CÓMO PROCESAR DATOS PROPIOS DE UNA ORGANIZACIÓN CON CHATGPT

Autores:
Lic. Esp. Mariano Campanello
Laureano Lorenzo

1. Presentación del Proyecto

El 30 de noviembre de 2022 con el lanzamiento público de ChatGPT comenzó una carrera vertiginosa de empresas relacionadas a la Inteligencia Artificial, específicamente en lo relacionado a los Modelos Grandes de Lenguaje, Large Language Model, como se denominan en inglés, o LLM como se los menciona en forma abreviada.

Rápidamente al interactuar con ChatGPT, pudimos darnos cuenta de las enormes potencialidades de esta herramienta. Así se hizo notar por el crecimiento espectacular de sus usuarios en pocos días, superando ampliamente a todo otro servicio tecnológico. Por ejemplo, alcanzó los 100 millones de usuarios activos solo dos meses después de su presentación, lo que la convierte en la aplicación de consumo de más rápido crecimiento de la historia.

Una vez descubiertas sus notables capacidades, comenzaron a idearse e implementarse numerosísimas aplicaciones, integrándose con otras herramientas informáticas, para potenciarlas y aumentar su productividad. Sin embargo, rápidamente también, comenzamos a descubrir sus importantes limitaciones, para las cuales se van proponiendo distintas alternativas de solución.

Entre las más destacadas están que su conocimiento de los hechos se limita a abril de 2023, es decir, no conoce hechos posteriores a esa fecha, por lo que no puede brindar o procesar información actualizada. Cada modelo puede tener información hasta la fecha en la que fue entrenado pero lo que hay que tener en cuenta no esa información no se actualiza automáticamente.

Sin embargo, nos parece que lo más preocupante, es la posibilidad de brindar información errónea y, sobre todo, lo que se conoce como alucinaciones. En este caso el modelo genera información falsa, pero redactada de una forma que resulta muy verosímil, lo cual induce a errores. Otra de los problemas que presenta es la posibilidad de respuestas sesgadas, con sesgos de diferentes tipos, como sesgos de género o raciales.

Aun así, conociendo sus limitaciones, la herramienta es sumamente poderosa y tomando los recaudos necesarios, tiene innumerables posibilidades que se van descubriendo con el paso del tiempo.

Una vez superada esta etapa de descubrimiento de la herramienta, surgió naturalmente en todo el mundo, una pregunta: ¿Cómo puedo hacer que ChatGPT procese la información específica de mi empresa? Lo que se quiere lograr es aplicar la inteligencia de ChatGPT, y las posibilidades de procesamiento que ha demostrado, pero para procesar datos propios de las empresas.

Dado el valor que podría tener la respuesta a esta pregunta, se han propuesto distintas soluciones, cada una con ciertas posibilidades y limitaciones. Como miembros del área de Transformación Digital del CDLab, en este trabajo queremos investigar las distintas posibilidades para lograr este objetivo, analizando las ventajas y desventajas de cada propuesta y, en la medida de nuestras posibilidades, desarrollar un prototipo de aplicación que permita usar ChatGPT para las necesidades de nuestra facultad. Pero teniendo en claro que los métodos estudiados pueden ser de aplicación para cualquier organización que quiera aplicar los LLM sobre datos propios.

Si se lograsen los objetivos de esta solución, las aplicaciones para el área académica serian muchísimas, ya que se lograría que el procesamiento, se realice específicamente sobre sobre un corpus de información especialmente seleccionado, por ejemplo, los contenidos a trabajar en una materia en particular, y no sobre la información sobre la que fue entrenado el modelo, sobre la que no tenemos control.

2. Observaciones importantes sobre la redacción de este informe

Si bien el objeto de estudio de esta investigación son los Sistemas Informáticos relacionados a la Inteligencia Artificial y su aprovechamiento, y por lo tanto, los temas analizados y la aplicación práctica a desarrollar, requieren de conocimientos técnicos informáticos, hemos adaptado las explicaciones a un lenguaje que pueda ser comprendido por profesionales del área de las ciencias económicas.

Nuestro objetivo es acercar estas tecnologías y conocimientos para que puedan ser aprovechados para la Transformación Digital de nuestra Facultad, principalmente, pero sabiendo que pueden ayudar a la implementación de esta tecnología en cualquier otra institución dirigidas por profesionales de las Ciencias Económicas.

Creemos que este es un camino viable y deseable para la Facultad. Así lo hemos podido comprobar en las Jornadas de Ciencias Económicas 2022, donde miembros de nuestra comunidad presentaron algunos proyectos relacionados a aplicaciones de lenguaje Python en las Ciencias Económicas, y cada vez más, se valora el dominio básico de este lenguaje por su simplicidad e interesantes aplicaciones.

Teniendo nuestra propuesta como base un dominio básico del lenguaje Python creemos que puede ser aprovechada por aquellos que tengan interés en este tipo de proyectos sin necesidad de ser del área de informática. De hecho la implementación del software de este trabajo fue realizado por un alumno de la carrera de la Licenciatura en Economía. De manera que hemos intentado transparentar las cuestiones técnicas tratando de que se puedan comprender los aspectos prácticos fundamentales para el aprovechamiento de estas tecnologías.

Muchas de las tecnologías aquí usadas son complejas y su explicación técnica requeriría de un desarrollo separado para cada una de ellas. Aquí hemos integrado muchas de ellas en un proceso que tiene como finalidad comprender cómo se pueden usar para cumplir con el objetivo de esta investigación aplicada, es decir, cómo procesar datos propios de una organización con ChatGPT. Y a su vez ejemplificar este proceso con la implementación práctica de un prototipo con el que se pueda interactuar y experimentar su funcionamiento.

Así también se ha simplificado el título haciendo referencia a ChatGPT ya que es el modelo de lenguaje que conoce el público en general. Sin embargo, los modelos de lenguaje que se usan en esta implementación son de la misma empresa OpenAI, creadora de ChatGPT, pero no específicamente ChatGPT, sino principalmente el modelo GPT 3.5, aunque se pueden usar otros modelos sin modificar los procedimientos aquí descriptos. Creemos que haciendo referencia a ChatGPT las personas no técnicas podrían interpretar mejor el objetivo de este trabajo.

Otro aspecto relevante a tener en cuenta es que, desde la propuesta de este trabajo hasta su presentación, este tema ha continuado desarrollándose, por lo que es difícil presentar en este trabajo hasta las últimas propuestas que puedan haber surgido para perfeccionar estas aplicaciones. Sin embargo, al habernos centrado en el proceso básico, la lógica central de la propuesta (denominada RAG), el trabajo permitirá comprender la naturaleza de este tipo de aplicación más allá de posibles mejoras, y en todo caso, comprender esas propuestas y mejoras sobre el esquema básico del funcionamiento del modelo.

3. Límites y alcances de este trabajo

Cabe aclarar, que habría distintas formas de interpretar la pregunta: ¿Cómo puedo hacer que ChatGPT procese la información de mi empresa?

Una interpretación es la que mencionamos en la introducción. En este estudio nos centraremos en particular, en aquella pregunta que surge al ver las capacidades de ChatGPT, pero que se limitan a procesar la información general con la que fue entrenado originalmente. Inmediatamente nos preguntamos si es posible aplicar esa inteligencia, pero sobre datos específicos de una organización. Esto lo interpretamos como la posibilidad de

pasarle nuestros datos, ya sea como un texto en un archivo de Word, un PDF o, incluso, indicarle una página web, para que procese su contenido.

Por ejemplo, en el caso de la facultad se le podría pasar un libro que sea una bibliografía de una materia para luego pedirle que extraiga las ideas principales, que nos haga un resumen, o que nos explique en otras palabras un concepto que se menciona en ese texto. También los docentes podríamos pedirle que nos genere preguntas sobre un determinado tema de esa bibliografía con alternativas de respuesta para poder generar un cuestionario en Moodle, o un mapa conceptual en formato markdown que se puede importar directamente a Mindomo sin necesidad de crearlo manualmente.

Es decir, sobre una información específica que se quiera trabajar, tanto los alumnos como los profesores pueden usar la herramienta para sus necesidades propias. En el caso de los alumnos para analizar y comprender los textos, por ejemplo, mientras que, en el caso de los profesores, para alguno de los ejemplos dados anteriormente y, en general, actividades específicas del docente, como, por ejemplo, crear las rubricas de evaluación de un tema.

Esto obviamente tiene una gran ventaja ya que, si le pedimos estos procesamientos, pero sobre información general no podemos controlar de dónde saca los contenidos, qué orientación puede tener en un determinado tema o si la información es correcta. Mientras que indicándole que el proceso lo debe hacer sobre la información suministrada nos aseguramos de disminuir posible errores o alucinaciones. La palabra usada es “disminuir” ya que según hemos podido comprobar en las pruebas realizadas igualmente puede producir información que no surja de la documentación suministrada, pero más adelante veremos que se disminuye sustancialmente esa posibilidad o es más controlable.

Otra posibilidad de interpretar la pregunta, es una interpretación más amplia, pero que inicialmente no surge de un primer contacto con un LLM. Nos referimos en este caso a todas las propuestas que están apareciendo para aprovechar las capacidades de la Inteligencia Artificial, pero ya dentro de una aplicación en particular, como un procesador de texto, una planilla de cálculo, una herramienta de diseño gráfico, etc.

De esta otra manera también se está utilizando la Inteligencia Artificial sobre datos propios, ya que, en este caso, las capacidades de los LLM se usan dentro de las mismas aplicaciones que usamos para llevar los datos de una organización. En este sentido se está produciendo una revolución en las empresas que proveen estos programas de aplicación, ya que todas, de una manera u otra, están incorporando usos de la Inteligencia Artificial dentro de sus productos y con perspectivas bastantes prometedoras.

A esta segunda interpretación no vamos a referirnos ya que podría ser objeto de otro estudio. Consideramos que encontrar soluciones para la primera interpretación es de gran importancia para aprovechar los modelos de lenguaje.

4. ¿De qué hablamos cuando hablamos de ChatGPT?

En principio nos hemos referido a ChatGPT por ser la aplicación que ha tomado dominio público, y con razón, ya que, si bien no es la única, es la que abrió el juego al público en general, puso el tema en un debate social, y comenzó una “guerra” entre los gigantes tecnológicos dedicadas a la Inteligencia Artificial, con el fin de no quedar postergadas en el liderazgo de esta tecnología.

La empresa detrás de ChatGPT es OpenAI, y otras empresas como Google o Meta tienen sus modelos y propuestas en este sentido, además de algunos modelos Open Source, es decir, proyectos de código abierto llevados adelante por una comunidad abierta de desarrolladores.

Sin embargo, para no referirnos a un producto en particular, la forma adecuada de llamar a esta tecnología es Large Language Model (LLM) o Modelos grandes de lenguaje o simplemente Modelos de Lenguaje. En la literatura se refieren comúnmente como LLM. Por lo que a lo largo de este trabajo nos referiremos en forma indistinta con estos términos.

En general, el estudio que hemos hecho está centrado en los modelos GPT 3.5 y GPT 4 de OpenAI y no en ChatGPT, como se aclaró al principio. Estos modelos son los que han estado más accesibles y estable hasta el momento, pero los métodos aquí propuestos se pueden aplicar a otros LLM. Al momento de escribir este trabajo se han liberado a todo publico varios Modelos de Lenguaje como Gemini de Google, Claude 2 de Anthropic, LLama de Meta y otros modelos, y por la gran competencia que hay es de esperar avances en el corto plazo de estos y otros modelos ya que en forma permanente surgen nuevas versiones de los mismos.

Si bien la carrera se desató con el lanzamiento de ChatGPT es sabido que estos modelos de lenguaje han venido evolucionando desde hace tiempo. De hecho, ChatGPT se base en el modelo GPT-3.5 pero ya existían sus precursores GPT-1, 2 y 3. Lo que cambio es que las versiones previas estaban accesibles solo a personas técnicas, ya que se podía acceder solo desde código de programación, mientras que con la aparición de ChatGPT se dio acceso al público en general desde una interfaz de tipo chatbot, lo que permitió que cualquier persona pudiera acceder sin ningún tipo de conocimientos técnicos.

Posteriormente al lanzamiento de ChatGPT, OpenAI lanzó su modelo GPT-4. Hasta este momento los modelos de lenguaje que se están usando en los distintos servicios de Inteligencia Artificial que se ofrecen son principalmente GPT-3 y GPT-4. No obstante el acceso a GPT-4 es aún bastante costoso por lo que en general las pruebas y aplicaciones que hemos probado se basan en GPT-3.5.

Por el momento hay una sola forma de acceder en forma abierta y gratuita a GPT-4 que es por medio del buscador Bing de Microsoft y que posteriormente paso a denominarse Copilot. Otras opciones son limitadas y en general de prueba. Sin embargo, eso no es algo que podamos comprobar, salvo por la afirmación de Microsoft, aunque si reconocemos que en algunas pruebas Copilot entrega mejores resultados que con GPT-3.5 y ChatGPT.

5. ¿Qué es un Large Language Model (LLM) y en qué se diferencia con un Buscador?

Dado que nuestra experiencia actual en la recuperación de información es el uso de buscadores en internet nos parece oportuno entender las diferencias entre un Buscador y un LLM.

Un Large Language Model es un tipo de modelo de lenguaje que utiliza inteligencia artificial para procesar y generar texto. Estos modelos están diseñados para comprender y producir lenguaje natural de manera similar a cómo lo hacemos los seres humanos.

Características Principales de los LLM:

- Capacidad de Generación de Texto: Los LLM pueden producir texto coherente y relevante.
- Aprendizaje a Gran Escala: Se entrenan en grandes cantidades de datos para comprender patrones lingüísticos.
- Adaptabilidad: Pueden ser finamente ajustados para tareas específicas.
- Contextualización: Consideran el contexto al generar respuestas.
- Conversación Interactiva: Pueden mantener diálogos con los usuarios.
- No tienen acceso a Internet de forma nativa: la información que proporcionan surge de la información en la que fueron entrenados por lo que no tienen información actualizada.

Aquí hay algunas diferencias clave entre un LLM y un buscador:

1. Generación de Contenido:

- LLM: Puede crear contenido nuevo basado en patrones aprendidos de grandes cantidades de texto. Por ejemplo, puede escribir historias, poemas o respuestas a preguntas.
- Buscador: Encuentra información existente en la web. No genera contenido original.

2. Interacción Conversacional:
    - LLM: Puede mantener conversaciones con los usuarios, respondiendo preguntas y siguiendo el contexto.
    - Buscador: Proporciona resultados basados en palabras clave, sin interacción conversacional.
3. Contexto y Comprensión:
    - LLM: Tiene memoria a corto plazo y puede recordar detalles anteriores de la conversación.
    - Buscador: No tiene memoria contextual; cada búsqueda es independiente.
4. Creatividad y Subjetividad:
    - LLM: Puede ser creativo y generar contenido imaginativo. También puede expresar opiniones subjetivas.
    - Buscador: Es objetivo y se basa en hechos verificables.

En resumen, los LLM son como compañeros de conversación inteligentes que pueden generar contenido original, mientras que los buscadores simplemente recuperan información existente. Ambos tienen sus usos y fortalezas, y su combinación podría dar lugar a herramientas aún más poderosas en el futuro[1]

Según las conclusiones del paper "Large Language Models vs. Search Engines: Evaluating User Preferences Across Varied Information Retrieval Scenarios" de Kevin Matthe Caramancion sobre la preferencia de los usuarios a la hora recuperar información afirma que:

- los hallazgos indican una división clara: los usuarios tienden a preferir los motores de búsqueda para consultas sencillas y basadas en hechos, mientras que los LLM son más favorecidos para tareas que requieren procesamiento de lenguaje natural y respuestas personalizadas.
- si bien los motores de búsqueda siguen siendo el recurso principal para obtener información rápida y factual, hay un creciente aprecio y utilidad por las capacidades conversacionales y conscientes del contexto de los LLM.
- Esta tendencia apunta a un futuro donde la integración de estas tecnologías podría proporcionar una experiencia de recuperación de información más holística y eficiente.

Esta es la tendencia tecnológica en este momento: enriquecer los LLM con acceso a información de Internet.

6. ¿Cuál es el problema que se desea resolver en este trabajo?

Los modelos de lenguaje permiten procesar la información con la que han sido previamente entrenados. De hecho, GPT significa Generative Pre-Trained Transformer o Transformador generativo preentrenado. Estos datos de entrenamiento surgen en general de datos abiertos disponibles en la web y una de las características necesarias para la capacidad que tienen, es que sean grandes volúmenes de información. Como los datos de

origen son de distinto tipo, y de un conocimiento general, el procesamiento que pueden hacer sobre ellos es realmente poderoso pero circunscripto a esos datos generales y con un corte en el tiempo hasta el momento en que fue entrenado el modelo.

De esta manera los modelos de lenguaje no están entrenados con información privada de ninguna organización en particular, por lo que no pueden procesar información específica de una empresa o negocio en específico. Ni tampoco tener en forma nativa acceso a Internet para procesar información actualizada que busque en la web. Todas estas limitaciones son las que se intentan solucionar con distintas servicios o implementaciones en particular.

Así surge el planteo, comentado en la introducción, sobre de qué manera se puede aprovechar la inteligencia de estos modelos para procesar información, pero que esa información sea la específica de una empresa u organización en particular o incluso sobre datos posteriores a sus datos de entrenamiento.

Inmediatamente surge una posibilidad nativa de ChatGPT que es solicitarle que procese la información que le vamos a pasar dentro del mismo prompt o indicación. Esto quiere decir que no solo podemos pedirle que procese información precargada, sino que podemos pasarle esa información escribiéndola en la misma ventana de texto donde escribimos el prompt. Generalmente copiando y pegando la información que deseamos que procese.

Por ejemplo, un prompt podría ser: Hazme un resumen del texto que te voy a pasar a continuación. A lo cual ChatGPT respondería: Por supuesto, puedo hacerte un resumen del texto que necesites. ¿Cuál es el texto? Y a continuación copiamos y pegamos el texto en la misma línea de ingreso donde hablamos con ChatGPT.

Este sencillo y poderoso método tiene una limitación importante ya que la cantidad de información que le podemos pasar es limitada. Esa cantidad la identificamos como la ventana de contexto que se le puede pasar a los modelos de lenguaje. Como ejemplo, tenemos los 4.096 tokens en los que se basa GPT 3.5 o los 8.192 de ChatGPT 4[1]. Como la unidad de medida que utilizan los modelos son los tokens tenemos que hacer una conversión, aproximada, para tener una idea de cuantas palabras son. Tomando como parámetro que 1000 tokens son aproximadamente 750 palabras en español nos da 3000 palabras para GPT 3.5 y 6000 palabras para GPT 4 aproximadamente.

Una de las mejoras sobre las que trabajan las empresas es aumentar este tamaño y en el caso de OpenAI has estado liberando versiones de sus modelos con mayores ventanas de contexto. Al momento de escribir este trabajo fue liberado para pruebas un LLM llamado Claude 2, de la empresa Anthropic, que asegura tener una ventana de contexto que permitiría subir libros, pero sin especificar cual es el tamaño concreto. Con cada actualización los modelos están ampliando este límite por lo que hablar de una cantidad en particular es muy relativo.

Hasta el momento la cantidad de información con que se puede alimentar a los modelos no es suficiente como para cargar los datos de una empresa u organización, por lo que se han propuestos algunos métodos de salvar esta limitación, que veremos a continuación.

7. Observaciones sobre la ventana de contexto

Para comunicarnos con un LLM tenemos un área de texto donde escribimos las solicitudes que le hacemos. Es decir, el área donde escribimos el prompt o solicitud para que el modelo realice un procesamiento y nos dé un resultado.

La ventana de contexto se refiere al límite máximo de texto que un LLM puede considerar en una sola instancia para entender y responder a una consulta. Cuando se supera esta cantidad, se pierde el contexto anterior, lo que

[1] El tamaño de las ventanas de contexto se está aumentando permanentemente por lo que aquí se mencionan las que están vigentes al momento de escribir este informe, pero es posible que vaya en aumento con las nuevas versiones de los modelos.

afecta la calidad de las respuestas generadas. Esta limitación es problemática al trabajar con conjuntos de datos extensos, ya que los modelos solo pueden procesar una fracción del corpus en cada solicitud.

Una ventana de atención más grande permite generar texto más contextualmente relevante, informativo, creativo y original. El LLM puede considerar una cantidad mayor de contexto, lo que mejora la precisión y completitud de las respuestas.

El primer método y más directo para procesar información propia es copiar la información que deseamos procesar en la misma ventana donde escribimos el prompt. Este método funciona realmente bien, aunque tiene el límite del tamaño de la ventana de contexto.

En la ventana de contexto no solo se pone información a procesar sino todo el mensaje que se quiere pasar al modelo. Es decir, el prompt o indicación que se pasa al modelo para pedirle lo que necesitamos con todos los elementos que se necesitan para un prompt efectivo, más los datos a procesar si deseamos que trabaje sobre información propia, más un historial de conversación para que el modelo siga el hilo de la conversación. Si no se incluyen ciclos anteriores de la charla cada pregunta seria independiente y no se seguiría el sentido de una conversación ya que en cada ciclo se usa implícitamente la información compartida en los ciclos anteriores de la charla.

Si interactuamos con ChatGPT en su propia interfaz, ChatGPT va "recordando" los ciclos de charla hasta que se llena la ventana de contexto y va recordando los últimos ciclos de conversación. Es decir, la información que supera ese límite se va "olvidando".

8. Posibles métodos para procesar datos propios con los LLM

Desde el lanzamiento de ChatGPT están surgiendo en forma permanente servicios y propuestas para aplicar la inteligencia de los LLM a distintos ámbitos. Han surgido servicios para ayudar con la escritura de textos, generar presentaciones e incluso se ha integrado la Inteligencia Artificial en los procesadores de texto, en las planillas de cálculo y en distintas herramientas que intentan mejorar sus servicios con aplicación de los modelos de lenguaje.

Pero como indicamos al principio del trabajo nuestro objetivo es lograr que los LLM procesen la información de nuestra organización. Con este objetivo, la búsqueda de información que realizamos nos llevó a 2 caminos principales. Uno de ellos es el denominado Fine Tuning o Ajuste Fino y el otro denominado RAG, siglas de Generación con Recuperación Aumentada.

Fine tuning es un procedimiento de ajuste de los modelos propuesto por el mismo OpenAI cuyo procedimiento se explica y se realiza dentro de la misma plataforma, quedando el modelo ajustado disponible para usar como otro modelo dentro de nuestra cuenta de OpenAI. Por el contrario, RAG es un procedimiento propuesto por los investigadores de Inteligencia Artificial de la empresa Meta[2] (Ex Facebook) y no requiere modificar o ajustar el modelo, sino que requiere un componente de recuperación de información previo a enviar la solicitud al modelo generador de texto.

Aunque daremos una descripción de ambos métodos encontrados, en este trabajo nos centraremos en el segundo método mencionado, ya que de acuerdo a las lecturas realizadas lo consideramos más adecuado a nuestros objetivos.

RAG implica un preprocesamiento de los datos antes de ser enviados a ChatGPT, método que tiene distintas formas de implementarse. En ambos casos, tanto Fine Turing como este método, son procedimientos técnicos reconocidos por las empresas de tecnología, por lo que tienen desarrollos diversos y sustentos técnicos

[2] https://ai.meta.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/

suficientes como para pensar en que son caminos adecuados para investigar una solución al planteo de este trabajo.

Hemos podido experimentar el método RAG implementándolo en forma local, es decir, sin enviar la información de la empresa al modelo de lenguaje. Esto requiere algunos conocimientos básicos de programación, especialmente del manejo del lenguaje Python. El procedimiento explica cómo se logra alimentar a ChatGPT con datos propios, para después pedirle en lenguaje natural que procese esa información, como se hace con los datos del entrenamiento original del modelo. Para estos fines se usan unos servicios, en forma de librerías, que se pueden llamar desde Python y que se han implementado justamente con el objetivo de aumentar las posibilidades de los LLM.

Sobre este modelo de trabajo han surgido startup que han desarrollados servicios en la nube que permiten subir documentos y luego pedirle que procese esa información con las capacidades de GPT 3.5 o GPT-4. Servicios como ChatPDF, ChatBase o Re:tune entre otros. Además de esos servicios hay complementos de navegadores, principalmente de Google Chrome, que ofrecen esta posibilidad. Esta forma de implementación tiene la ventaja que se puede procesar la información desde diferentes computadoras ya que el servicio está disponible en Internet. Como contrapartida tiene la desventaja que la información de la empresa se sube a servidores no propios lo que implica un riesgo de privacidad.

9. ¿Que es fine-tuning?

Consiste en ajustar el modelo a un caso de uso en específico que nosotros tengamos en mente. ChatGPT ya es un modelo ajustado sobre GPT-3 para un diálogo con el usuario mejorado, por lo que no se puede tomar de base. Si queremos hacer ajuste fino debemos partir de algún otro modelo base, siendo GPT-3.5 en este momento el más avanzado.

Nos conviene hacer fine tuning de GPT-3.5 Cuando tenemos una tarea muy específica, la cual queremos realizar con este modelo y hemos visto que tal cual sin modificación alguna no se ha podido adecuar muy bien. También nos conviene hacer fine tuning si hemos podido replicar u obtener las respuestas que esperamos de GPT-3 pero la calidad no está al nivel al que queremos y queremos mejorar la calidad de sus respuestas.

Otro punto también para hacer fine tuning es si queremos recortar los prompts que tenemos que enviarle a GPT-3 3. OpenAI nos cobra por tokens enviados y tokens recibidos. Entonces si queremos enviar menos tokens y no queremos estarle dando mucho contexto a GPT-3.5 cada vez que le mandamos un mensaje de la tarea que tiene que realizar, pues podemos hacer un fine tuning precisamente para acotar los prompts. De alguna manera es como darle el contexto una sola vez para no tener que repetirlo cada vez que le hagamos un pedido.

9.1. Costos

Otro punto también a mencionar es que si queremos hacer fine tuning de un modelo vamos a tener dos costos adicionales. Uno es un costo de hacer el fine tuning. Al momento que estamos entrenando a GPT-3 nos van a cobrar por el número de tokens que usemos con el set de datos con el que vamos a hacer el fine tuning de nuestro modelo. Una vez hecho terminado el proceso ya no vamos a consultar a GPT-3 sino que se ha creado un modelo propio. Las consultas sobre nuestro modelo, una vez que ya está entrenado, sube un poco el costo por token.

9.2. Proceso

Los procedimientos para entrenar los modelos no son complejos, pero requieren de un mínimo de dominio del lenguaje Python ya que se utiliza para preparar los datos de entrenamiento. Se utilizan algunas librerías de Python y otras específicas de OpenAi que también se llaman desde este lenguaje de programación. El proceso requiere de la preparación de gran cantidad de datos, ya que no se obtienen buenos resultados del fine tuning sin un buen volumen de datos de entrenamiento.

El fine tuning de un modelo de lenguaje grande (LLM) no consiste en enseñarle nuevas cosas de la misma manera que durante su entrenamiento inicial. En lugar de eso, el fine tuning ajusta el modelo preexistente para especializarse en tareas o contextos específicos. Se hace proporcionando al modelo ejemplos adicionales durante una fase de entrenamiento posterior, lo que le ayuda a mejorar en áreas particulares sin tener que reentrenarlo desde cero. Esto permite al modelo responder de manera más efectiva a consultas relacionadas con el área de especialización sin necesidad de repetir o volver a introducir la misma información específica en su ventana de contexto cada vez.

Otro punto también muy importante es que, sin importar la calidad y el volumen de datos que se usen para hacer fine tuning del modelo, este va a seguir teniendo alucinaciones. Aunque en el fine tuning nosotros le dimos la información específica, puede que genere una respuesta la cual no sea completamente verdad. Según las fuentes consultadas, si lo que queremos es hacer un sistema que mejore las respuestas y siempre conteste con hechos basado en el set de datos que le hemos alimentado, probablemente esta no sería la mejor opción.

En este caso se sugiere algo como búsquedas semánticas utilizando los embeddings, tema que veremos a continuación.

Al momento de escribir este trabajo OpenAI sugiere no realizar fine tuning momentáneamente, ya que los modelos que se pueden entrenar van a desaparecer en 2024 ya que están preparando modelos más avanzados para tomar como base para entrenar con esta técnica.

10. ¿Qué es generación aumentada de recuperación (RAG)?

La generación aumentada de recuperación (RAG) es una técnica que complementa la generación de texto con información de fuentes de datos privadas. Combina un modelo de recuperación, que está diseñado para buscar grandes sets de datos o bases de conocimiento, con un modelo de generación como un modelo de lenguaje grande (LLM), que toma esa información y genera una respuesta de texto.

La generación aumentada de recuperación puede mejorar la relevancia de una experiencia de búsqueda, al agregar contexto de fuentes de datos adicionales y complementando la base de conocimientos original del entrenamiento de un LLM. Esto mejora el resultado del modelo de lenguaje grande, sin tener que volver a entrenar el modelo. Las fuentes de información adicionales pueden variar desde información nueva en Internet en la que el LLM no recibió capacitación hasta contexto comercial propietario o documentos internos confidenciales que pertenecen a empresas.

También podemos destacar que es muy económico en términos de recursos de procesamiento. Los modelos de RAG son muchísimo más livianos que los de generación de texto.

La RAG es valiosa para tareas como la respuesta a preguntas y la generación de contenido porque permite que los sistemas de IA generativa utilicen fuentes de información externas para producir respuestas más precisas y conscientes del contexto. Implementa métodos de recuperación de búsqueda semántica para responder a la intención del usuario y ofrecer resultados más relevantes.

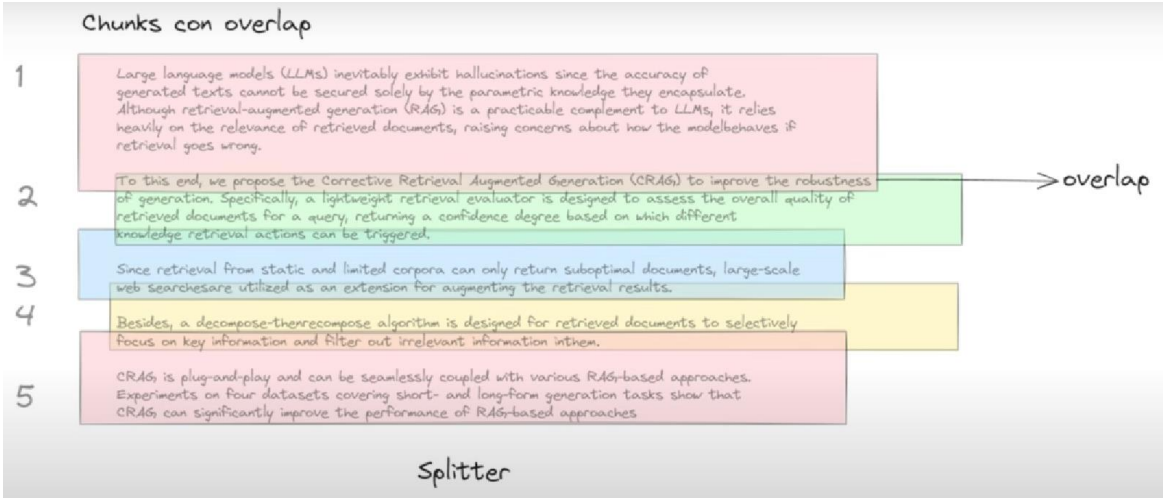
10. 1. Preparar la Base de Conocimiento

Lo primero que tenemos es nuestro conocimiento en uno o más documentos ya sean archivo de texto, PDF, Word, Json, etc. En definitiva, todo se resume a texto que queremos que tenga en cuenta el modelo a la hora de hacer una pregunta. En un esquema RAG sencillo, lo que se hace con estos documentos es dividirlos en fragmentos de texto llamados chunks.

Hay varios criterios o estrategias para estas partes; una de ellas consiste, por ejemplo, en fijar una cantidad de caracteres; otra, en fijar la cantidad de tokens, y otras más.

Aparte de eso, cada chunk podría tener un pedazo del anterior, o sea, un texto que se repite. Y esa sección se la llama overlap. Este es útil porque, a menudo, la información relevante para responder una pregunta puede abarcar varias oraciones o párrafos adyacentes a un documento. Al permitir el solapamiento de chunks, se aumenta la probabilidad de que la información relevante se capture en los chunks recuperados.

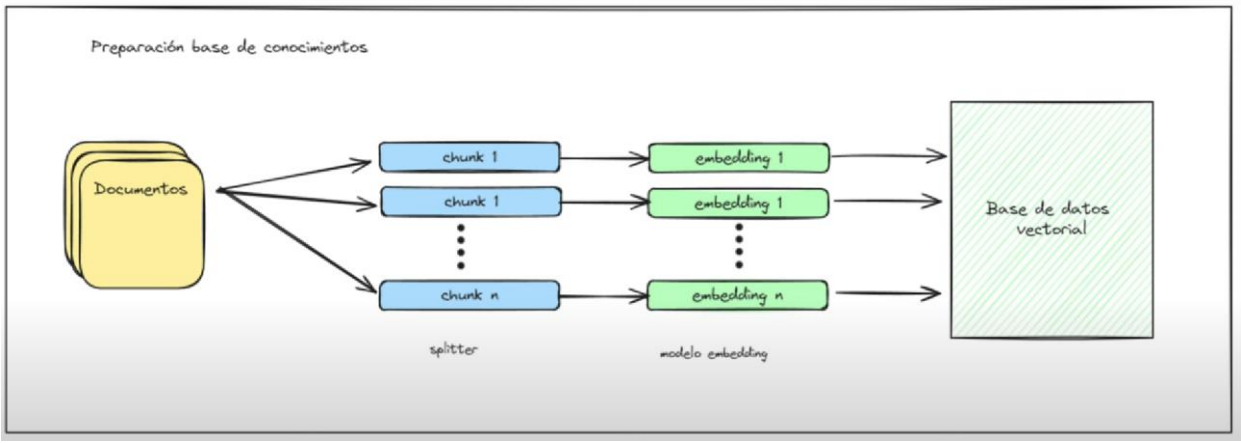
El módulo que se va a encargar de desarmar los documentos en chunks se llama splitter. Aparte de la estrategia de cantidad de caracteres o tokens, el splitter puede tomar oraciones completas o párrafos completos.



Una vez que tenemos los chunks debemos vectorizarlos, lo que se llama incrustaciones de texto o embeddings.

A modo de resumen, los embeddings son vectores que ubican una palabra o una frase en base a su semántica en un espacio vectorial, pero los mismos no son como los que pudimos haber visto en álgebra, en donde tenemos dos o tres dimensiones, sino que será un espacio multidimensional.

El siguiente paso consiste en que a todos estos embeddings o vectores debamos guardarlos en una base de datos, que será la base de conocimiento, o fuente de datos, cuando hagamos una consulta.

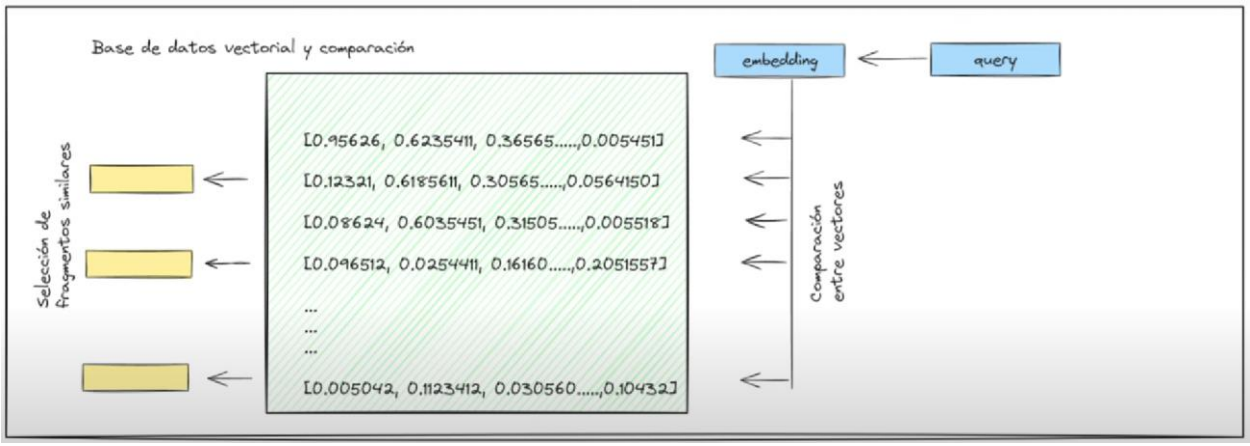


Hay varias bases de datos vectorial propietarias y algunas de código abierto como ElasticSearch y Pinecone.

10.2. Cómo se genera una respuesta cuando el usuario hace una consulta

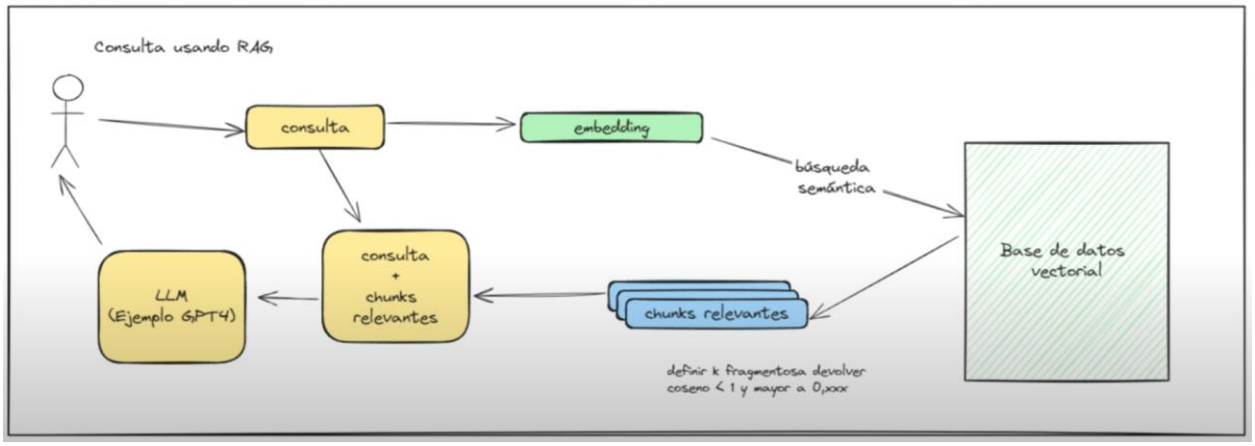
Todo arranca con la persona que hace una consulta al LLM. Del texto de esa consulta se calcula también el vector, con el mismo modelo de embedding que se usó para los chunks. Luego se hace una

búsqueda en la base de datos vectorial, de los cuales vamos a tomar los chunks que semánticamente son similares y se devuelven esos fragmentos de texto.



Antes de enviar el pedido al LLM se le agrega la consulta original. Es decir, se envían al LLM los chucks seleccionados más la pregunta del usuario.

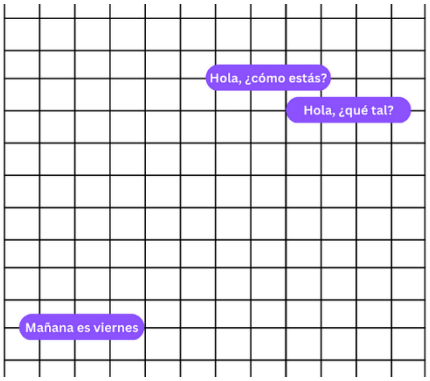
Se invoca al LLM que podría ser el GPT 3.5, 4 o el que se desee usar. Ahora el LLM tiene disponible la consulta del usuario con toda la información que hemos seleccionada como la más relevante para responder la consulta. Es decir, tiene información seleccionada para responder. Así que con eso va a poder elaborar la respuesta final y se la devuelve al usuario.



10. 3. Incrustaciones de texto y Búsqueda Semántica

Ahora bien, ¿cómo fue esa búsqueda semántica? Y ¿cómo sé que dos vectores son similares?

Vamos a pensar por un momento que podemos reducir las dimensionalidades de esos vectores a solo dos, como se muestra en la siguiente imagen. Todos los vectores tendrán un punto de origen en común. Entre ellos se va a formar un ángulo y para determinar la similitud semántica se compara el coseno del ángulo que se forma entre ellos. Cuanto más cercano sea uno el valor, entonces, esas palabras o frases serán más parecidas.



En un ejemplo práctico[3], si comparamos las frases "Hola, ¿cómo estás?" y "Hola, ¿qué tal?", los embeddings de ambas tendrán una puntuación de similitud alta, ya que ambas expresan un saludo y una pregunta sobre el bienestar de alguien.

Por otro lado, si comparamos "Hola, ¿cómo estás?" con "Mañana es viernes", la puntuación de similitud será baja, ya que estos dos textos no están relacionados en cuanto a su contenido.

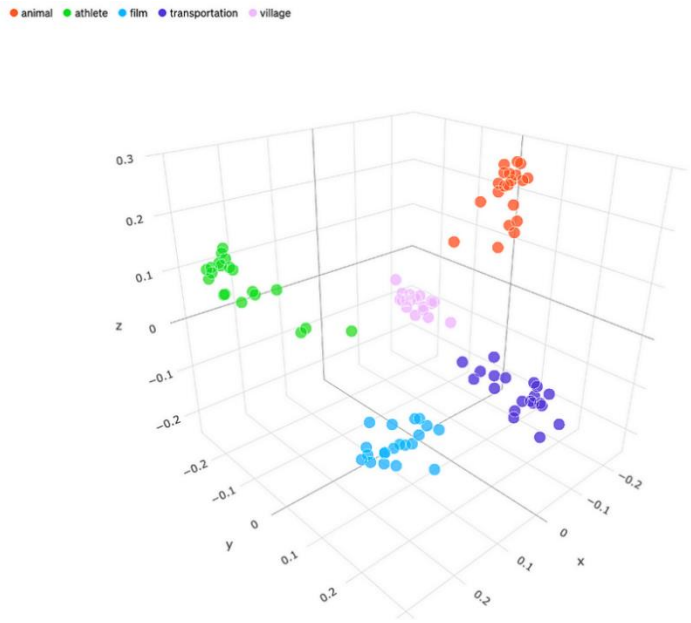
Las incrustaciones de texto o embeddings son el alma de las operaciones que permiten encadenar los LLM con los nuevos datos. Técnicamente, podemos trabajar con Modelos de Lenguaje en lenguaje natural, pero almacenar y recuperar lenguaje natural en forma de texto es muy ineficiente.

Para hacerlo más eficiente, necesitamos transformar los datos de texto en formas vectoriales. Hay modelos de Machine Learning dedicados para crear incrustaciones a partir de textos. Los textos se convierten en vectores multidimensionales. Estas incrustaciones de texto son las que nos permiten implementar la denominada Búsqueda Semántica.

Una de las técnicas popular en el campo del procesamiento del lenguaje natural que convierte palabras en vectores numéricos densos se llama Word2Vec Utiliza un modelo neuronal para aprender representaciones vectoriales de palabras a partir de grandes cantidades de texto sin etiquetar. La idea fundamental detrás de Word2Vec es capturar la semántica y las relaciones contextuales entre las palabras en un espacio vectorial de dimensiones bajas.

Una vez incrustados, podemos agrupar, ordenar, buscar y más sobre estos datos. Podemos calcular la distancia entre dos oraciones para saber cuán estrechamente relacionadas están. Y lo mejor de esto es que estas operaciones no se limitan solo a palabras clave como las búsquedas tradicionales en bases de datos, sino que capturan la cercanía semántica de dos oraciones.

[3] Ejemplos tomados de Búsqueda semántica y Embedding con Cohere | by Daniel Avila | LatinXinAI | Medium



Mediante una función de comparación simple (similitud coseno), podemos calcular una puntuación de similitud para dos embeddings y averiguar si dos textos están hablando de temas similares.

Ampliando el ejemplo podemos ver 100 muestras de un conjunto de datos que abarca 5 categorías (animal, athlete, film, transportation y village).

Las diferentes categorías se agrupan claramente en 5 clusters en el espacio de embeddings. Para visualizar este espacio, solo se tomaron 3 dimensiones (x, y, z) de las 2048 dimensiones que normalmente tiene un embedding.

La similitud coseno es una medida utilizada para evaluar la similitud entre dos vectores en un espacio vectorial. En el contexto de los embeddings, esta métrica es especialmente relevante para comprender las relaciones semánticas entre palabras o frases. La Similitud coseno se puede calcularen Python con una expresión simple como:

```
def calculate_similarity(a, b):
  return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))
```

utilizando el producto punto o producto escalar**,** y la norma (o magnitud) de los vectores. En esta función, np.dot se refiere al producto punto entre dos vectores, y np.linalg.norm es una función de la biblioteca NumPy que calcula la norma de un vector y su resultado mide el ángulo entre dos vectores. Cuanto más cercano a 1 sea el valor del coseno, más similares son los vectores. Si el valor es 1, los vectores son idénticos; si es 0, son completamente ortogonales (no similares); y si es -1, son opuestos.

Para realizar una búsqueda semántica, se debe generar el embedding del texto ingresado por el usuario para su consulta y calcular la similitud entre este embedding y los embeddings que ya tienes guardados de tu conjunto de datos. Luego, se pueden ordenar los documentos según su similitud con la consulta y devolver los resultados más relevantes. Este proceso es el que hemos implementado con la aplicación propuesta en este trabajo.

11. Generación con Recuperación Aumentada versus Ajuste Fino

RAG y ajuste fino son dos enfoques diferentes para entrenar modelos de lenguaje de IA. Mientras que la RAG combina la recuperación de una amplia gama de conocimientos externos con la generación de texto, el ajuste se centra en una gama limitada de datos para distintos propósitos.

En el ajuste, un modelo previamente entrenado se entrena aún más con datos especializados para adaptarlo a un subconjunto de tareas. Implica modificar los pesos y parámetros del modelo en función del nuevo set de datos, lo que le permite aprender patrones específicos de tareas mientras conserva el conocimiento de su entrenamiento previo inicial.

El ajuste se puede utilizar para todo tipo de IA. Un ejemplo básico es aprender a reconocer gatitos en el contexto de la identificación de fotografías de gatos en Internet. En los modelos basados en lenguaje, el ajuste puede ayudar con aspectos como la clasificación de texto, el análisis de opiniones y el reconocimiento de entidades con nombre, además de la generación de texto. Sin embargo, este proceso puede llevar mucho tiempo

y ser costoso. La RAG acelera el proceso y consolida estos costos con menos necesidades de computación y almacenamiento.

Debido a que tiene acceso a recursos externos, la RAG es particularmente útil cuando una tarea exige incorporar información dinámica o en tiempo real de la web o bases de conocimiento empresariales para generar respuestas fundamentadas. El ajuste tiene diferentes fortalezas: Si la tarea en cuestión está bien definida y el objetivo es optimizar el rendimiento únicamente en esa tarea, el ajuste puede ser muy eficiente. Ambas técnicas tienen la ventaja de no tener que formar un LLM desde cero para cada tarea.

12. Conectar datos externos con un LLM: Langchain

Langchain es una framework (entorno de trabajo) que provee herramientas de código abierto escrita en Python, que ayuda a conectar datos externos a modelos de lenguaje grandes. En cierto modo, Langchain proporciona una forma de alimentar a los LLM con nuevos datos en los que no ha sido capacitado.

Sus componentes son bloques de construcción modulares que se pueden combinar para crear cadenas. Una cadena es una secuencia de componentes (u otras cadenas) unidos para realizar una tarea específica. Langchain proporciona muchas cadenas que abstraen las complejidades al interactuar con los modelos de lenguaje.

Estas "cadenas" actúan como conexiones o puentes entre los datos de entrada y el modelo de lenguaje, permitiendo que el modelo asimile y comprenda la información nueva, aunque no haya sido previamente expuesto a ella durante su entrenamiento inicial.

Langchain es una meta-herramienta, es decir, provee de diferentes funciones, que abstrae muchas complicaciones de interactuar con tecnologías subyacentes, lo que facilita que cualquier persona con conocimientos básicos de programación construya aplicaciones de IA rápidamente.

Por ejemplo, en la V1 de nuestra aplicación se han usado rutinas como:

- División del documento en fragmentos (*chunks* en inglés): Langchain cuenta con una categoría de objetos que facilitan la división de texto en piezas más pequeñas. En la presente investigación, se utiliza un divisor de texto recursivo que delimita el texto por doble salto de línea si es posible, luego salto de línea simple, y finalmente espacio como señal de que empieza un nuevo fragmento.
- Manejo del historial de conversación y prompt: La librería cuenta con diversas herramientas para lograr que el modelo de lenguaje “recuerde” las interacciones recientes con el usuario. De esta forma, simula una conversación interactiva. También permite personalizar la cadena de texto que se pasa al agente como instrucción (*prompt* en inglés).
- Interacción con la API de OpenAI: Para este trabajo, tanto la creación de *embeddings* como la generación de texto se operan a través de los objetos de Langchain. Para acceder al *endpoint* de forma directa, hace falta generar pedidos del tipo *post,* aunque la integración con el resto del proceso se complejiza en cierta medida si se opta por seguir esta vía.

13. Procesando datos propios con ChatGPT. Un caso de aplicación.

En esta guía se explica cómo aprovechar las herramientas que ofrecen *OpenAI* [4] y *Langchain* [5] para alimentar un modelo de lenguaje con un documento propio, y luego hacerle preguntas sobre el mismo.

En este ejemplo usaremos un documento PDF para suministrar la información que queremos analizar, aunque se podrían usar otros formatos, como documentos de Word, archivos de texto plano o incluso obtener el texto de una página web para hacer preguntas sobre su contenido.

Resumidamente, el proceso consiste en dividir el texto en varios fragmentos o *chunks*, para luego representarlos en vectores que simbolizan el significado de las palabras. Cuando el usuario luego hace una pregunta, se somete al mismo proceso y se compara con estos vectores para encontrar aquellos más similares. Finalmente, se alimenta al modelo de lenguaje con los fragmentos de texto que resultaron más relevantes. Aunque existen varias opciones de implementación, este trabajo se desarrolla a través de *Python.* Los comentarios en el código se señalan con numeral: "#"

Nivel: se requieren mínimos conocimientos de programación dado que el código necesario se encuentra publicado en un repositorio de Github desde donde se puede descargar e instalar.

13.1. Versión 1

Repositorio Versión 1 y 2: PDF Chatbot using Python

Antes que nada, se debe instalar alguna versión reciente de Python[6], y un editor de código (Visual Studio Code en este caso). Lo siguiente es descargar las dependencias necesarias para el proyecto. Estas se encuentran en el archivo "requirements.txt". Trabajando en Windows, abrimos la terminal (cmd.exe), y navegamos al directorio donde se encuentra el archivo con los requerimientos. Una vez allí, corremos el comando: "pip install -r require ments.txt".

Se recomienda crear un ambiente virtual específico para el proyecto, aunque esto escapa los límites de este trabajo y no es indispensable.

Luego en el archivo de Python debemos importar las dependencias instaladas al archivo:

```
from dotenv import load_dotenv
import pickle
from pdfminer.high_level import extract_text #Simple. Se puede ampliar
muchísimo el procesamiento de pdfs
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import OpenAIEmbeddings
import os
from langchain.callbacks import get_openai_callback
from langchain.vectorstores import FAISS
```

[4] OpenAI: https://openai.com/blog/openai-api
[5] Langchain: https://python.langchain.com/docs/get_started/introduction.html
[6] Python 3: https://www.python.org/

```
from langchain.chains import ConversationalRetrievalChain
from langchain.chat_models import ChatOpenAI
from langchain.chains.conversation.memory import
ConversationBufferWindowMemory,ConversationSummaryMemory
from langchain.prompts import
ChatPromptTemplate,HumanMessagePromptTemplate,SystemMessagePromptTemplate
import time
load_dotenv() #Cargar la clave de la API
```

En el mismo directorio que nuestro archivo de python, debemos crear un archivo con la extensión ".env", y especificar nuestra clave de API de OpenAI, que se consigue en su página oficial[7]:

```
OPENAI_API_KEY = '(Clave de la API de OpenAI)'
```

Para que se cargue la clave como variable de entorno (y la reconozca la dependencia), se usa el comando "load_dotenv()" mostrado anteriormente (aunque puede hacerse de otras maneras).

A continuación, debemos cargar el archivo pdf que vamos a procesar:

```
###Lectura del pdf

file_path = 'documentos//Normas Académicas (corto).pdf'
file_path = file_path.replace(r'//',"\\")
file_name = file_path.split('\\')[-1]
if file_name[-4:] == '.pdf':
    file_name = file_name[:-4] #Para chequear si ya existen los embeddings
luego
text = extract_text(file_path)
```

Y reemplazamos la variable "file_path" por la ruta del archivo (relativa o absoluta). Cabe aclarar que, aunque la conversión del pdf a *string* de Python se hizo en forma muy sencilla, el procesamiento de pdfs puede resultar muy complejo. Además, puede que se necesiten otras herramientas o dependencias (dependiendo del caso específico). El método utilizado, (extract_text de la librería pdfminer) puede no funcionar correctamente con otros archivos. En esta etapa conviene ver el contenido de la variable text para asegurarnos de que se convirtió adecuadamente.

Con el archivo ya procesado, lo siguiente es dividir el texto en *chunks:*

```
###División del texto en chunks

character_splitter = RecursiveCharacterTextSplitter(#Se puede jugar con estos
parámetros. Recomendable adaptarlo a cada texto
    chunk_size = 500,
    chunk_overlap = 100,
    separators=['\n\n'] #Trata de capturar el final de un párrafo. Depende
mucho del pdf!
)
chunks = character_splitter.split_text(text)
```

[7] Clave de API de OpenAI: https://platform.openai.com/account/api-keys

De acuerdo a las características de nuestro documento, podemos ajustar los parámetros de la instancia del divisor de texto.

Chunk_size y chunk_overlap se refieren al tamaño máximo (en "tokens") que tendrá cada división y el tamaño de la superposición respectivamente. Un "token" equivale aproximadamente a tres cuartos de una palabra.

La variable separators es una lista de caracteres que va a usar recursivamente el divisor para ir separando. En nuestro caso, usamos un doble salto de línea por las características del pdf, pero se recomienda directamente añadir algún símbolo especial en las partes del documento que delimitan temas o párrafos específicos.

El siguiente pedazo de código revisa si ya existen en la carpeta "vector_stores" los embeddings del archivo para cargarlos. Si no existen, convierte los chunks y crea un archivo pickle en esa carpeta. De esta forma, se evita tener que usar muchas veces el servicio de embeddings para un archivo.

```
###Chequear si existe el archivo con los embeddings. Si no existe, crearlo
if os.path.exists(f'vector_stores//{file_name}.pkl'):
    print(f'{file_name} ya tiene embeddings cargados.')
    with open(f'vector_stores//{file_name}.pkl','rb') as f:
        vector_store = pickle.load(f)
    f.close()
else:
    #Al momento de hacer los embeddings no se puede ver el costo (aún no
implementado)
    embeddings = OpenAIEmbeddings(model='text-embedding-ada-002')
    vector_store = FAISS.from_texts(chunks,embedding=embeddings)
    if not os.path.exists(f'vector_stores'):
        os.mkdir('vector_stores')
    with open(f'vector_stores//{file_name}.pkl','wb') as f:
        pickle.dump(vector_store,f)
    f.close()
    print('Embeddings cargados con éxito.')
```

Lo siguiente es preparar el texto que acompaña las preguntas del usuario (llamado *prompt*). Esto es lo que va a recibir el modelo en cada interacción:

```
### Prompt personalizado (para ayudar a evitar alucinaciones)
system_template = '''
Eres una IA que ayuda a los usuarios. Intenta Continuar la siguiente
conversación:---{chat_history}---Basándote en el siguiente contexto:
---{context}---
El humano también puede preguntarte sobre la conversación que estaban
teniendo.
Si la respuesta no está en el Contexto o no puedes responder, di "lo siento,
no puedo ayudarte con esa información". No inventes una respuesta.
'''
user_template = 'Pregunta del humano: {question}. Respuesta de la IA (sólo
sobre el contexto o la conversación previa): '
messages = [SystemMessagePromptTemplate.from_template(system_template),
            HumanMessagePromptTemplate.from_template(user_template)]
qa_prompt = ChatPromptTemplate.from_messages(messages)
```

La variable “chat_history” va a contener un resumen de la conversación hasta el momento. “Context” va a incluir el fragmento de texto más relevante, y “question” el texto introducido por el usuario. Estas se van a ir agregando cuando avance la conversación.

También se deben crear ciertos objetos antes de poder conversar con el modelo:

```
### Cadena de memoria y chunks relevantes
llm = ChatOpenAI(model_name= 'gpt-3.5-turbo',temperature=0.05) #El modelo que
va a procesar los chunks y generar una respuesta
window_memory = ConversationBufferWindowMemory(memory_key='chat_history',
      llm=llm,k=4,input_key='question',
      output_key='answer',return_messages= True)
chain = ConversationalRetrievalChain.from_llm(llm = llm,
    retriever=vector_store.as_retriever(
        search_kwargs={"k": 2,"score_threshold": .78})
    memory=window_memory,

    combine_docs_chain_kwargs = {'prompt': qa_prompt},
    return_source_documents = True
```

De la porción de Código anterior, se pueden señalar algunos elementos relevantes:

- Llm: La instancia del modelo. En este caso el conocido gpt-3.5-turbo, pero existen otros más económicos, así como otros más potentes. Y la “temperatura” controla el grado de aleatoriedad en las respuestas.
- Window_memory: El objeto que controla la conversación pasada incluída en el prompt. En este caso es una ventana móvil, que va olvidando las primeras interacciones (4 en el presente ejemplo) para moderar el uso de tokens.
- Chain: El objeto principal. Permite alimentar al modelo con toda la información necesaria para mantener la conversación. El parámetro “k” de esta clase determina el número de chunks relevantes que devuelve la búsqueda (aunque se usa uno sólo).

Por último, se debe programar el bucle de conversación con el usuario:

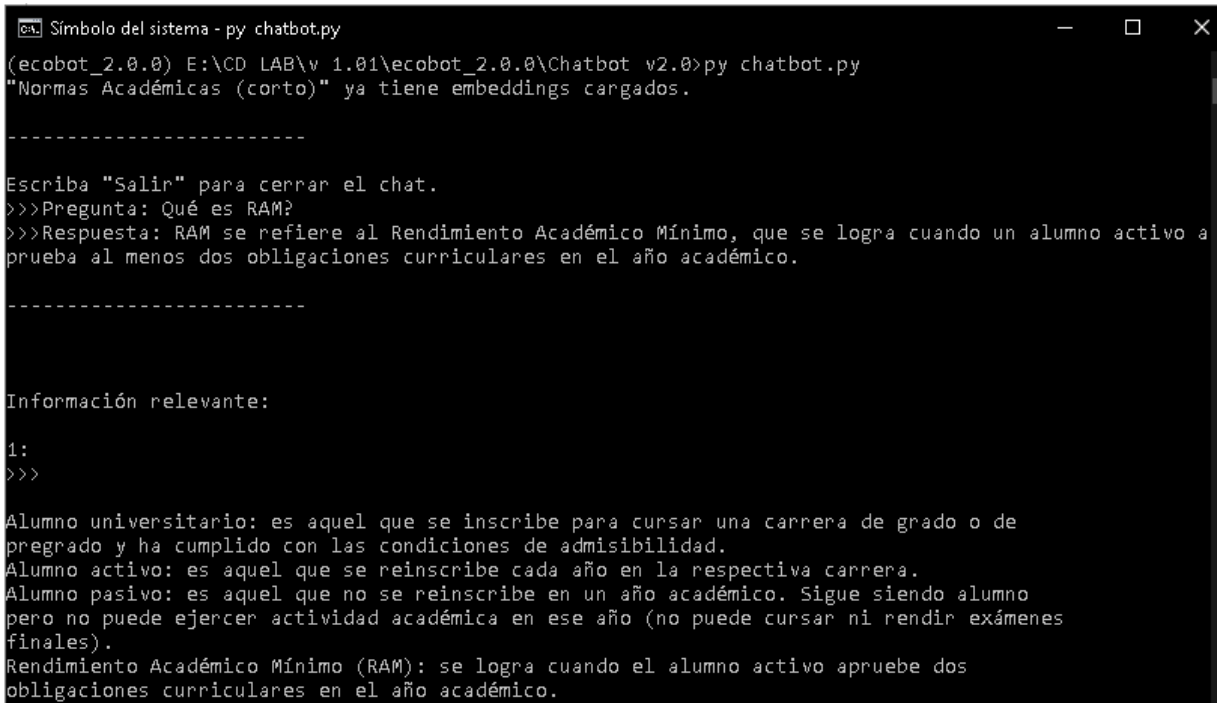
```
continuar, query = '',''
while True: #Loop principal
    print('Escriba "Salir" para cerrar el chat.' )
    query = input('>>>Pregunta: ')
    if query in ['Salir','salir','SALIR']: #Salir del loop
        break
    with get_openai_callback() as cb: #consumo de tokens
        result = chain({'question':query})
        print(f'>>>Respuesta: {result["answer"]}')
        window_memory.save_context( #Actualizar la memoria del chat
                {'question':query + '. ---Contexto: '
+       result['source_documents'][0].page_content+ '---'}
            {'answer': result['answer']}
                )
        time.sleep(4) #Da tiempo al usuario para que lea.
        print('\n'+'-'*25 + '\n\n\n')
```

```
        print('Información relevante: \n')
        for n,i in enumerate(result['source_documents']): #Chunks
            print(f'{n+1}: \n>>>{i.page_content}"\nDatos adicionales:
            \n{i.metadata}')
            print('\n'+'-'*25+ '\n')
            print('Consumo: ')
            print(cb)
```

Este bucle le va a permitir al usuario conversar con el modelo, y le va a mostrar los chunks más relevantes junto con el consumo de la API cada vez que pregunte.

Finalmente, para correr el programa debemos guardar el archivo con la extensión ".py" y correrlo en su ubicación desde la terminal con el comando py o python seguido del nombre del archivo con su extensión. En nuestro caso, "py chatbot.py":

Como ya habíamos creado los embeddings para el archivo (Normas Académicas en el ejemplo), el programa nos avisa. Luego nos permite escribir la pregunta:



13.2. Evaluación Versión 1

El modelo responde adecuadamente. Además, langchain nos permite recuperar los chunks que contienen la información. Finalmente, podemos conocer el consumo de tokens (y dinero) que significa cada pregunta y respuesta:

Aunque la tecnología funciona correctamente en muchos casos, no es perfecta. Se sugiere al lector que experimente con los parámetros, el *prompt*, y la forma de procesar sus documentos a la hora de aplicar las herramientas. Por otro lado, combinar el uso de la *memoria* y los *embeddings* puede resultar particularmente complicado. Existen algunas formas más simples de implementar la conversación con documentos, y se aconseja leer la documentación correspondiente antes de poner en práctica lo aprendido.

13.3. Versión 2:

En la búsqueda de mejores resultados, y de optimizar el proceso, seguimos buscando mejoras, aunque el proyecto inicial había cumplido con los objetivos básicos. La característica principal de esta implementación es el manejo propio de gran parte del proceso. Esto permite un mayor grado de control sobre las variables, y más flexibilidad a la hora de construir futuras versiones. También se destaca el uso de un "filtro doble" para evitar que el agente conteste preguntas no relacionadas con los fragmentos de texto.

Diferencias y características nuevas:

- Se aplica una funcionalidad propia para calcular los costos al interactuar con el modelo. Esto se logra haciendo uso de la información de uso de tokens que acompaña cada pedido a la API, junto con un objeto de Python que mapea cada modelo con los costos por token. La ventaja de este enfoque es que permite calcular costos cada vez que se crean *embeddings*, mientras que en la versión 1 sólo se calculan los costos de generación de texto.
- Se reemplaza la base de datos que contiene los *embeddings* por ChromaDB, facilitando la introducción de metadatos asociados a cada pieza de texto. Para opciones remotas, la de más fácil implementación sería Pinecone.
- Se crea una ventana de memoria propia (a modo de prueba), y los *prompts* se manejan en forma completamente personalizada. Esto limita las instrucciones que se pasan al modelo, disminuyendo a su vez el riesgo de respuestas no relacionadas con el contexto.
- Se ajusta un poco el *prompt* para obtener mejores resultados.
- Finalmente, se emplea un segundo filtro para evitar comportamiento no deseado por parte del modelo. Además de evitar la interacción con el humano si la coincidencia con los *embeddings* es muy baja, se procesa la respuesta nuevamente con otro modelo. El segundo pedido incluye un *prompt* distinto, que consulta al modelo si la respuesta a la pregunta del usuario está relacionada con los *embeddings*. Si la respuesta es afirmativa, se devuelve la respuesta del primer modelo. De otro modo, se descarta. En este caso se usa el mismo agente (gpt-3.5-turbo), aunque idealmente se trabajaría con un modelo especializado en clasificación.

Desafortunadamente, emplear la API por tercera vez puede ser muy costoso en términos de tiempo para una aplicación real.

Al poder manipular cada componente de la cadena (en vez de apoyarse en una cadena de Langchain), aumenta la flexibilidad y el control de ciertos detalles. Adicionalmente, este tratamiento es recomendable si el objetivo es entender más en profundidad el proceso completo.

13.4. Evaluación Versión 2

La aplicación del filtro en dos etapas resulta exitosa a la hora de evitar comportamiento indeseado, pero es costosa en términos de recursos. La calidad de las respuestas no cambia mucho entre ambas versiones, pues se utiliza el mismo modelo de generación de texto.

Sugerencias para futuras investigaciones:

- Extraer *stopwords* o palabras vacías antes de generar cada conjunto de embeddings.
- Usar un segundo modelo *Cross Encoder* con más resultados para mejorar la búsqueda semántica.
- Explorar nuevas posibilidades a la hora de preparar los documentos y dividirlos en *chunks* (no un enfoque tan generalista).
- Para el segundo filtro, usar un modelo especializado en clasificación.
- Hacer uso del *fine-tunning* o directamente investigar la posibilidad de entrenar modelos propios, o alternativas a OpenAI.

Con el objetivo de mejorar aún más el modelo, para obtener mejores resultados, comenzamos a explorar algunas de estas propuestas. Se creó un RAG destinado a responder preguntas sobre la biblia y se montó en un servidor. Sin embargo, la versión publicada para pruebas no está disponible por cuestiones de costos. Pero el código correspondiente a la versión local está disponible para su descarga y pruebas en: Smart Bible.

14. Generación Aumentada con Recuperación Correctiva

Repasemos los componentes y cantidad de variables que intervienen, ya que, si bien lógicamente el modelo es simple, intervienen muchos componentes y una mala elección puede afectar a todo el conjunto:

- Hay que decidir qué splitter emplear para dividir el texto en chuncks
- Decidir la longitud de los mismos y que tipo y cuanto overlap van a tener los chuncks
- También el modelo de embeddings es decir con qué sistema se van a generar
- Luego tenemos la base de datos, donde hay que almacenar esos embeddings, qué base de datos se va a utilizar
- Hay que ver también cuántos chunks pueden ser relevantes para una consulta y cómo determinamos que un chunk es relevante por el valor de su coseno.
- Luego hay que decidir el modelo a utilizar para respuesta final, es decir, GPT 3.5, GPT 4 u otro.

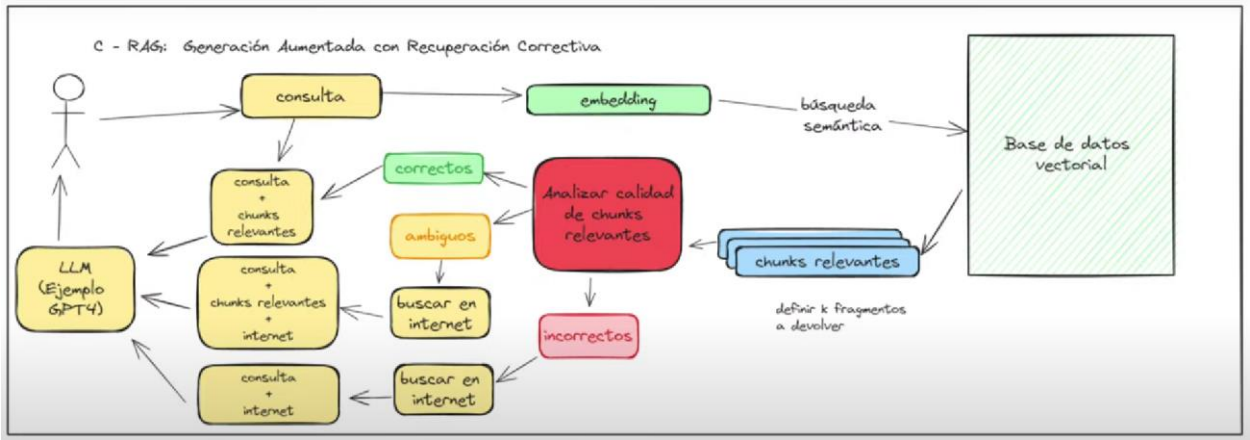
En nuestra implementación del modelo de RAG, hemos logrado combinar todas las variables que componen este modelo en una aplicación funcional que hemos probado y funciona correctamente. Sin embargo, no siempre las respuestas son óptimas y/o incorporan conceptos, que pueden ser correctos, pero no surgen directamente de la documentación suministrada.

En la versión 2 hay una mejora considerable en el control de las alucinaciones y las respuestas incoherentes con el contexto. Pero a costa de un mayor uso de recursos y la necesidad de armar varios componentes del código. También dejamos de lado la posibilidad de una ventana con el historial de conversación.

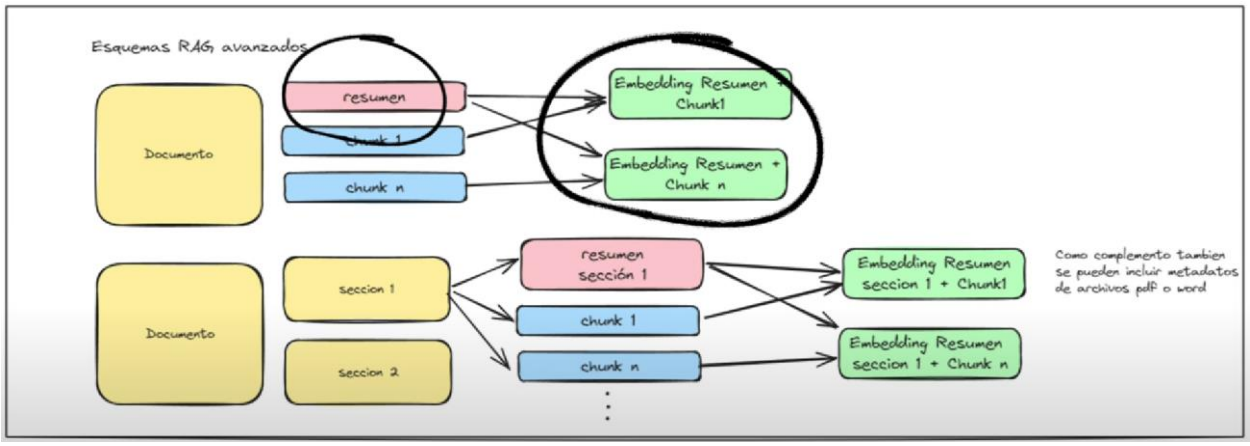
Independientemente de la forma que implementamos para mejorar los resultados de la recuperación de información, hemos visto algunas propuestas de optimización de este modelo, que se llama Generación Aumentada con Recuperación Correctiva, que comentamos a continuación y que se pueden probar en futuras implementaciones que surjan como extensión de este trabajo.

En la propuesta denominada Generación Aumentada con Recuperación Correctiva se agrega un componente adicional que analiza los chunks devueltos para ver que tan buenos son en calidad, como se muestra en la imagen siguiente. Esto es precisamente lo que se hace en la versión 2 del chatbot en el primer filtro, combinado con el control de calidad de la propia respuesta, o el segundo filtro. Aunque no una búsqueda de internet, sino simplemente retornar un mensaje predefinido "lo siento…"

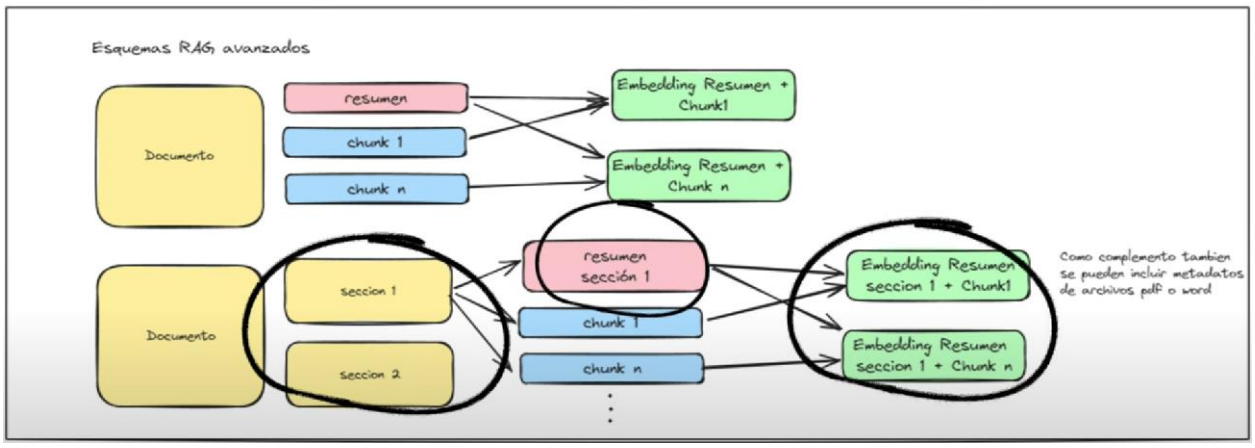
En este ejemplo se muestra que si el control de los chunks es satisfactorio se sigue por el camino normal, si son ambiguos se hace una búsqueda en internet y esa búsqueda se suma a los chunk devueltos y si son incorrectos se hace una búsqueda en internet y se usan los datos de internet.



En la búsqueda de mejorar los resultados encontramos que muchas empresas, organizaciones y personas de forma particular están dedicando tiempo a investigar las mejores soluciones o estrategias. Por ejemplo, esquemas donde se realizan embeddings de no solo los chunks, sino de los resúmenes de cada documento.



También, otra alternativa es realizar resúmenes de secciones de cada uno de estos documentos e incluirlos en los chunks, e incluso también los metadatos que por ejemplo existen en los archivos PDF o archivos de Word.



De esta manera vemos que, si bien el sistema es muy prometedor en cuanto a resultados, y ya se encuentra funcionando en varios servicios disponibles en Internet, aún requeriría mejoras para resultados óptimos. Por el momento se encuentra en una etapa de desarrollo donde se busca mejorar aún más la generación de las respuestas del sistema, para que se adecuen al objetivo principal, que es que la información generada surja exclusivamente de los documentos almacenados en la base de conocimiento.

15. CONCLUSIONES

En este trabajo de investigación, hemos implementado con éxito un prototipo de Generación Aumentada con Recuperación (RAG) que se ejecuta localmente. A lo largo de nuestro estudio, desarrollamos y perfeccionamos tres versiones del modelo, cada una mejorando la coherencia y relevancia de las respuestas generadas con respecto a la documentación suministrada al modelo.

La técnica RAG es de reciente aparición y hemos seguido en cierta manera su evolución, consultando diferentes fuentes, y utilizando recursos generados por empresas de software que proveen recursos adecuados para este tipo de implementaciones. Nuestro trabajo no es original en cuanto a aportar un nuevo conocimiento, sino en cuanto a demostrar que, con ciertos conocimientos técnicos relativamente básicos, es posible implementar esta técnica que puede aportar mucho valor al procesamiento de datos de una empresa, ya que permite procesar datos propios de la empresa con Inteligencia Artificial, conectando los datos con un LLM como ChatGPT.

Desde la primera versión, el prototipo ha funcionado correctamente, aunque las respuestas generadas no siempre fueron óptimas. Esto nos llevó a introducir controles adicionales en el modelo, dando lugar a lo que se denomina “Generación Aumentada con Recuperación Correctiva”. Estos controles adicionales han mejorado significativamente la calidad de las respuestas generadas por el modelo, asegurando que se basen más fielmente en los documentos suministrados como fuente de información.

Nuestro objetivo final es proporcionar un sistema que permita a las empresas procesar su propia información utilizando el modelo RAG ejecutado localmente. Ya que el modelo RAG ha sido implementado por varias empresas que ofrecen el servicio en Internet (Chatbase, Retune, MeetCody, etc.) lo cual significa que se debe subir la información de nuestra empresa a servidores de terceros, lo que implica riesgos de seguridad. En nuestra propuesta la información de la empresa no se comparte con ningún servicio externo, garantizando así la privacidad y seguridad de los datos corporativos. Solo se envía a OpenAI la pregunta y los fragmentos de texto que contienen la información sobre la que GPT generará la respuesta.

En conclusión, nuestro trabajo representa un avance significativo en la implementación de modelos de IA de manera segura y eficiente, demostrando que no es necesario recurrir a grandes empresas, con elevados costos, para aprovechar las ventajas de procesar datos propios con Inteligencia Artificial.

Hemos experimentado que es posible generar respuestas de calidad a partir de la documentación suministrada, manteniendo al mismo tiempo la privacidad y seguridad de los datos. Creemos que con nuestro trabajo mostramos el potencial que tiene esta tecnología para transformar la forma en que las empresas interactúan con la IA, permitiéndoles aprovechar el poder de los modelos de lenguaje de última generación de una manera segura y controlada.

16. Bibliografía

Es relevante destacar respecto de la bibliografía que, al momento de desarrollar este proyecto, la documentación existente de este tema, era muy acotada y no tan desarrollada como lo es al momento de presentar este trabajo, ya que en un año esta tecnología se ha estado desarrollando y difundiendo rápidamente. Por esa razón hemos mencionado algunas fuentes más actualizadas que consideramos son acordes a lo que se ha desarrollado en este trabajo, ya que están más claramente explicadas que las fuentes consultadas originalmente, que eran de características más técnicas.

Hemos consultado varios videos de explicaciones sobre el funcionamiento de RAG y algunas formas de implementación, sobre las que hemos tomado ideas para nuestra aplicación. Mencionamos uno de ellos especialmente por lo completo y por estar explicado por Lance Martin, Softtware Engineer, de la propia empresa LangChain, que es uno de los recursos utilizados en la implementación del prototipo implementado.

Bibliografía

Avila, D. (27 de 04 de 2023). *Búsqueda semántica y Embedding con Cohere.* Obtenido de Medium: https://medium.com/latinxinai/b%C3%BAsqueda-sem%C3%A1ntica-y-embedding-con-cohere-4a14e0209cd9

Cohen Solano, K. (04 de 12 de 2023). *Recuperación Aumentada Generativa (RAG) para la creación de chatbot de dominio específico.* Obtenido de Univeersidad de Los Andes. Colombia.: https://hdl.handle.net/1992/73530

Elastic. (01 de 01 de 2023). *¿Qué es la generación aumentada de recuperación?* Obtenido de Elastic: https://www.elastic.co/es/what-is/retrieval-augmented-generation/

Greyling, C. (12 de 10 de 2023). *Ajuste del modelo turbo GPT-3.5 de OpenAI.* Obtenido de planetachatbot: https://planetachatbot.com/ajuste-del-modelo-turbo-gpt-3-5-de-openai/

Huynh, D. (14 de 12 de 2023). *Un tutorial sobre cómo construir su propio RAG y cómo ejecutarlo localmente.* Obtenido de Medium: https://hackernoon.com/es/un-tutorial-sobre-como-construir-su-propio-trapo-y-como-ejecutarlo-localmente-langchain-ollama-streamlit

Martin., L. (20 de 02 de 2024). *RAG From Scratch.* Obtenido de YouTube: https://www.youtube.com/playlist?list=PLfaIDFEXuae2LXbO1_PKyVJiQ23ZztA0x

otros, P. L. (12 de 04 de 2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* Obtenido de https://arxiv.org/abs/2005.11401

Rose, J. (21 de 04 de 2024). *Recuperación-Aumentada Generación (RAG): Explicado Claramente.* Obtenido de CheatSheet.md: https://cheatsheet.md/es/prompt-engineering/rag-llm.es

Zachary Proser. (03 de 08 de 2023). *What is retrieval augmented generation (RAG)?* Obtenido de pinecone.io: https://www.pinecone.io/learn/retrieval-augmented-generation/