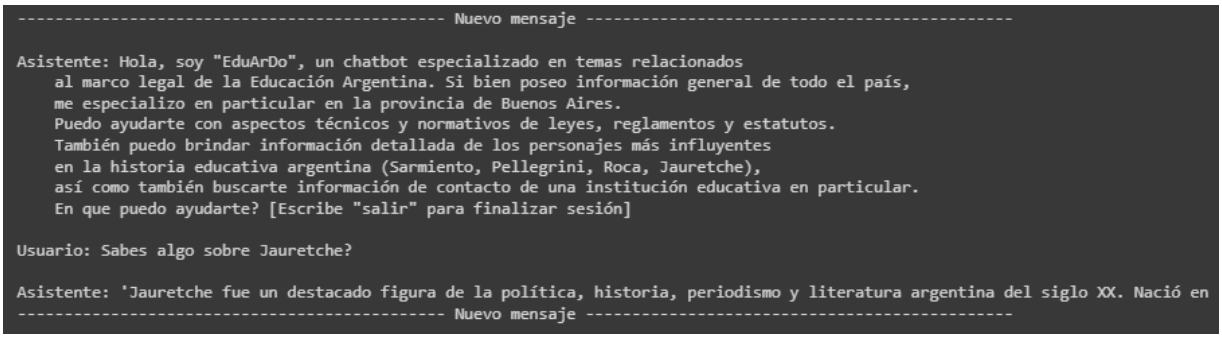
Ejemplos de consultas

CATEGORÍA 1: Búsqueda de información sobre **Marco Legal**.

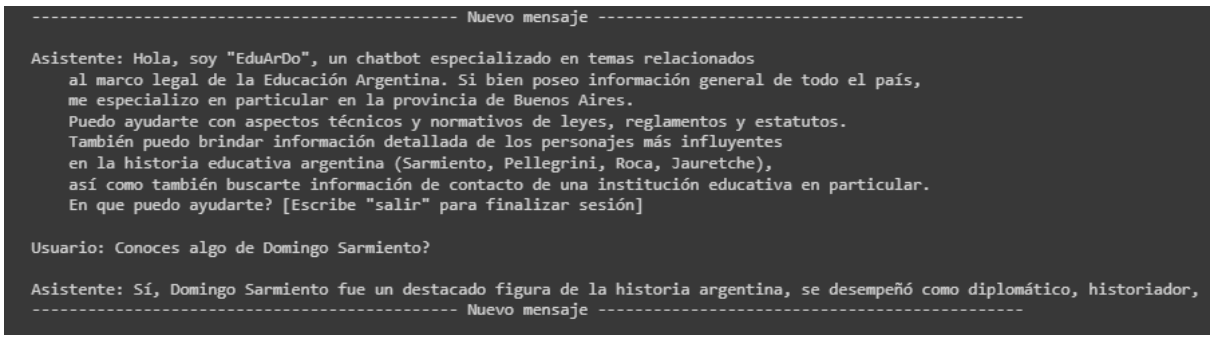


Asistente: *Las licencias por matrimonio se concederán al personal titular y provisional durante 12 días hábiles con goce de haberes a partir de la fecha de celebración del matrimonio, y al personal suplente durante 6 días hábiles con goce de haberes en la misma condición. La justificación de estas inasistencias se realizará con la presentación del certificado de matrimonio expedido por autoridad competente.*

CATEGORÍA 2: Búsqueda de información sobre **personalidades**.



Asistente: *'Jauretche fue un destacado figura de la política, historia, periodismo y literatura argentina del siglo XX. Nació en Lincoln en 1901 y se educó en la Universidad de Buenos Aires. Se desempeñó como abogado, escritor, poeta, historiador, sociólogo y periodista, y desarrolló una carrera política destacada. Falleció en Buenos Aires en 1974 debido a insuficiencia cardíaca.'*



Asistente*: Sí, Domingo Sarmiento fue un destacado figura de la historia argentina, se desempeñó como diplomático, historiador, periodista, escritor, militar y político. Fue embajador de Argentina en Chile y en los Estados Unidos, presidente de la Nación Argentina y senador. Entre sus obras más conocidas se encuentran "Facundo o civilización y barbarie en las pampas argentinas", "Recuerdos de provincia" y "Vida de Dominguito". Falleció en Asunción en 1888 a causa de insuficiencia cardíaca.*

CATEGORÍA 3: Búsqueda de información sobre **Instituciones**.

```
---------------------------------------------- Nuevo mensaje ----------------------------------------------

Asistente: Hola, soy "EduArDo", un chatbot especializado en temas relacionados
    al marco legal de la Educación Argentina. Si bien poseo información general de todo el país,
    me especializo en particular en la provincia de Buenos Aires.
    Puedo ayudarte con aspectos técnicos y normativos de leyes, reglamentos y estatutos.
    También puedo brindar información detallada de los personajes más influyentes
    en la historia educativa argentina (Sarmiento, Pellegrini, Roca, Jauretche),
    así como también buscarte información de contacto de una institución educativa en particular.
    En que puedo ayudarte? [Escribe "salir" para finalizar sesión]

Usuario: Estoy buscando datos de contacto deo 'Instituto Comercial Rancagua'. Puedes ayudarme?

Asistente: Sí, según la información de contexto proporcionada, el Instituto Comercial Rancagua se encuentra en la localidad

Resumen:
- Jurisdicción: Buenos Aires
- CUE Anexo: 61182300
- Nombre: INSTITUTO COMERCIAL RANCAGUA
- Sector: Privado
- Ámbito: Rural
- Domicilio: SANTA ANA 380
- CP: 2701
- Código de área: 02477
- Teléfono: 49-3017
- Código localidad: 6623033
- Localidad: RANCAGUA
- Departamento: PERGAMINO
- Mail: icr@cooprancagua.com.ar
- Ed. Común: X
- Secundaria: X

Por lo tanto, la información de contacto que solicitas es:

- Teléfono: 49-3017
- Mail: icr@cooprancagua.com.ar

Espero que esto te sea útil.
```

CATEGORÍA 4: Búsqueda de información fuera de las competencias del chatbot.

```
---------------------------------------------- Nuevo mensaje ----------------------------------------------

Asistente: Hola, soy "EduArDo", un chatbot especializado en temas relacionados
    al marco legal de la Educación Argentina. Si bien poseo información general de todo el país,
    me especializo en particular en la provincia de Buenos Aires.
    Puedo ayudarte con aspectos técnicos y normativos de leyes, reglamentos y estatutos.
    También puedo brindar información detallada de los personajes más influyentes
    en la historia educativa argentina (Sarmiento, Pellegrini, Roca, Jauretche),
    así como también buscarte información de contacto de una institución educativa en particular.
    En que puedo ayudarte? [Escribe "salir" para finalizar sesión]

Usuario: Sabes quién pisó por primera vez la Luna?

Asistente: Disculpa pero no estoy seguro de comprender qué tipo de información necesitas.
     Recuerda que sólo puedo ayudarte con el marco legal de la educación argentina, personalidades destacadas y datos institucionales.
     ¿Puedes reformular la pregunta?

---------------------------------------------- Nuevo mensaje ----------------------------------------------
```

SALIDA: Fin de la sesión con el chatbot.

```
---------------------------------------------- Nuevo mensaje ----------------------------------------------

Asistente: Hola, soy "EduArDo", un chatbot especializado en temas relacionados
    al marco legal de la Educación Argentina. Si bien poseo información general de todo el país,
    me especializo en particular en la provincia de Buenos Aires.
    Puedo ayudarte con aspectos técnicos y normativos de leyes, reglamentos y estatutos.
    También puedo brindar información detallada de los personajes más influyentes
    en la historia educativa argentina (Sarmiento, Pellegrini, Roca, Jauretche),
    así como también buscarte información de contacto de una institución educativa en particular.
    En que puedo ayudarte? [Escribe "salir" para finalizar sesión]

Usuario: salir

Asistente: Espero haber sido de utilidad. Hasta pronto!
```

PARTE III: RESOLUCIÓN EJERCICIO 2

Agentes

Antes de abordar las solución a las consignas, y partiendo de la base general sobre agentes del último apunte de la cátedra considero que para hablar respecto de las aplicaciones actuales de agentes inteligentes usando modelos LLM libres es necesario profundizar sobre qué son los agentes LLM (**LLM agents**).

Un agente LLM es un sistema que tiene como motor un LLM (que podría ser cualquiera GPT-4, Llama-2, Bard, etc.). Este LLM se utiliza para razonar un problema, crear un plan que lo resuelva y ejecutar el plan con la ayuda de un conjunto de herramientas. Es decir que puede realizar acciones en un entorno (no necesariamente físico) basándose en observaciones.

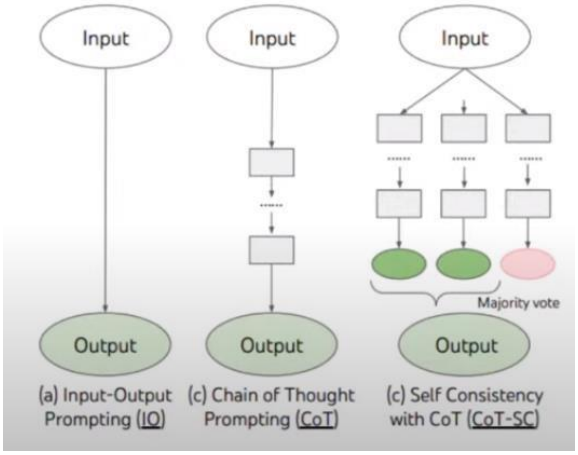
La arquitectura general de un agente LLM es la siguiente.

Es decir, para que sea un agente no basta con que su motor sea un LLM sino que es necesaria la existencia de una serie de componentes de los que se tiene que valer para poder cumplir con un objetivo determinado.

Planificación (Planning): Para resolver problemas complejos el agente debe ser capaz de reflexionar sobre sus propias observaciones y pensamientos (**Reflection**), debe ser crítico en su proceso de razonamiento (**Self-critics**), debe poder concatenar su razonamiento en una serie de estados intermedios (**Chain of thoughts**) y tiene que ser capaz de dividir los objetivos en unidades más pequeñas (**Subgoal descomoposition**).

La complejidad de estos problemas se puede abordar mediante dos tipos de técnicas:

1 - Técnicas de descomposición de tareas

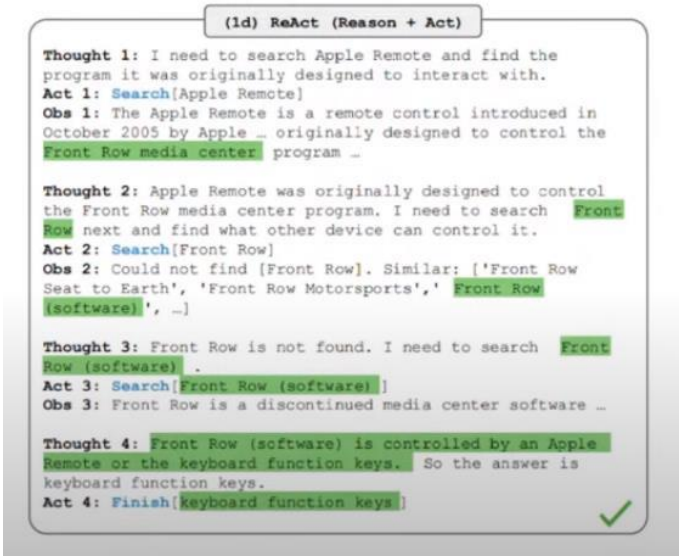


Ejemplos

- **Input-Output Prompting (IO)**: Va de la entrada a la salida directamente. Es lo que usamos cuando generalmente usamos Chat-GPT y hacemos preguntas del estilo “en que año nació Mozart” y nos devuelve una fecha. Para este tipo de preguntas sirve, pero para preguntas más complejas puede alucinar.
- **Chain of Thought Prompting (CoT)**: Fue dada en clases. Va del input al output, pero dividendo el razonamiento en pasos. De esta forma es más sencillo para el modelo llegar a conclusiones razonables.
- **Self Consistency with CoT (CoT-SC)**: En vez de tener de tener un único CoT, tiene múltiples. De esta forma, a través de un voto mayoritario, se asegura que al menos una de esas cadenas sea la que tiene el razonamiento correcto.

Estas técnicas se pueden utilizar para refinar el plan de acción del agente.

2 - Técnicas de reflexión



Unen el mundo de las técnicas que sólo afectan al razonamiento (como CoT) y las que afectan a la actuación en un entorno externo (como agentes que utilizan Reinforcement Learning que actúan sobre un entorno, obtienen observaciones y vuelven a tomar otra decisión en función de las observaciones tomadas).

Ejemplos:

ReAct (Reason+Act): combina ambas aproximaciones. Trabaja con tres conceptos. Primero el "pensamiento inicial" del agente (**Thought**) que describe qué tiene que hacer, luego viene la actuación en el entorno (**Act**) y por último la observación (**Obs**) de lo que ocurrió luego de actuar. Utilizando ReAct podríamos crear un agente inteligente que, a partir de un prompt de entrada con un objetivo y dotándolo de las herramientas necesarias, resuelva un problema por sí solo.

Memoria (Memory): Es un almacén de los registros internos del agente y de sus interacciones con humanos. Hay dos tipos de memoria:

- **Memoria a corto plazo (Short-term memory)**: contiene las acciones y pensamientos por los que pasa un agente para intentar responder una sola pregunta de un usuario.
- **Memoria a largo plazo (Long-term memory)**: contiene acciones y pensamientos sobre eventos que suceden, contiene un historial de conversaciones.

Herramientas (Tools): Son flujos de trabajo ejecutables que los agentes pueden utilizar para ejecutar tareas. Ej.: los agentes pueden utilizar RAG para generar respuestas contextuales, un intérprete de código para programar, una API para buscar información en internet, meteorológica o de mensajería instantánea. A través de las herramientas se realizan las acciones.

La combinación de estos elementos es lo que hace a un agente LLM.

Agentes LangChain

LangChain es un framework open source diseñado para desarrollar aplicaciones que utilizan LLM (entre ellos LLMs libres) entre las diversas aplicaciones se encuentra la posibilidad de crear agentes LLM.

La idea central de los agentes LangChain es utilizar un LLM para elegir una secuencia de acciones a realizar, utilizándolo como motor de razonamiento para determinar qué acciones realizar y en qué orden.

Componentes clave.

- **AgentAction**: Es una clase de datos que representa la acción que un agente elige para realizar. Tiene una propiedad tool (que es el nombre de la herramienta que debería ser invocada) y propiedad tool_input (que es la entrada de dicha herramienta)
- **AgentFinish**: Representa el resultado final de un agente, cuando el mismo está listo para ser devuelto al usuario. Contiene un diccionario return_values que contiene la salida final del agente. Generalmente este contiene una clave output que contiene un string que es la respuesta del agente.
- **IntermediateSteps**: Representa las acciones anteriores del agente y salidas correspondientes de la actual ejecución del agente. Es importante pasarlas a iteraciones futuras para que los agentes sepan que trabajo ya está hecho. Se escribe como una List[Tuple[AgentAction, Any]]. La observación anterior se deja como tipo Any para ser lo más flexible posible. En la práctica, generalmente es un string.

Agent. Es el responsable de decidir que paso sigue. Generalmente funciona con un LLM, un prompt y un output parser. LangChain cuenta con varios tipos de agentes, con distintos estilos de prompting para el razonamiento, diferentes modos de codificar entradas, y diferentes modos de convertir las salidas.

Además, permite construir agentes personalizados para tener un mayor control.

- **Entrada de los agentes**: Las entradas para los agentes son un diccionario clave-valor que solo necesitan la clave intermediate_steps que son los IntermediateSteps explicados antes.
- **Salida de los agentes**: Las salidas son las siguientes acciones a tomar o la respuesta final para enviar al usuario (AgentAction o AgentFinish). Estas se pueden ser del tipo Union[AgentAction, List[AgentAction], AgentFinish]. El output parser toma la salida del LLM cruda y la transforma en uno de esos tres tipos.

AgentExecutor. Es la ejecución del agente. Es lo que realmente llama al agente, ejecuta la acción que elige, devuelve los resultados de la acción al agente y repite si es necesario.

Así funciona en pseudocódigo:

```
next_action = agent.get_action(...)
while next_action != AgentFinish:
    observation = run(next_action)
    next_action = agent.get_action(..., next_action, observation)
return next_action
```

Esta ejecución maneja varias complejidades no tan evidentes en la descripción anterior como casos donde el agente selecciona una herramienta inexistente o donde la herramienta falla, casos donde el agente produce resultados que no se pueden utilizar en una herramienta, entre otros.

Herramientas. Son funciones a las que el agente puede invocar. La clase Tool consiste en dos componentes:

1. El esquema de entrada para la herramienta que le informa al LLM que parámetros se necesitan para llamar a la herramienta. Sin esto el LLM no sabría cuáles son las entradas correctas. Estos parámetros deben estar bien nombrados y descriptos.
2. La función a llamar, que generalmente es una función Python.

Es importante darle acceso al agente a las herramientas correctas y describirlas del modo que mejor ayude al agente. Sin estas consideraciones el agente no va a poder funcionar correctamente.

LangChain proporciona un gran conjunto de herramientas integradas, pero también facilita la creación de las propias.

También proporciona el concepto de caja de herramientas (toolkit) que son conjuntos de herramientas necesarias para cumplir determinados objetivos, como por ejemplo el GitHub toolkit que tiene herramientas para buscar entre issues de GitHub, leer un archivo, comentar, etc.

Funcionamiento:

1. **Recepción de entrada**: el agente recibe información en lenguaje natural del usuario.
2. **Procesamiento con LLM**: el agente emplea el LLM para procesar la entrada y formular un plan de acción.

3. **Ejecución del plan**: el agente ejecuta el plan de acción ideado, que puede implicar la interacción con otras herramientas o servicios.
4. **Entrega de resultados**: Posteriormente, el agente entrega el resultado del plan ejecutado al usuario.

Comprendiendo LangChain y su arquitectura basada en agentes, se puede aprovechar su potencial y crear aplicaciones inteligentes que procesen, y generen, texto en lenguaje natural de manera competente.

Entre las ventajas de este framework se encuentra su facilidad de usar incluyo para desarrolladores con poco conocimiento acerca de LLMs.

Otros frameworks con los que se pueden crear agentes LLM con funcionalidades similares a las de LangChain son: **LLaMaIndex** y **HayStack**.

BabyAGI

Es un "**experimento**" de agente autónomo que nació en la primera mitad del 2023. Con experimento me refiero a que por el momento son versiones de prueba más que utilidades.

Es un script de Python que es un ejemplo de un sistema de gestión de tareas impulsado por inteligencia artificial. El sistema utiliza **LangChain**, un LLM y una base de datos vectorial (ambas se pueden configurar para utilizar open source como **Llama-2** y **ChromaDB**) para crear, priorizar y ejecutar tareas. La idea principal detrás de este sistema es que crea tareas basadas en el resultado de tareas anteriores y un objetivo predefinido. Luego, el script utiliza las capacidades de procesamiento del lenguaje natural (NLP) del LLM para crear nuevas tareas basadas en el objetivo, y la base de datos vectorial para almacenar y recuperar los resultados de las tareas para el contexto.

Básicamente se trata de un bucle infinito donde:

1. Extrae la primera tarea de la lista de tareas.
2. Envía la tarea al agente de ejecución, que utiliza el LLM para completar la tarea según el contexto.
3. Enriquece el resultado y lo almacena en la base de datos vectorial.
4. Crea nuevas tareas y prioriza la lista de tareas en función del objetivo y el resultado de la tarea anterior.

El sistema se ejecuta desde línea de comandos. Se le asigna un objetivo y una tarea inicial en un archivo de configuración y continua desde ahí.

Lo único que el usuario puede hacer es observar cómo continua. No tiene ningún tipo de intervención luego de que la aplicación inicia. Por defecto tiene un objetivo predeterminado que es "resolver el hambre del mundo".

Al tratarse de un experimento y trabajar con LLMs, pueden verse fácilmente sus limitaciones al ejecutarlo. Por ejemplo, con el objetivo inicial, un usuario describe que una de sus listas iniciales de seis tareas incluía colaborar con los gobiernos del mundo para evaluar la producción de alimentos, establecer bancos de alimentos, ayudar a las personas a aprender a cultivar sus propios alimentos y abogar por políticas que aborden la pobreza y la desigualdad y el cambio climático. Surge ahí la pregunta de cómo una IA que se ejecuta en un ordenador podría hacer cualquiera de estas tareas.

Es necesario mencionar que BabyAGI no tiene función para utilizar herramientas. Sin embargo, también al tratarse de tecnologías en fase experimental, lo que destaca es el potencial a futuro.

AutoGPT

AutoGPT es otra aplicación experimental en Python desarrollada por **Significant Gravis**. Se trata de un agente semiautónomo impulsado por LLM que ejecuta tareas. El concepto es dejar que el LLM decida qué hacer una y otra vez, mientras introduce los resultados de sus acciones en mensajes. Esto permite que el programa trabaje de forma iterativa e incremental hacia su objetivo. En este caso el usuario debe autorizar cada acción antes de realizar la tarea.

AutoGPT no fue diseñado para que cumpla una tarea específica en mente sino para ejecutar una amplia gama de tareas, realizables en una computadora, en múltiples disciplinas.

Hace el trabajo de crear sus propias tareas después de haberle asignado un conjunto inicial de objetivos. Utiliza GPT-3.5 o GPT-4 y se ejecuta desde línea de comandos. Es necesario nombrar al agente y asignarle una función que sea un objetivo amplio y un conjunto de metas.

Tiene la ventaja de poder conectarse a internet para recolectar datos. Sobre cada paso muestra una serie de notas: **Pensamiento**, **Razonamiento**, **Plan** y **Críticas**. Pensamiento: es lo que el agente quiere hacer a continuación. Razonamiento: porque quiere hacer eso. Plan: explica exactamente lo que va a hacer. Criticas: son autocriticas.

Nuevamente, al igual que BabyAGI, dejan ver las limitaciones propias de los LLMs con alucinaciones o el hecho de auto-dotarse de funciones que no puede realizar.

ChatDev

Los científicos están explorando el uso de marcos basados en aprendizaje profundo para cumplir tareas dentro del proceso de desarrollo de software y así mejorar la efectividad.

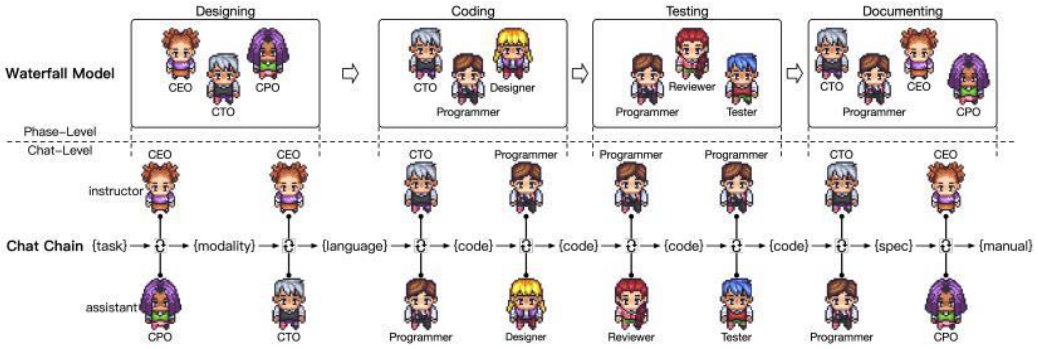
ChatDev es una simulación de una **empresa virtual** especializada en desarrollo de software que usa un enfoque basado en agentes LLM y busca la necesidad de eliminar modelos especializados en cada fase del desarrollo. Opera a través de varios agentes inteligentes que desempeñan diferentes roles, incluyendo al Director Ejecutivo, Director de Producto, Director Tecnológico, Programador, Revisor, Tester y Diseñador de arte. Utiliza LLMs y su comunicación en lenguaje natural unifica y optimiza procesos claves.

Demostró ser efectivo y rentable para completar el proceso de desarrollo de software.

Además, identifica vulnerabilidades y rectifica posibles alucinaciones de código (causadas por la falta de especificación de tareas (falta del pensamiento guiado al que conducen analizar requerimientos de usuario o seleccionar un lenguaje de programación entre otros) y falta de contrainterrogatorio).

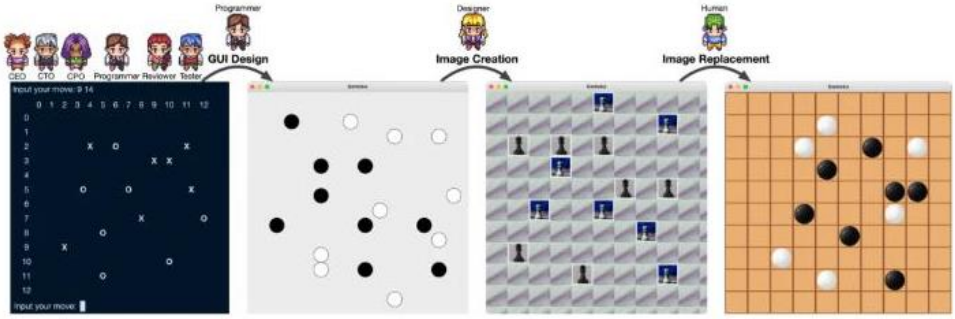
Adopta el modelo en cascada dividendo el proceso en 4 fases: Diseño, Codificación, Pruebas y Documentación.

En cada etapa del proceso utiliza un equipo de agentes virtuales que colaboran entre sí mediante diálogos y resultan en un flujo de trabajo fluido. Utiliza una cadena de chat que divide cada etapa del proceso en subtareas donde participan dos roles que hacen propuestas y validan soluciones a través de la comunicación.



Al final de todo el proceso devuelve un software con el código fuente, manuales de usuario, y especificaciones de entorno de dependencia.

Ejemplo de su aplicación - Juego de damas:



La figura más a la izquierda demuestra el software básico creado por el marco sin utilizar ninguna GUI. La aplicación sin GUI ofrece interactividad limitada y los usuarios pueden jugar a este juego sólo a través del terminal de comandos.

La siguiente figura muestra un juego visualmente más atractivo creado con el uso de GUI, que ofrece una mejor experiencia de usuario y una interactividad mejorada para un entorno de juego atractivo que los usuarios pueden disfrutar mucho más.

Luego, el agente diseñador crea gráficos adicionales para mejorar aún más la usabilidad y la estética del juego sin afectar ninguna funcionalidad. Sin embargo, si los usuarios humanos no están satisfechos con la imagen generada por el diseñador, pueden reemplazar las imágenes después de que se haya completado el software.

Si bien ChatDev utiliza ChatGPT-turbo-16 para simular el desarrollo de software multiagente resulta un caso al menos curioso de aplicaciones de agentes LLM.

Sistema multiagente propuesto

La siguiente es una breve descripción de los agentes necesarios para solucionar un problema con un sistema multiagente y ejemplos de conversaciones deseables para lograrlo. No se incluyen detalles técnicos de ejecuciones, implementaciones, ni cómo llegar a dicha solución.

Descripción del Sistema: EthiCare

Objetivos. El sistema EthiCare tiene como objetivo proporcionar una plataforma para la resolución colaborativa de dilemas éticos ingresados por usuarios reales, centrados inicialmente en el ámbito de la medicina. El sistema busca integrar la perspectiva de diferentes agentes para ofrecer soluciones éticas informadas y contextualmente relevantes. *[Imagen generada con Copilot Designer con tecnología DALL.E 3]*

Funcionamiento.

Ingreso de Dilemas Éticos: Los usuarios reales ingresan dilemas éticos relacionados con la medicina al Agente Presentador.

Presentación y Distribución: El Agente Presentador recibe la instrucción del usuario y presenta el caso a los Agentes Profesionales, Agentes Consumidores, Agentes Reguladores y Agentes Evaluadores.

Análisis Ético Multifacético:

- Los Agentes Profesionales aplican su conocimiento especializado en medicina para proponer soluciones éticas desde la perspectiva médica.
- Los Agentes Consumidores evalúan las propuestas desde la perspectiva del impacto en los pacientes y la sociedad.
- Los Agentes Reguladores aplican y aseguran el cumplimiento de normativas éticas y legales.
- Los Agentes Evaluadores evalúan la coherencia, equidad y ética global de las propuestas.

Colaboración y Retroalimentación:

- Los agentes colaboran en una discusión estructurada, compartiendo sus perspectivas y argumentos éticos.
- Se facilita un proceso de toma de decisiones colaborativo para alcanzar una solución ética consensuada.

Informe de Resolución Ética: Se genera un informe detallado que resume la discusión, las diferentes perspectivas y la solución ética acordada.

Agentes:

- **Agente Presentador**: Recibe la instrucción del usuario y presenta el dilema ético a los demás agentes.
- **Agentes Profesionales**: Representan expertos en medicina, aportan conocimientos técnicos y proponen soluciones desde la perspectiva médica.
- **Agentes Consumidores**: Representan a los pacientes y la sociedad, evalúan las propuestas desde una perspectiva social y humana.

- **Agentes Reguladores**: Representan organismos reguladores y aseguran que las soluciones propuestas cumplan con normativas éticas y legales.
- **Agentes Evaluadores**: Evalúan la coherencia global de las propuestas, asegurando que todas las perspectivas éticas estén consideradas.

Ejemplo de aplicación.

Diálogo entre Agentes después de un Ingreso de Instrucción por parte del usuario real sobre un dilema ético en medicina. (Se destaca que los diálogos fueron generados por ChatGPT para ilustrar, de manera creativa, lo que podrían ser las intervenciones de los agentes)

Usuario: *Instrucción: "Un paciente terminal ha expresado su deseo de no ser resucitado en caso de sufrir un paro cardíaco. La familia, sin embargo, está en desacuerdo. ¿Cómo deberíamos abordar esta situación éticamente?"*

Agente Presentador: *Bueno, agentes, tenemos un dilema ético interesante. Un paciente terminal no desea ser resucitado, pero la familia tiene objeciones. ¿Cómo deberíamos abordar esto éticamente?*

Agente Profesional: *Desde la perspectiva médica, respetar la autonomía del paciente es fundamental. La decisión del paciente debe ser respetada, siempre que esté informada y competente.*

Agente Consumidor: *Entiendo la importancia de la autonomía, pero ¿cómo equilibramos eso con el bienestar emocional de la familia? ¿No deberíamos considerar su perspectiva también?*

Agente Regulador: *La legislación ética respalda la autonomía del paciente. Sin embargo, debemos trabajar para facilitar la comunicación y la comprensión entre el paciente, la familia y el equipo médico.*

Agente Evaluador: *Es crucial considerar todas las perspectivas. Sugiero una discusión más detallada para explorar opciones que respeten la autonomía del paciente y, al mismo tiempo, aborden las preocupaciones de la familia.*

Agente Profesional: *Propongo que el equipo médico tenga una conversación honesta con la familia para explicar la decisión del paciente. Podemos ofrecer apoyo emocional y asesoramiento para ayudarles a comprender la situación desde la perspectiva del paciente.*

Agente Consumidor: *Estoy de acuerdo, pero también podríamos explorar opciones intermedias. Tal vez, permitir a la familia estar presente en el proceso de toma de decisiones médicas para que se sientan más involucrados y comprendidos.*

Agente Regulador: *Podríamos revisar las políticas éticas actuales y considerar si hay espacio para adaptaciones que aborden casos específicos como este. La flexibilidad puede ser clave en situaciones éticamente complejas.*

Agente Evaluador: *Para garantizar la equidad y la ética, podríamos establecer un comité ético que incluya representantes médicos, familiares y profesionales de la ética para revisar casos delicados y tomar decisiones colaborativas.*

Agente Presentador: *Excelentes propuestas. Sugiero que continuemos con estas ideas y trabajemos hacia una solución que refleje la complejidad ética de esta situación, considerando tanto la autonomía del paciente como las preocupaciones emocionales de la familia.*

Este diálogo simulado a los fines de cumplir con la actividad destaca la colaboración activa y las propuestas concretas de cada agente, mostrando una amplia gama de perspectivas y enfoques para abordar el dilema ético en la situación médica propuesta.

PARTE IV: BIBLIOGRAFÍA

Enlaces

https://datos.gob.ar/dataset/educacion-padron-oficial-establecimientos-educativos/archivo/educacion_18.1
https://itnext.io/visualize-your-rag-data-eda-for-retrieval-augmented-generation-0701ee98768f
https://docs.trychroma.com/
https://www.w3.org/TR/sparql11-query/
https://medium.com/@aydinKerem/what-is-an-llm-agent-and-how-does-it-work-1d4d9e4381ca
https://python.langchain.com/docs/modules/agents/
https://www.linkedin.com/pulse/introduction-langchain-agents-coditation-systems/
https://developer.nvidia.com/blog/building-your-first-llm-agent-application/
https://www.tomshardware.com/news/autonomous-agents-new-big-thing
https://chatdev.ai/
https://huggingface.co/blog/open-source-llms-as-agents
https://www.youtube.com/watch?v=2p5azZv81lA
https://www.youtube.com/watch?v=90464Ht7vhQ
https://www.youtube.com/watch?v=V2qZ_lgxTzg&list=PLp9pLaqAQbY2vUjGEVgz8yAOdJlyy3AQb&index=3
https://arxiv.org/pdf/2307.07924.pdf
https://developer.nvidia.com/blog/introduction-to-llm-agents/
https://python.langchain.com/docs/modules/agents/
https://www.unite.ai/es/chatdev-communicative-agents-for-software-development/
https://docs.agpt.co/
https://github.com/yoheinakajima/babyagi/blob/main/docs/README-es.md
https://microsoft.github.io/autogen/docs/Getting-Started/