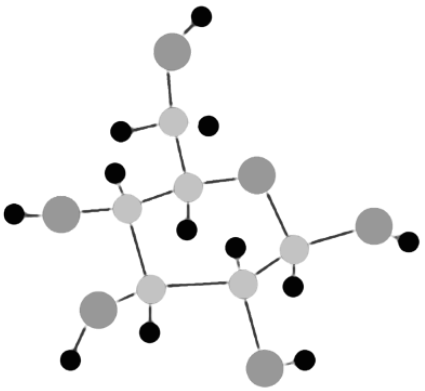
Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Agentes Expertos



TUIA - IA 4.2

Procesamiento de Lenguaje Natural

Trabajo Práctico Integrador

Comisión: Única

Docentes:

- *Juan Pablo Manson*
- *Alan Geary*
- *Andrea Leon Cavallo*
- *Ariel D'Alessandro*

Fecha: 20/02/2023

Grupo:

- *Giampaoli Fabio*

Resumen

Este proyecto consiste en la creación de un modelo de lenguaje como asistente para conversar con lenguaje natural sobre algún dominio de interés a elección, y la investigación de la posibilidad de convertirlo en un sistema multiagente que pueda conectar con diversas herramientas.

En este caso, mi interés es que el agente sea experto en Genexus. Genexus es un entorno de desarrollo de software que mediante un lenguaje de programación simplificado e integraciones con otras herramientas, se puede generar código fuente para compilar y desplegar aplicaciones.

El objetivo es que el agente pueda tener a disposición la documentación oficial de Genexus para que pueda mantener una conversación al respecto como si Genexus fuera parte de su fuente de su conocimiento, para finalmente darle al usuario que interactúa la sensación de que el agente puede ayudar a resolver dudas y problemas relacionados.

El modelo conversacional alimentará su contexto mediante diferentes fuentes de datos. Las tres fuentes principales que tomara serán:

- **Wiki de Genexus**: mediante extracción del texto de las páginas de la documentación oficial mediante técnicas de scraping.
- **Cursos de Genexus**: Genexus posee videos que enseñan desde los conceptos más básicos hasta conceptos y metodologías avanzadas. Los videos son acompañados por documentos pdf como transcripciones de los videos.
- **WikiData**: Es una base de conocimiento de grafos abierta de dominio general que dotará al modelo de conocimiento general de cualquier tópico en general.

Enunciado - Ejercicio 1

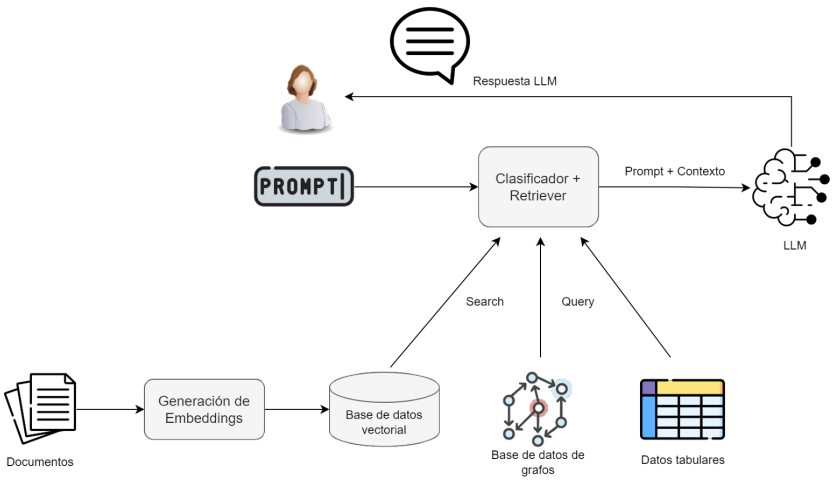
Crear un chatbot experto en un tema a elección, usando la técnica RAG (Retrieval Augmented Generation). Como fuentes de conocimiento se utilizarán al menos las siguientes fuentes:

- 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.

A continuación se muestra una figura conceptual de la arquitectura esperada de este agente:



Resolución - Ejercicio 1

Entorno

Si bien los entornos de Google Colaboratory se inicializan con gran cantidad de librerías preinstaladas para tarea de procesamiento de lenguaje, existen otras dependencias a resolver del entorno. Por lo que el primer paso es la preparación del mismo realizando todas las instalaciones, descargas e importaciones de las librerías en la primera sección con el fin organizar las dependencias al inicio del proyecto.

Entre ellas se encuentran librerías como *llama_index* que permiten construir bases de datos de búsqueda y comparación de texto, librerías de *transformers* que permite descargar y utilizar modelos de lenguajes pre entrenados, *langchain*, *nltk,* entre otras para procesamiento de texto y depuración. *qwikidata* para realizar consultas a la base de conocimiento de WikiData, entre otras.

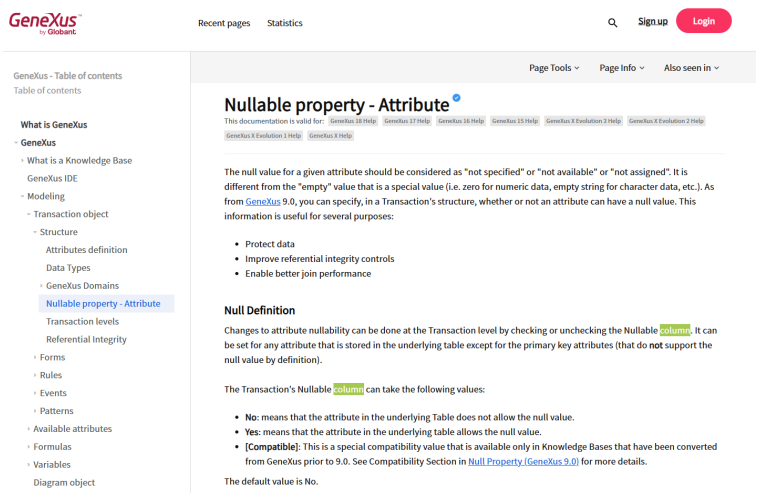
Fuentes

Para el objetivo de alimentar modelos de lenguaje con información de cómo funciona Genexus se recurre a tres fuentes distintas que proveen información de las capacidades y sus características como herramienta de desarrollo. Las fuentes son:

- La documentación oficial de su sitio oficial: La wiki de Genexus.

  La wiki de Genexus se divide en tópicos. Cada tópico tiene un menú asociado a las diferentes páginas de ese tópico en particular. Por lo que al situarse en los menús principales de cada tópico de interés (seleccionados manualmente), podemos extraer de cada menú los links a cada una de las páginas de ese tópico.

  Así luce la estructura típica de un tópico de Genexus:



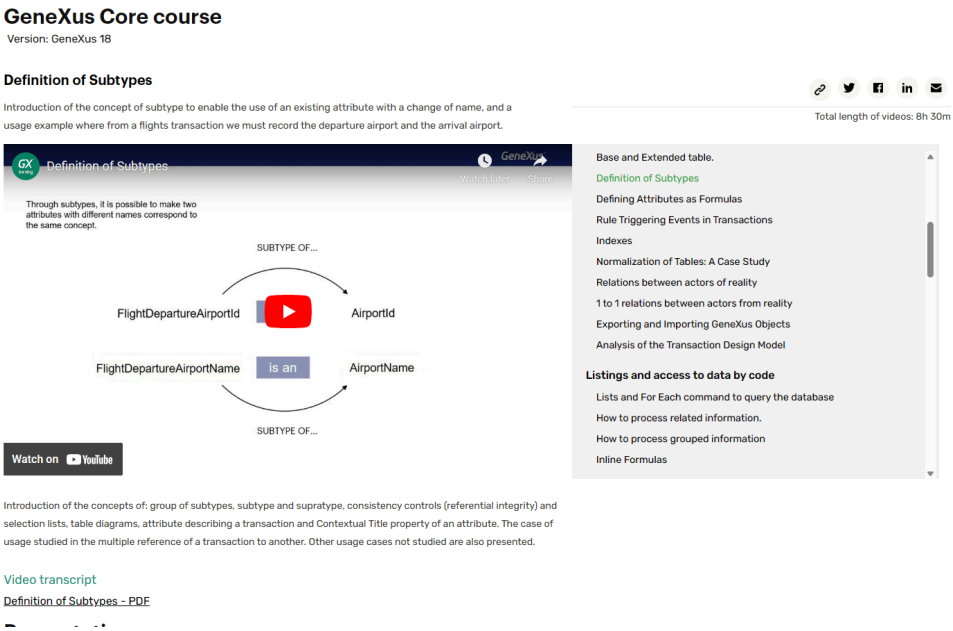
Se crea una clase que leé una página de la documentación de los menús extraídos, y extrae de ella el contenedor principal del texto que es un tag con un id particular.

El contenido se separa en párrafos y se limpia el formato del texto, grabando en un dataframe cada párrafo de la página como texto, referenciado a la página y tópico al que pertenece. Muestra del dataframe:

Source	Title	URL	Section	Content
Version 18	Total Experience	https://wiki.genexus…	1	GeneXus 18 promotes Total Experience (https://...
Version 18	Total Experience	https://wiki.genexus…	2	Import from Figma\nTo get the best user experi...

- Los cursos oficiales de [Genexus Training]().

Genexus posee una página donde ofrece cursos audiovisuales en diferentes niveles y para diferentes objetivos. Un curso tiene esta estructura genérica:



Puede notar que cada video o elemento del menú contiene una sección de particular interés, que es el acceso a la versión PDF de la clase. Por lo que, de cada elemento del menú de cada curso, se extrae dicho link al pdf de la clase.

De un documento podemos extraer su contenido. Para facilitar la lectura se decide distinguir el documento por página, y los párrafos que corresponden a dicha página. Ejemplo de un documento:

```
Page number: 2
GeneXus is an intelligent platform that simplifies software development, automating
everything that can be automated.
It is LOW-CODE, which means that the least possible code is written,
most of it in a declarative way. Then, through artificial intelligence techniques
GeneXus automatically generates source code, minimizing the times required for
application development, evolution, and maintenance.


Page number: 3
GeneXus allows building different types of applications, from the green screen ones
for Cobol or RPG systems, to state-of-the-art Web applications in Angular framework,
native code applications for mobile devices such as watches, cell phones, tablets
or Apple TV, as well as Artificial Intelligence applications, Chatbots, business process
modeling or the Internet of Things.
```

De cada pdf se genera un archivo de texto plano con el formato anterior y se almacena para facilidad de acceso a estos documentos.

- Base de conocimiento de WikiData.

  Esta fuente no tiene el fin de nutrir con conocimiento técnico o específico de la herramienta, sino brindar contexto para responder preguntas más genéricas, de modo que el modelo final pueda responder de manera informada con información extraída de una base de conocimiento en constante modificación.

  WikiData es una base de datos de grafos donde podemos consultar información relacionada a una entidad. Es decir, de una entidad que exista en la base de conocimiento, se puede obtener sus objetos relacionados y cómo se relacionan con ellos.

  Para extraer información de esta fuente, se construyen dos funciones: Una para detectar entidades principales de un texto, y la segunda para extraer de WikiData todas las relaciones de las entidades detectadas. Por ejemplo, si una consulta determinar que *GeneXus* es la entidad principal, retorna desde WikiData la información relacionada a la misma en este formato:

```
genexus:
        instance of: programming language
        instance of: application framework
        instance of: declarative programming language
        developer: ARTech Consultores SRL
        programmed in: Prolog
        inception: 1988-01-01T00:00:00Z
        Freebase ID: /m/06vqgk
        official website: https://www.genexus.com
        Quora topic ID: GeneXus
        programming paradigm: declarative programming
```

Almacenamiento

Para los datos tabulares utilizaremos una base de datos vectorial FAISS con el fin de almacenar en forma numérica los párrafos de las páginas de la Wiki de Genexus para su posterior recuperación en base a la cercanía de los textos a ciertas oraciones de ingreso.

Notar que el uso de FAISS como base de datos vectorial en lugar de ChromaDB es debido a errores de dependencia a la hora de instalar en el entorno la librería de ChromaDB.

Para los documentos se utiliza un enfoque diferente para realizar búsquedas de textos similares a una consulta inicial. Para ello se utiliza un forma de generar nodos de texto que representan en espacios numéricos multidimensionales los textos para ser buscados y comprados rápidamente.

Se construyen tres funciones que simplifican la consulta de información relacionada de las diferentes fuentes. Ejemplo de contenido similar de las tres fuentes a una misma consulta:

INPUT
find function on Automatic data provider

TABULARES
Source: General Development, Title: Find
Content: {"syntax find (<aggregateexpression>, <aggregatecondition>, <defaultvalue>) [<triggeringcondition>]; where: <aggregateexpression>: expression whose resultant value....

DOCUMENTOS
file_path: llamaindex_data/Advanced_Data_Providers._Language_and_Some_Examples.txt
Here is an example where the Data Provider returns a collection
of Business Components whose data is obtained from another table…

GRAFOS
automatic:
instance of: technology
subclass of: information and communications technology
Commons category: Automation
opposite of: manual operation
…

Modelo de clasificación

El modelo de clasificación tiene la intención de que determine la complejidad de la pregunta del usuario, con el fin de no recuperar información cercana a la misma de todas las fuentes a la vez, si no solo de una acorde a su contenido. Se determina:

- Relaciones de WikiData: corresponde a preguntas genéricas.
- Páginas de la Wiki: corresponde a preguntas de dificultad media.
- Documentos de los cursos: corresponde a preguntas más complejas.

Para determinar la clase de una consulta, se utiliza el modelo de lenguaje *Zephyr* con un prompt y configuración muy particular que nos ayudará con esta función. En la instrucción y rol que debe cumplir el modelo, se le indica algo como esto:

```
quest_to_model = f"""<|system|>
You are a query classifier that does not response queries, but only classify queries on these categories:
 0 - general knowledge (Generic questions about GeneXus or other things)
 1 - rules, funcions and sysntax (Basic definitions in GeneXus to develop logic and object)
 2 - development and techniques (Integration among external technologies, and high level methodologies)
<|user|> +"What is the programming paradigm of Genexus?"
<|assistant|> 0
<|user|> +"Explain me the use of each parameter in the Msg function"
<|assistant|> 1
<|user|> +"Teach me how to intregrate a remote github repository in my KB"
<|assistant|> 2
…"""
```

En esencia, se le enseña al modelo a clasificar las preguntas en base a ejemplos de cómo debe ser la clase y la respuesta dadas las consultas posibles del usuario. Se nota un comportamiento muy efectivo a la hora de clasificar con este método. Ejemplo:

```
user_query = 'What is the most recommend way to write efficent and cleaned programas?'
>>> 0
```

Una vez obtenida la clase a la que pertenece la consulta, podemos obtener la información de contexto similar a la misma desde la fuente establecida. Por lo que se construye una función que determina la clase y retorna el contexto de la debida fuente. Se trata de una llamada al modelo anterior una serie de condicionales. Se puede probar la efectividad en una prueba como esta:

```
queries = ['Tell me about the parameters the find formula accept',
           'how to automate your Calendar to save time',
            'Design the arquitecture of an application that runs throug microservices and
            orquested by containered programs running in a cloud server',
           ]

for query in queries:
    context = retrive_from_source(query)
```

Esto retorna el texto relacionado a cada consulta de las tres fuentes esperadas según la complejidad de las consultas.

Modelo de Lenguaje

En la instancia anterior hemos definido funciones que son de utilidad para enviar consultas y obtener respuestas mediante API al modelo Zephyr. Este mismo mecanismo será utilizado para el modelo conversacional. En este caso solo se cambiara la instrucción del comportamiento esperado del modelo, esta vez un instrucción mucho más personalizable a cada consulta:

```
quest_to_model = f"""
  <|system|> {bot_role}
  <|user|> Context: {context}
           Question: {user_query}
   <|assistant|>"""
```

Implementaciones

- **Contexto:** Un contexto voluminoso para un modelo puede no resultar beneficioso. La idea es separar los documentos cercanos en partes, y quedarse con las partes particulares de estos documentos que más se parezcan a la pregunta inicial del usuario.

  Para determinar cuáles son las partes de los documentos más cercanas al contexto de la pregunta, se llevan tanto las oraciones como la pregunta a un espacio vectorial que permitirá determinar qué tan cercanas son las partes o párrafos del texto a de la pregunta, y con ello podemos quedarnos con un top de las mejores partes para utilizar como contexto en lugar de un documento entero o una serie muy larga de párrafos.

- **Memoria:** Otra implementación es la capacidad de recordar mensajes previos. La idea es que el modelo pueda mantener la coherencia de la conversación respecto a las consultas y respuestas previas. Por ellos persisten dos memorias. Una para las respuestas que el modelo genera a lo largo de la conversación, y otra para las preguntas del usuario.

  Como el tamaño del contexto es limitado, no consideramos todo el historial de la conversación como contexto del modelo, sino solo las preguntas y respuestas más relevantes de la conversación para la consulta de la iteración actual.

- **Autocompletado:** Muchas veces el modelo de lenguaje está limitado por la cantidad de tokens que puede generar. Esto da lugar a inconsistencias donde genera textos que no tiene un final coherente por la generación de la respuesta se corta abruptamente.

  Para darle al usuario la sensación de respuestas completas en todo momento, se genera una técnica de autocompletado de la respuesta del modelo para terminar de formar la oración o párrafo. Esto, nuevamente mediante el uso de un modelo de lenguaje con una instrucción particular que le enseña a realizar esta tarea.

```
text_to_model = f"""
  <|system|> You are a language model that completes the given incomplete text
             with few words as possible. You should not regenerate the entire text, but
             only generate the missing part to make the text complete.
  <|user|>   By moving to the cloud, businesses can reduce their energy
             consumption  and carbon footprint by up to 90%. Rather than having
             in-house servers and software, businesses can use cloud-based services
             to access the same applications
  <|assistant|> and data from any computer.
…"""
```

- **Traducción**: Debido a que la documentación de Genexus está en inglés y no se puede obtener directamente en español, se opta por añadir un paso más a la hora de interactuar, que es la traducción automática del texto tanto ingresado como retornado, debido a en la interfaz el usuario debe poder conversar en español, pero el modelo de fondo trabaja en inglés.

Finalmente, luego de todas las implementaciones individuales, se construye una única llamada que reúne toda esta lógica de manera ordenada con el fin de simplificar la interacción final del usuario con el modelo.

El siguiente es un pseudocódigo de python que simula la construcción de la función que implementa todas las partes del agente.

```
# incializo la memoria vacia

def query_to_model(user_query, assitant_role):

   # traduce al ingles la pregunta del usuario para usar en busquedas

   # obtiene la informacion de contexto desde las fuentes
   # obtiene la informacion de contexto desde el historial

   # formato del contexto para que Zhepyr interprete los origines
   final_context = 'This is the associated documentation:\n' +
                   relevant_source_context + '\n\n'
   final_context += 'This is a related response in the conversation:\n' +
                   relevant_from_model + '\n\n'
   final_context += 'And this is a related query in the conversation:\n' +
                   relevant_from_user

   # obtine respuesta del modelo
   model_response = get_output(context_instruction(user_query_en…
   # asegura la completitud de la respuesta
   completed_content = automatic_autocompletation(model_response…

   # traduce la respuesta al español

   # guarda en memoria la pregunta y respuesta en ingles

   return model_answer_es
```

Una segunda función que simplifica aún más el inicio de la conversión con el modelo, inicializa la conversación con la memoria vacía.

Ejemplo de conversación recortada entre un usuario y el modelo.

start_genexus_model()

>>> **[Usuario]:** Es posible usar una formula find en la definicion de un data provider…

>>> **[Modelo]:** Sí, es posible utilizar una fórmula de búsqueda en la definición…

>>> **[Usuario]:** Si pongo esta formula de busqueda en un procedimiento…

>>> **[Modelo]:** No, la fórmula de búsqueda proporcionada recupera solo las líneas…

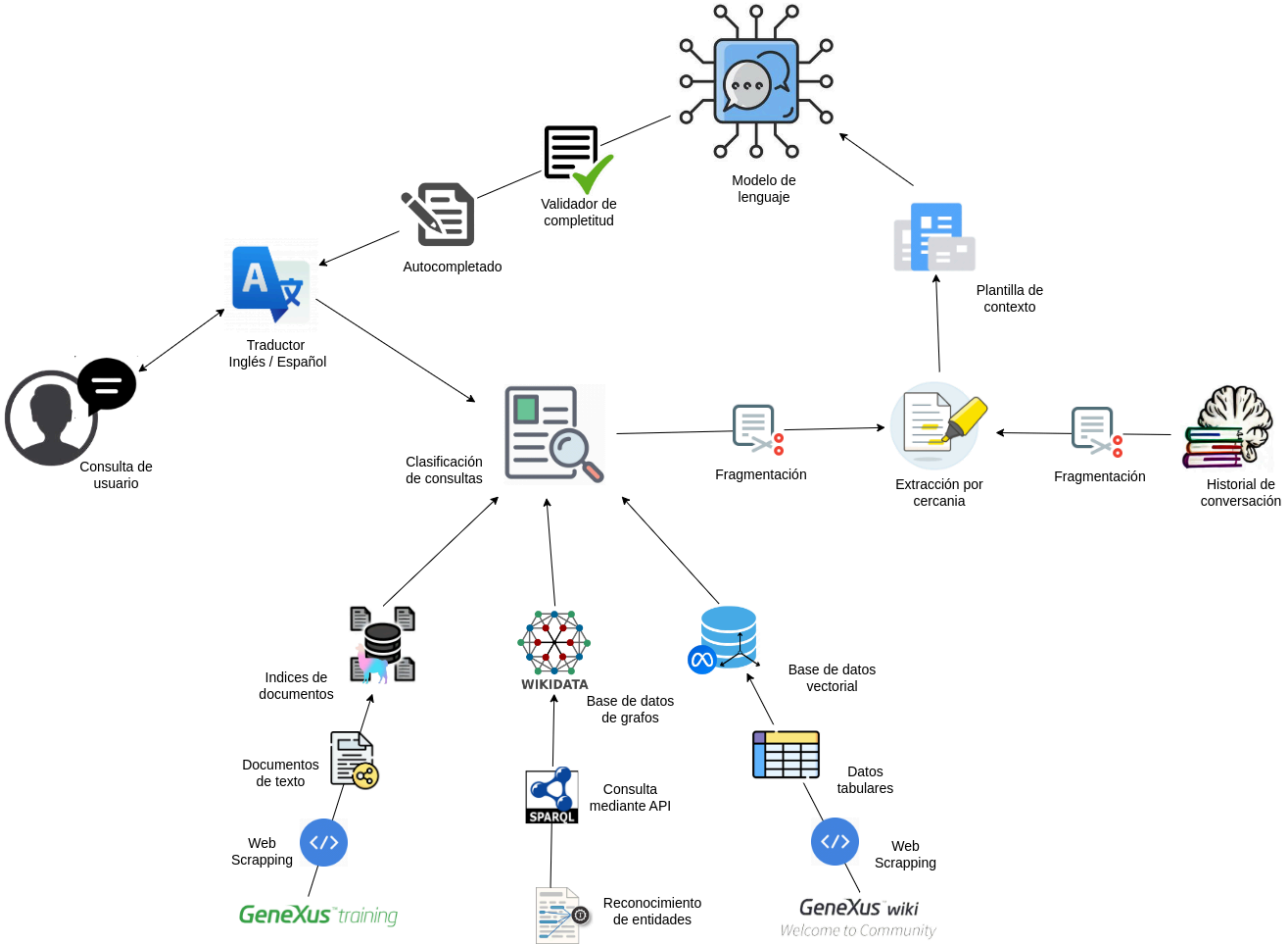
>>> **[Usuario]:** Esta bien que no retorne una coleccion de pedidos, ya que…

>>> **[Modelo]:** Sí, puede utilizar una fórmula de búsqueda en la definición de…

Se puede analizar la conversación en profundidad en el respectivo cuaderno de Google colab adjunto a este proyecto, y determinar que el modelo es capaz de llevar una conversación con el usuario de forma coherente. Incluso pudiendo generar código de programación y demostrar un cierto conocimiento y dominio sobre el contexto de GeneXus.

La implementación de una memoria sobre el historial de la conversación demuestra ser de utilidad por el que modelo demuestra ‘recordar’ tanto lo que él mismo ha respondido antes como lo que el usuario ha preguntado previamente, dando al usuario final una sensación de conversión más coherente a los largo de la misma.

A continuación se representa esquemáticamente la arquitectura del agente desarrollado:



Enunciado - Ejercicio 2

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.

Resolución - Ejercicio 2

Los agentes inteligentes son modelos que pueden percibir su entorno, razonar sobre él y actuar de forma autónoma o semiautónoma para lograr sus objetivos. Los modelos LLM son sistemas de inteligencia artificial que pueden generar y comprender lenguaje natural a gran escala, utilizando una gran cantidad de datos textuales como entrenamiento.

Por lo que un agente basado en LLM tiene gran potencial para resolver problemas reales de cualquier tipo si se lo ubica en un contexto adecuado. Actualmente existe gran variedad de modelos tanto privados como de código libre.

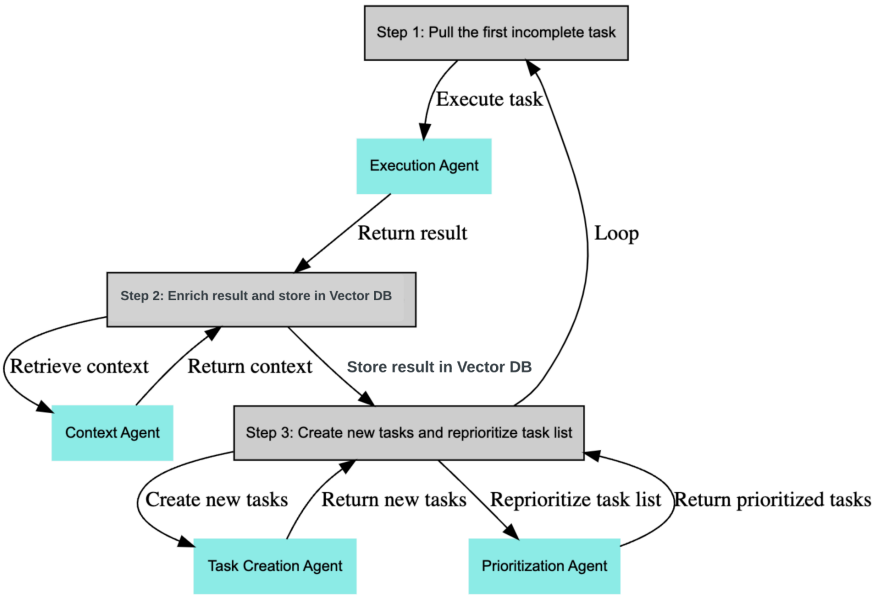
Ejemplos de agentes inteligentes privados son Chat GPT o Bard. Ambos modelos son potencialmente útiles y ampliamente utilizados a lo largo del mundo, y tienen la habilidad de responder a situaciones complejas o simples que presentan los usuarios en diversos contextos usando solo lenguaje natural humano hablado o escrito.

Últimas actualizaciones de modelos como estos han sido las integraciones con otros sistemas para que estos agentes inteligentes orquesta un sistema donde interactúan múltiples agentes para resolver problemas. Una de ellas es la carga y el análisis de archivos, que permite al usuario subir documentos o imágenes y obtener información relevante sobre ellos, como resúmenes, categorías o etiquetas. Otra es la integración con otras herramientas externas, como Navegación, Análisis avanzado de datos y sistemas de generación y reconocimiento de imágenes.

Sin embargo, estos sistemas multiagente por el momento solo están disponibles para aquellos usuarios que deseen abonar por el servicio. Es decir, no son sistemas de libre acceso, y mucho menos, de código libre para que cualquiera que desee pueda interpretar su funcionamiento.

Actualmente existen proyectos públicos de código abierto que intentan lograr sistemas multiagentes como los sistemas anteriores. Un ejemplo de ello es AutoGPT. Es un proyecto de código libre en Github similar a Chat GPT como modelo de lenguaje para resolver problemas genéricos, pero combinado con técnicas de auto-prompting para aumentar la autonomía del modelo.
Últimas actualizaciones del modelo han sido por ejemplo integraciones con 11 Labs, un sistema de IA que a partir del texto genera una voz sintetizada para que puedas "hablar" con Auto-GPT.

Otra alternativa muy interesante es el proyecto BabyAGI, un modelo publicado en Github que tiene la intención de ser un modelo de lenguaje natural interactivo que hace uso de diferentes agentes que interactúan entre sí para resolver problemas complejos. En su repositorio se obtiene una imagen explicativa como esta donde se abstrae el funcionamiento, coordinación e interacción entre los agentes del sistema:



En este proyecto se explica que la intención es que el modelo sea lo más autónomo posible, generando respuestas de acuerdo a la prioridad que le asigna un agente en el sistema a las tareas que crea otro agente, y luego generar el contexto y ejecutar la resolución con otros agentes especializados en dicha tarea.

El estado actual de este proyecto aún está en curso y en etapas tempranas, pero actualmente es posible ejecutar este sistema basado en otros modelos lenguajes mediante uso de API’s.

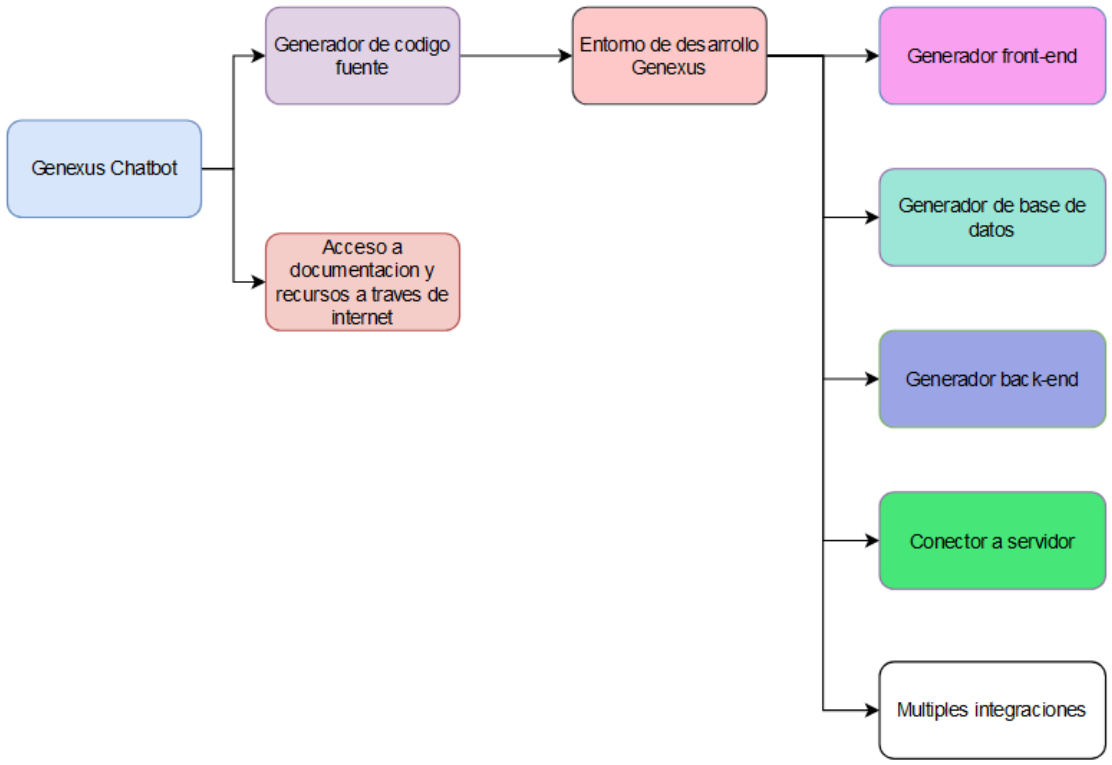
Un problema que se puede solucionar con un sistema multiagente en el contexto de este proyecto, puede ser la integración del chatbot con el generador de código de Genexus. Es decir, Genexus es un modelo que procesa versiones simplificadas de código en su propio lenguaje de programación para posteriormente con generadores de códigos configurados, se traslade esta lógica a un lenguaje de más bajo nivel (con respecto a lenguaje Genexus) como .Net o Java.

De esta forma se podría lograr un agente conectado a un entorno de desarrollo funcional que pueda permitir la ejecución y despliegue de funcionalidades que el usuario desee con prompts simplificados a solo su funcionamiento lógico a nivel de lenguaje natural, y así absorber aún más la complejidad del desarrollo de aplicaciones.

De hecho, sería de gran utilidad si se pudiera integrar este chatbot a un entorno de desarrollo ‘vivo’ donde el código generado por el chatbot conectado al generador de codigo de Genexus, pueda ser ejecutado y compilado en tiempo real, para desarrollar aplicaciones de forma mucho más interactiva.

De por sí Genexus abarca muchísimas integraciones de forma simplificada al desarrollador. Por lo que actividades como versionado del proyecto, seguridad, generación de interfaces de usuario, llamada a módulos externos con gran diversidad de funcionalidades no nativas, podrían ser accedidos desde la integración con Genexus, ya que Genexus com entorno sería el puente entre todas las integraciones externas y el chatbot.

El siguiente diagrama sería una representación de los agentes que interactúan en un sistema como el planteado.



Referencias

Integraciones Chat GPT:
https://blog.clickpanda.com/tecnologia/chat-gpt-y-sus-ultimas-actualizaciones/

Carga de archivos Chat GPT:
https://appmaster.io/es/news/nueva-actualizacion-beta-chatgpt-plus

Navegación y análisis Chat GPT:
https://www.elgrupoinformatico.com/utilidades/chatgpt-se-renueva-2023/

Integración de YouTube a Bard:
https://www.lanacion.com.ar/tecnologia/bard-el-chatgpt-de-google-ahora-va-a-mirar-videos-de-youtube-por-vos-y-armara-un-resumen-nid23112023/

Que es AutoGPT:
El siguiente salto de ChatGPT se llama Auto-GPT, genera código de forma "autónoma" y ya está aquí (xataka.com)

Repositorio Github AutoGPT:
GitHub - Significant-Gravitas/AutoGPT: AutoGPT is the vision of accessible AI for everyone, to use and to build on. Our mission is to provide the tools, so that you can focus on what matters.

Repositorio Github BabyAGI:
GitHub - yoheinakajima/babyagi

Funcionamiento BabyAGI:
BabyAGI: El revolucionario sistema de gestión de tareas impulsado por IA |2023 ✅ (gptpro.es)