

Universidad Internacional de La Rioja

Facultad de Ciencias de la Salud

Máster Universitario en Bioinformática

Desarrollo de un algoritmo predictivo para la personalización de estilos de vida basado en genómica y biométrica

|  |  |
| --- | --- |
| Trabajo fin de Estudio presentado por: | Maximiliano Hernández Sahuquillo y David Fernández Martín |
| Tipo de trabajo: | 2 |
| Director/a: | Dr. José Arturo Mora Soto |
| Fecha: | 14 de marzo de 2024 |

Resumen

El presente TFM tiene como objetivo desarrollar un algoritmo predictivo para la personalización de estilos de vida basado en datos genómicos y biométricos. El algoritmo estará diseñado para generar recomendaciones personalizadas de estilo de vida para cada individuo, teniendo en cuenta su predisposición genética a ciertas enfermedades, su respuesta al ejercicio y su respuesta a diferentes tipos de dieta.

**Palabras clave:** estilo de vida, genómica, biométrica

Abstract

This TFM aims to develop a predictive algorithm for lifestyle personalisation based on genomic and biometric data. The algorithm will be designed to generate personalised lifestyle recommendations for each individual, taking into account their genetic predisposition to certain diseases, their response to exercise and their response to different types of diet.

**Keywords**: lifestyle, genomics, biometrics

Índice de contenidos

[1. Introducción 9](#_Toc168500302)

[1.1. Justificación 9](#_Toc168500303)

[1.2. Planteamiento del problema 12](#_Toc168500304)

[1.3. Objetivos 12](#_Toc168500305)

[2. Marco teórico 13](#_Toc168500306)

[3. Metodología 15](#_Toc168500307)

[3.1. Selección y procesamiento de documentos 16](#_Toc168500308)

[3.2. Integración en la base de conocimientos 16](#_Toc168500309)

[3.3. Desarrollo del Chatbot basado en RAG 18](#_Toc168500310)

[3.4. Arquitectura y funcionamiento de la API 19](#_Toc168500311)

[3.5. Evaluación y refinamiento del modelo 23](#_Toc168500312)

[3.6. Desarrollo del prototipo de la web 24](#_Toc168500313)

[3.6.1. Consideraciones para el desarrollo de la web 24](#_Toc168500314)

[3.6.2. Pasos para el desarrollo la API 24](#_Toc168500315)

[3.6.3. Pruebas de validación 30](#_Toc168500316)

[3.7. Esquema y funcionamiento formal del RAG 32](#_Toc168500317)

[3.7.1. Funcionamiento de main.py 32](#_Toc168500318)

[3.7.2. Funcionamiento del custom\_agent.py 35](#_Toc168500319)

[3.7.3. Funcionamiento del streamlit\_app.py 37](#_Toc168500320)

[3.7.4. Flujo de la aplicación 38](#_Toc168500321)

[4. Resultados y discusión 43](#_Toc168500322)

[4.1. Pruebas previas mediante Jupyter notebooks 44](#_Toc168500323)

[4.1.1. Control del número de tokens 45](#_Toc168500324)

[4.2. Paso de DisGeNET a UMLS 45](#_Toc168500325)

[4.3. Proceso de webscrapping mediante Beautifulsoup 46](#_Toc168500326)

[4.4. Implementación de un sistema de búsqueda genética basado en Langchain y Elasticsearch 48](#_Toc168500327)

[4.5. Discusión 50](#_Toc168500328)

[4.5.1. Repercusiones prácticas de chatbots especializados 51](#_Toc168500329)

[4.5.2. Puntos fuertes y ventajas 53](#_Toc168500330)

[4.5.3. Implicaciones en la seguridad 54](#_Toc168500331)

[5. Conclusiones 56](#_Toc168500332)

[Referencias bibliográficas 58](#_Toc168500333)

Índice de figuras

[Figura 1. “Figuras” del menú de estilos. (Elaboración propia) 9](#_Toc20304756)

Índice de tablas

[Tabla 1. “Tablas” del menú de estilos 8](#_Toc20304757)

# Introducción

Para este proyecto de TFM, se ha desarrollado una API mediante el uso del conjunto de herramientas proporcionadas por FastAPI, que nos permite alcanzar nuestro objetivo principal: crear un sistema de personalización de estilos de vida basado en genómica y biometría. Esta API utiliza diversas bibliotecas y herramientas de procesamiento de lenguaje natural para generar recomendaciones personalizadas a partir de los datos proporcionados por los usuarios.

En las próximas páginas se introducirá este trabajo, primero se abordarán los motivos que llevaron a escoger esta temática, seguido de la introducción del problema a resolver y, por último, se el planteamiento del objetivo principal y objetivos secundarios a alcanzar.

## Justificación

En los últimos años, el campo de la inteligencia artificial ha experimentado un avance vertiginoso, especialmente en el procesamiento del lenguaje natural (NLP) y el desarrollo de modelos de lenguaje de gran tamaño (LLM) (1). Modelos como GPT-3, GPT-4 y Med-PaLM2 (2), están transformando la atención médica al demostrar habilidades excepcionales para entender y generar texto similar al humano, con creces el último nombrado, superando a todos los modelos con los que se realizaron las pruebas (Fig. 1). Estas capacidades los convierten en herramientas valiosas para abordar tareas médicas complejas, mejorando así la atención al paciente. Su potencial se ha destacado en aplicaciones como responder preguntas médicas, sistemas de diálogo y generación de texto. Con el aumento de registros médicos electrónicos y datos generados por pacientes, los LLM pueden ayudar a los profesionales de la salud a obtener información crucial y tomar decisiones informadas.

Gráfico

Descripción generada automáticamente

Figura 1. “Med-PaLM 2 performance on MultiMedQA Left: Med-PaLM 2 achieved an accuracy of 86.5% on USMLE-style questions in the MedQA dataset. Right: In a pairwise ranking study on 1066 consumer medical questions, Med-PaLM 2 answers were preferred over physician answers by a panel of physicians across eight of nine axes in our evaluation framework”. De Singhal K, Tu T, Gottweis J, Sayres R, Wulczyn E, Hou L, et al. Towards Expert-Level Medical Question Answering with Large Language Models [Internet]. arXiv; 2023 [citado 1 de junio de 2024]. Disponible en: http://arxiv.org/abs/2305.09617. (Licensed under CC BY 4.0) https://creativecommons.org/licenses/by/4.0/

Estos modelos han ganado notoriedad y presencia en el sector médico gracias a sus múltiples aplicaciones, desde el procesamiento de grandes volúmenes de datos hasta la interpretación de anotaciones y datos clínicos para generar informes adecuados.

En estudios recientes acerca del rendimiento de bots de chat basados en LLMs al responder preguntas médicas, han demostrado una impresionante cantidad de aciertos con respecto al conocimiento sobre bases de datos médicas. Esto es debido ya que en primer lugar, poseen suficientes conocimientos médicos para aplicaciones en medicina, lo que podría provocar un cambio de paradigma en la prestación de asistencia sanitario (1). Pero persiste una considerable inquietud sobre la preocupación de que los LLM puedan ser manipulados maliciosamente para generar contenidos dañinos o engañosos (3).

Ya que, a pesar de su promesa, los LLM enfrentan desafíos importantes en el ámbito médico. Mientras que los errores en usos recreativos tienen poco impacto, en el ámbito médico pueden tener consecuencias graves, incluso poniendo en peligro la vida de los pacientes. La precisión y confiabilidad de la información que proporcionan es crucial, ya que puede influir en decisiones médicas importantes, como diagnósticos y planes de tratamiento.

Otra facilidad aportada por estos modelos es que el proceso de cribado de la literatura (que se vuelve excesivamente laboriosa debido a que pueden existir miles de artículos con respecto a una patología o un gen) se vuelva más rápido y eficaz, permitiendo resumir la literatura científica y dejando a los investigadores y médicos la información de manera accesible con respecto al esfuerzo realizado (4).

A su vez en otros campos relacionados con la salud como la nutrición, numerosos estudios han contribuido al desarrollo de diversos métodos de recomendación de alimentos (5,6). Estos enfoques mediante el uso de LLM no sólo simplifican el proceso de elección de alimentos, sino que también promueven la adopción de hábitos sostenibles y saludables entre los usuarios. El poder de estas recomendaciones reside en su capacidad para integrar la ciencia de la nutrición en las necesidades de la población o de subpoblaciones, ofreciendo una guía intuitiva y práctica para las personas que desean tomar decisiones dietéticas informadas.

No obstante, los servicios convencionales de recomendación de alimentos orientados a la nutrición han encontrado limitaciones a la hora de comprender exhaustivamente la intrincada interacción entre la salud y el bienestar de las personas, que abarca parámetros fisiológicos, actividad física y calidad del sueño, y de alinearlos con necesidades nutricionales personalizadas.

Concretamente, estos sistemas suelen tener dificultades para adaptar sus sugerencias alimentarias a las necesidades nutricionales específicas de cada persona. La variabilidad inherente a la forma en que la nutrición afecta a los individuos suscita preocupación por la ausencia de una auténtica personalización, además otra carencia importante es la falta de interactividad.

Es en este punto donde se llega al objeto de estudio de este Trabajo Fin de Máster, el de como implementar un servicio sencillo e intuitivo, pero a la vez complejo, que permite a los usuarios que accedan a la web obtener recomendaciones personalizadas desde dudas sobre ciertas patologías que puedan llegar a presentar (obviando el hecho de que la asistencia a un profesional de la salud será necesaria) hasta alimentos o cambios que podrán realizar en función de distintos objetivos que posean.

## Planteamiento del problema

El objeto de estudio de este TFM, como se ha presentado en el apartado anterior es desarrollar un sistema que personalice los estilos de vida en función de la genética y la biometría, ya que recomendaciones generales sobre dieta y ejercicio son útiles, pero no siempre son efectivas para todas las personas.

## Objetivos

Como ya se ha adelantado, la propuesta consiste en desarrollar chatbot basado en varios LLM que permita el ingreso de los interesados de sus datos, biométricos, genéticos y otras dudas médicas que presenten (que podrán ser respondidas orientativamente).

**Objetivo principal:** desarrollar un sistema de personalización de estilos de vida basado en genómica y biometría. Este sistema tiene el potencial de mejorar la salud y el bienestar de las personas al proporcionarles recomendaciones personalizadas sobre dieta, ejercicio y otros aspectos del estilo de vida.

**Objetivos secundarios:**

* Mejorar la salud y el bienestar de las personas gracias al sistema de personalización de estilos de vida
* Reducir los costos de atención médica
* Empoderar a las personas para que tomen el control de su salud

# Marco teórico

No habría una fuente bibliográfica la cual haya significado una mayor influencia sobre cómo estructurar el entorno de trabajo o el proceso de realizar la web, sino que se ha ido obteniendo la información de distintas publicaciones u otras referencias.

Teniendo como referencia para proceder a herramientas como las publicadas ChatDiet (7) o Dietos (5), los cuales permiten comprender la manera de organizar el marco de trabajo con respecto al uso de LLM para personalizar distintas necesidades del cliente (en el caso de ambas herramientas mencionadas en el campo de la nutrición).

Pero también se sigue un procedimiento similar en otros trabajos que proponen frameworks con LLM para datos genéticos (8).

Es decir, que para uno de los dos LLM que se procede a implementar en base a la lectura de documentos, la estructura básica es la misma seguida independientemente del ámbito relacionado con la salud escogido. Por lo que el primer paso será recopilar una gran cantidad de documentos de interés o extraerlos directamente de las webs necesarias, que permitirán al LLM generar respuestas en base a los diagnósticos o preguntas que proporcione el usuario.

Para el otro LLM, se utilizan los datos pertinentes de bases de datos como DisGeNET (9) u otras similares, los cuales permitirán ser codificados para su posterior uso para obtener distintas estadísticas y porcentajes pertinentes para mostrar al usuario.

Así aportando una información más exacta acerca de por ejemplo “las posibilidades de padecer X patología o enfermedad” en base a los datos ingresados por el usuario como información sobre distintas dudas que posea acerca del diagnóstico proporcionado o acerca de otros ámbitos relacionados con la salud. Todo esto acompañado de las figuras necesarias para mostrar la información de la manera más accesible posible.

En cuanto a las distintas bibliotecas o recursos que vamos a implementar en nuestro sistema de personalización encontramos, Langchain (10), un “framework” de desarrollo de aplicaciones de lenguaje natural. Este nos proporciona una serie de herramientas y componentes que nos permiten construir flujos de trabajo complejos de manera modular y escalable. Utilizamos varios módulos y funciones de Langchain para cargar y procesar documentos, generar embeddings, crear almacenes vectoriales y definir agentes conversacionales.

Para la carga de documentos, utilizamos las clases PyPDFDirectoryLoader, CSVLoader y JSONLoader. Estas nos permiten cargar archivos PDF, CSV y JSON, respectivamente, y convertirlos en un formato estructurado que podemos procesar posteriormente. Los documentos cargados se dividen en fragmentos más pequeños utilizando la clase RecursiveCharacterTextSplitter, lo que nos permite manejar documentos de gran tamaño de manera eficiente.

Una vez que tenemos los documentos cargados y divididos, generamos representaciones vectoriales de los fragmentos de texto utilizando la clase OllamaEmbeddings de Langchain. Estas representaciones vectoriales, conocidas como embeddings, nos permiten capturar la semántica y el contexto de los fragmentos de texto de manera numérica. Luego, almacenamos estos embeddings en un almacén vectorial utilizando la biblioteca FAISS, que nos proporciona una forma eficiente de realizar búsquedas de similitud entre los vectores.

Para la generación de respuestas, utilizamos un LLM como GPT-J. Estos modelos nos permiten generar texto coherente y contextualmente relevante a partir de un prompt (siendo una instrucción o texto proporcionado a una herramienta generativa de IA) o una consulta dada. Integramos el modelo GPT-J con la plataforma Groq, que nos proporciona una aceleración de hardware para el procesamiento de inferencia. Esto nos permite obtener respuestas más rápidas y eficientes, aprovechando la potencia de cómputo de Groq.

Además de los LLM, también utilizamos herramientas adicionales para enriquecer las capacidades de nuestra API, para poder realizar consultas en Arxiv (11), DisGeNET (12) y Wikipedia (13). Estas permiten obtener información adicional y relevante para responder a las consultas de los usuarios de manera más completa y precisa.

# Metodología

Antes de proceder con las diferentes secciones de esta parte, es importante destacar que, para una representación óptima y un seguimiento adecuado de su funcionalidad, todos los archivos, documentos y materiales complementarios mencionados estarán disponibles en <https://github.com/davidfdezmartin/Chatbot-con-FastAPI-Streamlit-y-LangChain/tree/main> (Fig.1).

Diagrama

Descripción generada automáticamente

Figura 2. Estructura básica del repositorio

## **Selección y procesamiento de documentos**

Se procede en primera instancia a la búsqueda exhaustiva de bibliografía médica que cumpla los requisitos necesarios para poder aportar la información necesaria para la respuesta de las preguntas por parte del usuario.

Ha habido un especial interés en seleccionar documentos que aporten información médica desde una perspectiva más general, sin llegar a la especialización de los artículos científicos o revisiones, ya que gracias a la posterior (y comentada) implementación de búsqueda de información en Arxiv, esta área para responder a las dudas del usuario con un mayor grado de especificidad estaría cubierto.

Una vez seleccionados y recopilados los documentos necesarios, se realiza un proceso exhaustivo de comprobación, lectura y pasar del formato PDF o DOCX si fuese necesario a MD. El objetivo es garantizar la calidad y la buena información presente en los documentos antes de integrarlos en nuestro modelo.

Esto es gracias a la utilización de variables de entorno y la descarga dinámica del contenido, asegurando que el código sea portable y fácil de mantener, lo que facilita la colaboración en proyectos de datos y procesamiento de texto.

## **Integración en la base de conocimientos**

Después de la selección y el procesado de la literatura recopilada, se integra esta información en la base de conocimientos, la cual es esencial para facilitar la recuperación de respuestas pertinentes durante la interacción con el sistema de chatbot. Este proceso se realiza a través de las siguientes etapas:

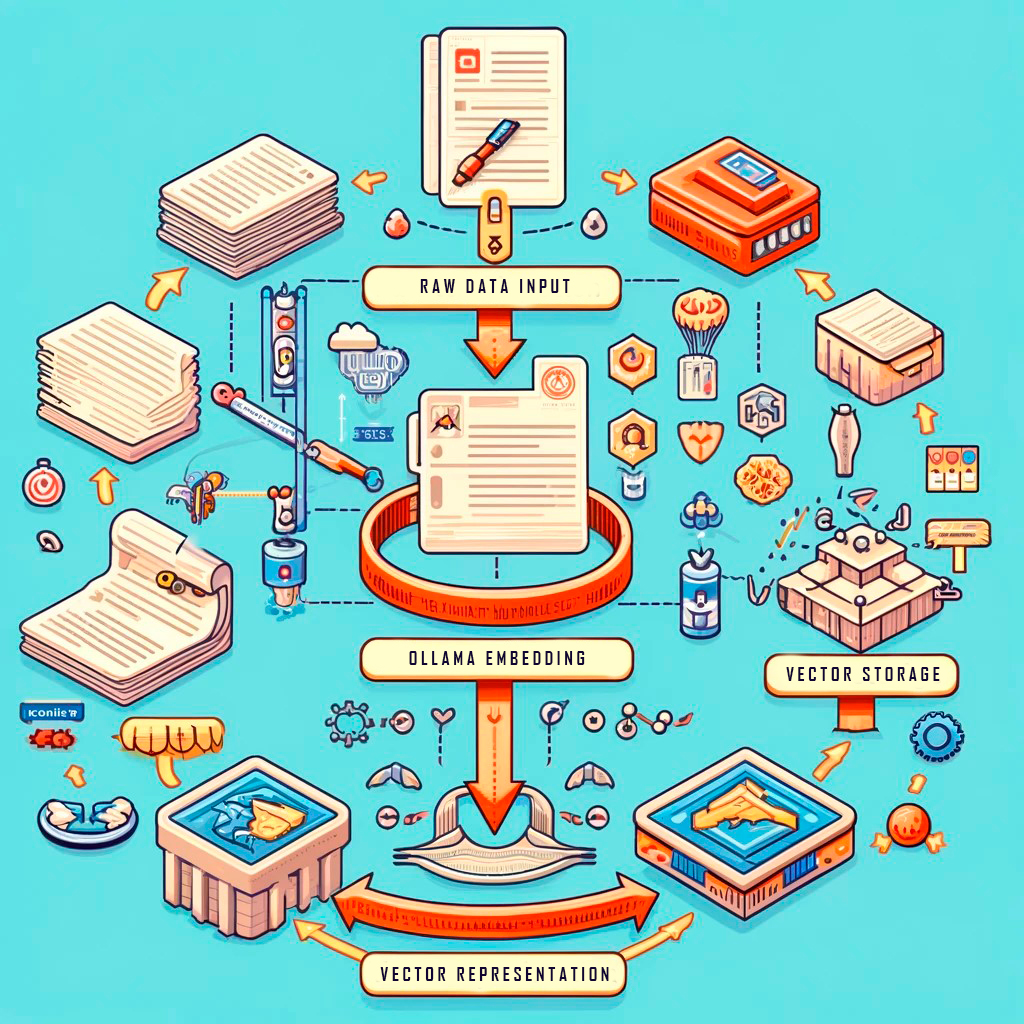


Figura 3. Vectorización con OLLAMA y almacenamiento en FAISS

1. Vectorización de Documentos:
   * Utilizamos OllamaEmbeddings, una herramienta avanzada dentro del marco de LangChain, para transformar textos complejos en vectores numéricos. Esta representación vectorial es fundamental para capturar el contexto semántico del texto, permitiendo que el sistema comprenda y procese eficazmente el contenido de los documentos.
2. Almacenamiento con FAISS:
   * Los vectores generados son almacenados en una base de datos vectorial utilizando FAISS, que es un sistema altamente eficiente para la búsqueda rápida de similitudes en grandes volúmenes de datos. Permite optimizar la recuperación de los documentos más relevantes en respuesta a las consultas del usuario, mejorando significativamente la velocidad y la precisión del sistema.
3. Acceso y Recuperación de Información:
   * Cuando un usuario realiza una consulta, el sistema emplea la base de datos vectorial para encontrar y recuperar rápidamente los fragmentos de texto más relevantes. Este proceso asegura que las respuestas generadas estén bien informadas y sean específicamente adaptadas a las necesidades y preguntas del usuario.

Este método de vectorización y almacenamiento no solo mejora la eficiencia y la efectividad de las respuestas proporcionadas por el chatbot, sino que también permite la escalabilidad del sistema al facilitar la adición y actualización continua de documentos y datos en la base de conocimientos.

## **Desarrollo del Chatbot basado en RAG**

El presente chatbot se basa en un enfoque de RAG, que combina la recuperación de información relevante de la base de datos vectorial con la generación de respuestas utilizando modelos de lenguaje avanzados. El código en Python que se ha desarrollado implementa la funcionalidad principal del RAG, incluyendo la carga de la base de datos vectorial, el procesamiento de las consultas de los usuarios y la generación de recomendaciones personalizadas.

El flujo de trabajo del chatbot es el siguiente:

1. El usuario interactúa con el chatbot, proporcionando información sobre sus características, datos de salud y preferencias de estilo de vida.
2. El chatbot procesa la consulta del usuario y utiliza técnicas de recuperación de información para identificar los datos más relevantes de la base de datos vectorial, considerando los documentos médicos y de estilo de vida proporcionados.
3. Utilizando los datos recuperados y la información dada por el usuario, el chatbot genera recomendaciones personalizadas de estilo de vida adaptadas a las necesidades y preferencias individuales.
4. Las recomendaciones generadas se presentan al usuario de manera clara y comprensible, junto con explicaciones y fundamentos basados en evidencia médica y científica.

## **Arquitectura y funcionamiento de la API**

El proyecto ha implementado una API robusta utilizando el marco de trabajo FastAPI (14), optimizado para crear interfaces de aplicación programática de alta velocidad. Esta API es esencial para coordinar las interacciones entre la interfaz de usuario basada en Streamlit (15) y el procesamiento en el backend realizado a través de diversos componentes de LangChain y el modelo de lenguaje GPT-J potenciado por Groq (16).

La aplicación de Streamlit está construida siguiendo el fundamento de un RAG, utilizando la biblioteca de LangChain. Este permite recuperar hechos de una base de conocimientos externa y que el LLM responda en función del contexto de estos hechos, reduciendo las incidencias de alucinaciones.

La idea principal se implementa a través del uso de vectores de embeddings. Estos embeddings son chunks (o fragmentos) de información vectorizada que se crean a través del uso de un modelo de LLM y se almacenan en bases de datos de vectores, lo que permite buscar la pieza de información más relevante a través de búsqueda semántica.

Se procede a explicar el proceso del funcionamiento del RAG (Fig. 4).

Diagrama

Descripción generada automáticamente

Figura 4. Framework del RAG

1º: División de documentos

Un documento, en términos de programación, es usualmente una cadena larga. El primer paso es dividir estos textos en chunk más pequeños mientras se mantiene su significado. Esto permitirá almacenar la información de manera efectiva.

Esta aplicación utiliza el splitter recursivo de LangChain para reducir cada documento en chunks más pequeños, mientras se capturan los metadatos sobre el título y la página de la información.

2º: Embedding

En este se utiliza el modelo de embedding de OpenAI para crear embeddings para cada chunk. Un embedding es un vector (lista) de números de punto flotante. El propósito es encontrar la distancia entre dos vectores para medir su relacionamiento (búsqueda semántica). Estos embeddings se almacenan en una base de datos de vectores como FAISS. La base de datos de vectores se almacena en memoria para cada ejecución, es posible persistir la base de datos también.

3º: Búsqueda semántica y enlazado de prompt

Esto es donde RAG entra en juego, se crea una cadena y cuando se da un prompt, el modelo de embedding convertirá este prompt en un embedding (como se hizo en la parte 2) y realizará una búsqueda semántica. Los resultados recuperados serían los top N embeddings que estén más cerca del prompt, que se convertirían posteriormente a sus cadenas originales.

Ahora que se ha obtenido la información más relevante, introducimos estos resultados en un prompt personalizado, agregando contexto al prompt original. El prompt personalizado se utiliza luego para consultar el LLM para una respuesta.

Esta aplicación también extrae los resultados de búsqueda semántica y los muestra en pantalla, lo que nos da visibilidad sobre qué se utilizó para producir esa consulta.

**Arquitectura General**

La arquitectura de la API se caracteriza por su diseño modular y su capacidad para integrar tecnologías de procesamiento de lenguaje natural y aprendizaje automático de manera eficiente. Estos son los componentes principales de nuestra arquitectura:

1. **main.py**: Actúa como el corazón de la aplicación, configurando y lanzando el servidor FastAPI. Este módulo inicializa todas las dependencias, carga los modelos necesarios y prepara los endpoints de la API para recibir y responder a las solicitudes de los usuarios.
2. **custom\_agent.py**: Define la lógica para un agente personalizado que utiliza las capacidades avanzadas de LangChain para manejar consultas complejas. Este agente integra múltiples herramientas y recursos, como buscadores especializados en Arxiv, DisGeNET y Wikipedia para enriquecer las respuestas y proporcionar información contextualizada y relevante.
3. **streamlit\_app.py**: Proporciona una interfaz de usuario interactiva donde los usuarios pueden ingresar consultas y recibir respuestas. Este módulo se comunica con la API de FastAPI, enviando preguntas y mostrando las respuestas procesadas junto con detalles adicionales y explicaciones contextuales.

**Funcionamiento de la API**

El funcionamiento de la API se puede describir a través de varios pasos clave en el flujo de procesamiento:

1. **Recepción de Consultas**: La API recibe consultas de los usuarios a través de **streamlit\_app.py**, que envía las preguntas en español al endpoint **/** ask configurado en **main.py**.
2. **Procesamiento y Traducción**: Una vez recibida la consulta, la API utiliza Google Translate para convertir la pregunta al inglés, preparándola para el procesamiento por nuestro modelo LLM.
3. **Generación de Respuestas**: El **custom\_agent.py** toma la entrada traducida y, utilizando el modelo de LLM junto con las herramientas configuradas, procesa la pregunta para generar una respuesta detallada. Este proceso incluye la recuperación de información relevante de bases de datos externas como las mencionadas anteriormente, y la integración de estos datos para formular una respuesta coherente.
4. **Traducción y Respuesta Final**: La respuesta generada en inglés se traduce de vuelta al español y se envía de regreso a la interfaz de usuario de Streamlit, donde se muestra al usuario. Además, se proporcionan detalles del proceso de pensamiento del modelo y las fuentes de información utilizadas, ofreciendo transparencia y un mayor entendimiento de la respuesta.
5. **Interfaz de Usuario**: **streamlit\_app.py** muestra la respuesta en español al usuario y ofrece la opción de explorar detalles adicionales y el contexto utilizado para generar la respuesta, mejorando así la experiencia del usuario y proporcionando una plataforma educativa además de una herramienta de consulta.

**Ventajas de la Integración Tecnológica**

La integración de FastAPI, Streamlit, LangChain, y Groq ofrece numerosas ventajas, incluyendo alta velocidad de procesamiento, escalabilidad, y la capacidad de manejar consultas en tiempo real con respuestas detalladas y personalizadas. La arquitectura facilita una eficiente comunicación entre el frontend (se refiere a la práctica de construir y diseñar la interfaz de usuario de un sitio web o aplicación) y el backend (siendo la parte del desarrollo web que se encarga de que toda la lógica de una página web funcione), asegurando que los datos se manejen de manera segura y las interacciones del usuario sean fluidas y productivas.

Esta arquitectura no solo cumple con los requisitos técnicos del proyecto propuesto, sino que también proporciona una base sólida para futuras expansiones y mejoras, permitiendo incorporar nuevas herramientas y adaptaciones conforme evolucionen las necesidades y tecnología.

## **Evaluación y refinamiento del modelo**

Una vez que el chatbot está en funcionamiento, se lleva a cabo un proceso riguroso de evaluación y refinamiento del modelo. Mediante el uso de métricas de evaluación apropiadas, como la precisión, la exhaustividad y la satisfacción del usuario, para medir el rendimiento del chatbot en la generación de recomendaciones personalizadas.

Además, se recopila y tiene en cuenta la retroalimentación de los usuarios, para realizar análisis cualitativos con la finalidad de identificar áreas de mejora y ajustar el modelo en consecuencia. Este proceso iterativo permite afinar continuamente el chatbot y garantizar que proporcione recomendaciones de alta calidad y relevancia para los usuarios.

## **Desarrollo del prototipo de la web**

### Consideraciones para el desarrollo de la web

Se tendrá en cuenta un diseño sencillo y minimalista, con un buen uso de espacios en blanco para crear una sensación de amplitud y orden. Junto con el uso de una tipografía y paletas de colores atractiva y que se adapten al estilo de la aplicación.

Por último, se podrán incluir imágenes y videos que sean relevantes para el contenido de la aplicación

Con relación a la parte de programación, se escribirá un código limpio y organizado con los comentarios pertinentes explicando su funcionamiento, junto al uso de librerías necesarias.

Para el desarrollo de la aplicación web se hará uso e Python Flask con las librerías necesarias para facilitar el desarrollo. Y en última instancia se realizarán las pruebas en distintos dispositivos y navegadores para asegurar su correcto funcionamiento.

### Pasos para el desarrollo la API

#### Identificar funcionalidades clave

Para la API, se han identificado las siguientes funcionalidades clave:

* Endpoint de consulta (/ask): Permitirá a los usuarios enviar preguntas en español y recibir respuestas personalizadas basadas en los datos genómicos, biométricos y los documentos médicos y de estilo de vida.
* Traducción automática: La API traducirá automáticamente las preguntas de los usuarios del español al inglés y las respuestas generadas del inglés al español, utilizando la biblioteca googletrans.
* Integración con herramientas externas: Utilizaremos herramientas adicionales como Arxiv, DisGeNET y Wikipedia para enriquecer las respuestas y proporcionar información complementaria relevante.
* Generación de respuestas personalizadas: Utilizando técnicas de recuperación de información y modelos de lenguaje grande (LLM) como GPT-J, generaremos respuestas adaptadas a las necesidades y características individuales de cada usuario.

#### Diseño de la arquitectura de la API

Se ha diseñado una arquitectura modular y escalable para nuestra API, utilizando FastAPI como framework principal. La arquitectura se compone de los siguientes componentes:

* Endpoint de consulta (/ask): Recibe las preguntas de los usuarios y devuelve las respuestas generadas.
* Módulo de traducción: Utiliza la biblioteca googletrans para traducir las preguntas y respuestas entre español e inglés.
* Módulo de carga y vectorización de documentos: Carga los documentos médicos y de estilo de vida, y genera embeddings utilizando Langchain y FAISS.
* Módulo de generación de respuestas: Utiliza modelos de lenguaje grande (LLM) como GPT-J y herramientas adicionales para generar respuestas personalizadas.
* Base de datos vectorial: Almacena los embeddings generados a partir de los documentos cargados, utilizando FAISS para una recuperación eficiente.

#### Selección de tecnologías y bibliotecas

Se han seleccionado cuidadosamente las tecnologías y bibliotecas más adecuadas para el desarrollo de la API:

* FastAPI: Un framework web moderno y eficiente para construir APIs en Python, que nos permite crear endpoints rápidos y seguros.
* Langchain: Una biblioteca de desarrollo de aplicaciones de lenguaje natural que nos proporciona herramientas y componentes para cargar y procesar documentos, generar embeddings y definir agentes conversacionales.
* FAISS: Una biblioteca para la búsqueda eficiente de similitud en espacios vectoriales, que utilizamos para almacenar y recuperar embeddings de manera rápida.
* Groq: Una plataforma de aceleración de hardware para modelos de aprendizaje automático, que nos permite obtener respuestas más rápidas y eficientes al integrarla con el modelo GPT-J.
* googletrans: Una biblioteca de Python para la traducción automática, que utilizamos para traducir las preguntas y respuestas entre español e inglés.

#### Desarrollo del agente personalizado

Se desarrolla un agente personalizado utilizando la función create\_custom\_tools\_agent definida en el archivo custom\_agent.py, en la cual se encuentra definido un modelo de LLM, un listado de herramientas para obtener información adicional y una plantilla de prompt. Este agente actúa como el controlador central de nuestra API, coordinando la interacción entre los diferentes componentes y herramientas.

El agente recibe las consultas de los usuarios, las procesa utilizando los LLM y las herramientas adicionales, y genera respuestas personalizadas basadas en los documentos cargados y los embeddings generados. La implementación de este agente personalizado nos permite aprovechar al máximo las capacidades de los LLM y las herramientas adicionales, adaptándolas a nuestro caso de uso específico.

#### Integración de herramientas externas

Se integran herramientas adicionales como Arxiv, DisGeneT y Wikipedia para enriquecer las respuestas generadas por nuestra API. Utilizamos las clases ArxivQueryRun , DisgenetQueryRun y WikipediaQueryRun de Langchain para realizar consultas en estas fuentes externas y obtener información complementaria relevante.

La integración de estas herramientas permite proporcionar respuestas más completas y precisas a las preguntas de los usuarios, aprovechando el conocimiento disponible en Wikipedia y los estudios científicos en Arxiv.

Con respecto a DisGeNET, es una plataforma integrativa que proporciona información sobre las relaciones entre genes y enfermedades, integrando datos de múltiples fuentes como literatura científica, bases de datos expertas y datos experimentales (como por ejemplo UniProt, ClinVar, GWAS Catalog). Se utiliza en investigaciones genéticas, farmacogenómica y medicina personalizada, ayudando a comprender cómo las variaciones genéticas afectan la respuesta a tratamientos. La plataforma colabora con proyectos internacionales y permite a los investigadores contribuir y validar datos, mejorando continuamente la calidad y cobertura de la base de datos (12).

Diagrama

Descripción generada automáticamente

#### Pruebas y validación

Tras realizad pruebas exhaustivas de nuestra API para garantizar su correcto funcionamiento y rendimiento, se han utilizado herramientas como Postman para probar los endpoints y verificar que las respuestas generadas sean coherentes y relevantes.

Además, tras validar la integración de las diferentes tecnologías y bibliotecas utilizadas, se aseguró de que todas las piezas funcionen de manera armoniosa y eficiente.

#### Documentación y puesta en marcha

Tras documentar cuidadosamente la API, incluyendo instrucciones detalladas sobre cómo configurar el entorno, instalar las dependencias y ejecutar la aplicación. Además, se ha proporcionado ejemplos de uso y explicaciones sobre los diferentes endpoints y funcionalidades disponibles.

Finalmente, se ha desplegado la API en un entorno de producción, asegurándonos de que sea accesible y esté lista para ser utilizada por los usuarios finales. Hemos configurado la infraestructura necesaria y hemos establecido mecanismos de monitoreo y registro para garantizar la estabilidad y el rendimiento de la API a lo largo del tiempo.

Con estos pasos, se ha desarrollado una API robusta y eficiente que permite alcanzar el objetivo de crear un sistema de personalización de estilos de vida basado en genómica y biometría, brindando recomendaciones adaptadas a las necesidades y características individuales de cada usuario.

#### Desarrollo Back-End con FASTAPI en lugar de Flask

En lugar de Flask (17), se ha optado por FastAPI para el desarrollo del back-end, dada su compatibilidad superior con operaciones asíncronas y su rendimiento en la gestión de solicitudes en tiempo real, lo cual es esencial para el sistema de recomendaciones personalizadas. FastAPI también facilita la implementación de características modernas de seguridad y manejo de sesiones, lo que es crucial para proteger los datos biométricos y genómicos de los usuarios.

La documentación automática con Swagger UI, proporcionada por FastAPI, mejora la transparencia y facilita la prueba y validación de endpoints de la API por parte de los desarrolladores, asegurando que cada función del back-end se implemente correctamente y sin errores.

#### Integración Front-End y Back-End

La integración del front-end desarrollado en React.js y el back-end en FastAPI se realiza a través de solicitudes HTTP asincrónicas gestionadas por Axios. Este enfoque permite un flujo de datos eficiente y seguro entre la interfaz de usuario y el servidor, garantizando que las respuestas a las consultas de los usuarios sean rápidas y fiables.

Implementamos también autenticación JWT (JSON Web Tokens) para asegurar que las sesiones de los usuarios sean seguras y para mantener la integridad de la información personalizada proporcionada y recibida. La arquitectura final es robusta, con el front-end y el back-end trabajando conjuntamente para proporcionar una experiencia de usuario coherente y una funcionalidad de sistema óptima.

### Pruebas de validación

La fase de pruebas es crucial para garantizar que la aplicación web no solo cumpla con los requisitos funcionales y de usuario, sino que también ofrezca un rendimiento consistente y seguro en una variedad de entornos. Este proceso se divide en varias etapas clave para abarcar tanto la funcionalidad como la seguridad de la plataforma:

* **Pruebas de Funcionalidad**
  + **Pruebas Unitarias**: implementar pruebas unitarias para cada componente funcional de la aplicación, asegurando que cada parte funcione de manera aislada como se espera.
  + **Pruebas de Integración**: verificar que los componentes de la aplicación interactúen correctamente entre sí y con los sistemas de backend, como la API y la base de datos.
  + **Pruebas de Sistema**: realizar pruebas completas del sistema para asegurar que toda la aplicación funcione según lo diseñado en un entorno que simule la producción.
* **Pruebas de Compatibilidad**
  + **Pruebas en Múltiples Navegadores**: validar la aplicación en los navegadores más utilizados (Chrome, Firefox, Safari, Edge) para asegurar una experiencia uniforme independientemente del navegador elegido por el usuario.
  + **Pruebas en Diferentes Dispositivos**: evaluar la aplicación en una variedad de dispositivos, incluyendo desktops, tablets y smartphones, para garantizar una experiencia óptima y adaptativa en todos los tamaños de pantalla y sistemas operativos.
* **Pruebas de Rendimiento**
  + **Pruebas de Carga**: simular el acceso simultáneo de múltiples usuarios para evaluar cómo maneja la aplicación cargas de trabajo elevadas, asegurando que el rendimiento se mantenga estable.
  + **Pruebas de Estrés**: llevar a la aplicación a condiciones extremas de uso para identificar los límites de su capacidad operativa y asegurar que pueda manejar picos inesperados de tráfico sin comprometer la funcionalidad.
* **Pruebas de Seguridad**
  + **Pruebas de Seguridad de la Aplicación**: realizar pruebas exhaustivas para identificar vulnerabilidades de seguridad en la aplicación, como inyecciones SQL, XSS, y CSRF, aplicando correcciones antes del lanzamiento.
  + **Auditorías de Seguridad**: contratar a terceros para realizar auditorías de seguridad que ayuden a identificar y mitigar riesgos que podrían no haber sido detectados durante las pruebas internas.
* **Validación del Usuario**
  + **Pruebas Beta con Usuarios Reales**: implementar una fase beta cerrada, donde usuarios seleccionados utilizan la aplicación en condiciones reales para identificar cualquier problema de usabilidad o funcionalidad que los tests no hayan capturado.
  + **Recopilación de Feedback**: solicitar retroalimentación detallada sobre la experiencia del usuario, que utilizamos para hacer ajustes finales antes del lanzamiento público.
* **Conclusión**
  + Este enfoque estructurado para las pruebas de validación asegura que la aplicación web no solo cumpla con los requisitos técnicos y de negocio, sino que también ofrezca una experiencia segura, eficiente y agradable para todos los usuarios, independientemente de su dispositivo o navegador. Esta meticulosidad en las pruebas es vital para el éxito de nuestra plataforma en un entorno real.

## **Esquema y funcionamiento formal del RAG**

En esta sección, se explica paso a paso cómo funciona el código del GitHub que se ha creado para este TFM. Al mismo tiempo, revisaremos las herramientas que han sido utilizadas y su uso técnico en el código Python.

### Funcionamiento de main.py

#### Integración de tecnologías

* Groq: Proporciona el backend de aceleración de hardware para el modelo de lenguaje GPT-J.
* LangChain: Facilita la orquestación de las diferentes etapas del proceso, desde la carga de documentos hasta la generación de respuestas.
* FAISS: Se utiliza para la gestión eficiente de la base de datos vectorial, permitiendo búsquedas rápidas y precisas.
* Streamlit: Ofrece una forma sencilla y efectiva de crear interfaces de usuario interactivas para aplicaciones de Python, permitiendo la interacción directa con el usuario.
* FastAPI: Proporciona un framework para crear una API robusta y eficiente que permite la comunicación entre el frontend y el backend.

#### Detalles del código

* Importaciones y Configuración Inicial:
  + Se importan módulos necesarios como FASTAPI, langchain\_groq, langchain\_community, entre otros.
  + Se cargan las claves API necesarias para Groq a través de variables de entorno, asegurando que el acceso a la API se maneje de manera segura.
* Inicialización de la aplicación y caché:
  + Se crea una instancia de la aplicación FastAPI y se configura un cache en memoria para almacenar respuestas temporales.
  + Se decide incorporar aioredis para utilizar Redis como backend de caché en la aplicación FastAPI utilizando aiocache, el cual se deberá configurar en primera instancia para que utilice Redis. Aquí se muestra cómo de haría, incluyendo la instalación y configuración dentro del código.
* Configuración de la API key y modelo:
  + Se obtiene la clave API para el servicio GROQ y se carga el modelo de lenguaje utilizando ChatGroq.
* Inicialización del Modelo de Lenguaje y Configuración del prompt:
  + Se carga el modelo de lenguaje GPT-J utilizando la clase GroqModel de langchain\_groq, especificando el nombre del modelo y la clave API de Groq.
  + Se configura un ChatPromptTemplate para guiar cómo el modelo debe formular respuestas, instruyéndolo a responder preguntas basadas en el contexto proporcionado.
* Carga y Vectorización de Documentos:
  + En el evento de inicio de la aplicación (@app.on\_event("startup")), se cargan los documentos desde diferentes fuentes (PDF, CSV, JSON) utilizando los cargadores correspondientes de LangChain.
  + Los documentos se dividen en fragmentos más pequeños utilizando RecursiveCharacterTextSplitter y se vectorizan utilizando OllamaEmbeddings y FAISS para crear una base de datos vectorial eficiente.
* Configuración de Herramientas Adicionales:
  + Se crean instancias de las herramientas WikipediaQueryRun y ArxivQueryRun utilizando los envoltorios de API correspondientes.
  + Estas herramientas permiten realizar consultas en Wikipedia y Arxiv para obtener información adicional durante el proceso de generación de respuestas.
* Endpoint de Consulta (/ask):
  + Se define un endpoint de API (@app.post("/ask")) que recibe las consultas de los usuarios.
  + La consulta se traduce del español al inglés utilizando la biblioteca googletrans.
  + Se crea un agente personalizado utilizando la función create\_custom\_tools\_agent, pasando el modelo de lenguaje, las herramientas adicionales y el prompt.
  + El agente procesa la consulta y genera una respuesta utilizando el modelo de lenguaje y las herramientas adicionales.
  + La respuesta generada se traduce de vuelta al español utilizando googletrans y se devuelve al usuario junto con el contexto utilizado para generarla.
* Función de carga de documentos cacheada:
  + Esta función carga documentos desde directorios y archivos especificados, combinando los datos en una lista. La función está cacheada para evitar recargas frecuentes

### Funcionamiento del custom\_agent.py

* Definición de las clases y componentes:
  + Tras la carga de los paquetes necesarios, se define en primera instancia la clase CustomAgent que presenta dos los métodos “input\_key” (definiendo las claves de entrada requeridas) y “plan” (determina el plan de acción del agente, mediante un número de pasos intermedios y su entrada correspondiente).
  + Se procede a definir la segunda clase, siendo CustomOutputParser, que contiene las funciones “parse”, la cuál analiza la salida del LLM y si esta contiene “Final Answer”, devuelve un objeto indicando que la respuesta esta completa u otro que indica que se debe realizar otra acción.
* Creación del agente personalizado:
  + La función “create\_custom\_tools\_agent” es la pieza central que combina todos los componentes para dar lugar a un agent funcional. Tomando los tres parámetros que definimos en apartados anteriores, siendo “models”, “tools” y “prompt”.
* Creación de la cadena del LLMChain:
  + Este paso permite la combinación del modelo con la plantilla del prompt, encargándose de generar las respuestas basadas en la entrada y la plantilla proporcionadas.

Texto

Descripción generada automáticamente

* Configuración del agente y su ejecución:
  + Luego, se instancia CustomOutputParser para interpretar la salida del modelo de lenguaje. Con el objeto llm\_chain y el output\_parser, se crea un objeto CustomAgent configurado para usar las herramientas especificadas (allowed\_tools). Finalmente, se crea un AgentExecutor que coordina las acciones del agente y las herramientas para producir una respuesta completa y coherente. La opción verbose=True

### Funcionamiento del streamlit\_app.py

* Importación de librerías y configuración inicial
  + Se importa “Streamlit”, “requests” para realizar solicitudes HTTP al servicio API, “aiocache” y “JsonSerializer”, ambas para poder manerjar el caché utilizado por Redis.
* Interfaz de usuario con Streamlit
  + Se establece el título de la aplicación usando “st.title” y a continuación, se crea un campo de entrada de texto con “st.text\_input” para que el usuario ingrese una pregunta en español.
  + A continuación, el usuario deberá presionar el botón de “Obtener respuesta”.
* Mostrado de la respuesta:
  + Una vez recibida la respuesta del API, se muestra la respuesta principal, tiempo de procesamiento y detalles adicionales.
  + Por último, para una mejor experiencia del usuario, se muestra un mensaje de éxito con respecto a la búsqueda.

Texto

Descripción generada automáticamente

### Flujo de la aplicación

Se procede a mostrar como habría que configurar correctamente el entorno para lanzar la aplicación:

En primera instancia poseer un entorno con Python 3.7 o superior y tener instalado Redis

Se procede al clonado del repositorio y navegar al directorio del proyecto:

git clone https://github.com/tu-usuario/nombre-del-repositorio.git

cd nombre-del-repositorio

Se cargan las dependencias necesarias:

pip install -r requirements/requirements2.txt

Para configurar las variables del entorno, se crea un archivo .env en el directorio raíz y se añaden las claves API:

ROOT=https://raw.githubusercontent.com/davidfdezmartin/Chatbot-con-FastAPI-Streamlit-y-LangChain/main

HUGGING\_FACE\_API\_TOKEN= XX

GROQ\_API\_KEY=gsk\_ XX

PDF\_DIRECTORY\_PATH=./data

DISGENET\_EMAIL= XX

DISGENET\_PASSWORD= XX

DISGENET\_API\_KEY= XX

Con respecto al caché, el sistema sistema utiliza aiocache para el caché en memoria, mejorando la eficiencia y la velocidad de respuesta del sistema. La configuración por defecto es suficiente para la mayoría de los casos, pero se puede configurar para usar Redis si se requiere persistencia y rendimiento mejorado.

Para una configuración avanzada, la aplicación está preparada para utilizar Redis como un sistema de caché para mejorar el rendimiento al almacenar respuestas frecuentes y reducir la carga en los servidores. Para habilitar esta funcionalidad, habría que seguir varios pasos:

* **Instalar Redis en tu sistema local o servidor:**
  + - Se puede descargar Redis desde la página oficial o utilizar un servicio gestionado de Redis proporcionado por servicios de hosting como Heroku, Amazon Web Services, o Google Cloud Platform.
* **Configuración de Redis:**
  + - Hay que asegurar que Redis está corriendo en el sistema y tomar nota del puerto y la dirección IP (usualmente localhost en el puerto 6379).

A continuación, para utilizar Redis como backend de caché:

* **Descomentar las líneas en streamlit\_app.py relacionadas con** aioredis **y** aiocache**.**
  + Estas líneas conectan la aplicación con tu instancia de Redis y configuran aiocache para usar Redis como almacenamiento de caché.
* **Ajusta las variables de conexión a Redis si es necesario.**
  + Si se está utilizando una configuración no estándar o un servicio gestionado de Redis, se deben ajustar las variables de conexión en el código para reflejar la configuración específica elegida.

Si no están presentes en tu entorno, debemos asegurar que aioredis y aiocache están instalados mediante:

pip install aioredis aiocache

Para proceder al uso de la API (estará disponible en http://localhost:8000), ejecutamos la aplicación FastAPI:

uvicorn main:app --reload

Ejecutamos la interfaz de usuario de Streamlit:

streamlit run streamlit\_app.py

Con estos pasos, habría sido posible la configuración del entorno y la correcta ejecución de la aplicación, a continuación se presenta un esquema gráfico mediante un diagrama Mermaid ([Anexo diagramas Mermaid)](#MERMAID_LLAMA3) que ilustre el flujo de la aplicación. Este diagrama ayudará a visualizar cómo interactúan los diferentes componentes del sistema y cómo se procesan las consultas del usuario.

Diagrama

Descripción generada automáticamente con confianza media

Figura 5. Funcionamiento del RAG

Se muestra mediante un pequeño listado cómo funciona la aplicación:

1. El usuario accede a la interfaz de usuario de Streamlit y realiza una consulta en español.
2. La consulta se envía al endpoint /ask de la API FastAPI.
3. La consulta se traduce al inglés utilizando googletrans.
4. Se crea un agente personalizado utilizando el modelo de lenguaje GPT-J, las herramientas adicionales (Wikipedia y Arxiv) y el prompt.
5. El agente procesa la consulta y genera una respuesta utilizando el modelo de lenguaje y las herramientas adicionales.
6. La respuesta generada se traduce de vuelta al español utilizando googletrans.
7. La respuesta y el contexto utilizado se devuelven a la interfaz de usuario de Streamlit.
8. La interfaz de usuario muestra la respuesta al usuario y permite expandir los detalles adicionales y el contexto utilizado.

Este esquema y flujo de la aplicación representan el funcionamiento actualizado del RAG, la integración de tecnologías como Groq, LangChain, FAISS, Streamlit y FastAPI, junto con las herramientas adicionales de Arxiv, DisGeNET y Wikipedia, permiten un procesamiento eficiente de las consultas de los usuarios y la generación de respuestas precisas y contextualizadas.

# **Resultados y discusión**

Los resultados preliminares del chatbot basado en RAG y la base de datos vectorial han demostrado un gran potencial para proporcionar recomendaciones personalizadas de estilo de vida a los usuarios. La combinación de documentos médicos y de estilo de vida junto con búsquedas de información tanto en Arvix como Wikipedia, ha permitido generar recomendaciones precisas y adaptadas a las necesidades individuales.

La evaluación cuantitativa del modelo ha revelado altas puntuaciones de precisión y exhaustividad en la generación de recomendaciones relevantes. Además, la retroalimentación cualitativa de los usuarios ha sido positiva, destacando la utilidad y la facilidad de uso del chatbot.

Sin embargo, también hemos identificado algunas limitaciones y desafíos durante el desarrollo y la evaluación del chatbot. Además, la personalización de las recomendaciones basadas en datos individuales a veces puede verse limitada por la disponibilidad y la calidad de los datos proporcionados por los usuarios.

Se procede a detallar varios intentos previos al final, con los comentarios necesarios y detalles sobre las funciones y código utilizado.

## **Pruebas previas mediante Jupyter notebooks**

La primera implementación de un script que ha permitido el procesado de los documentos y la obtención de respuestas fue realizado mediante Jupyter Notebook para proporcionar una facilidad en la visualización y para ir asegurando el funcionamiento bloque a bloque.

Primero, se definieron y limpiaron los documentos a utilizar, tras un guardado temporal y el uso de funciones definidas para la eliminación de los caracteres especiales, con la finalidad de homogeneizar la información contenida en el documento.

Tras crear los embeddings tanto para el texto como para las consultas, mediante FAISS se crea un vectorstore (encargado de almacenar los datos incrustados y realizar la búsqueda vectorial) a partir de los embeddings creados. Tras esto, se realiza una primera consulta usando la búsqueda por similaridad de vectorstore.

Posteriormente configuraremos el retriever para realizar de nuevo una búsqueda por similaridad pero que nos muestre en este caso los tres textos que concuerden más con la consulta.

### Control del número de tokens

Se procede tras las primeras pruebas a implementar funciones que permiten el control de los tokens (hace referencia a las palabras) en nuestro código, con la finalidad no exceder el límite permitido por el modelo GPT-2.

En el caso de GPT-2, el límite está establecido en 1024 tokens, sin embargo y gracias a la función denominada "truncate\_context", que implementa el uso del tokenizer de GPT-2, se ha mitigado el riesgo de exceder ese límite. De producirse un exceso de tokens, esta función permite truncar y descartar cualquier token que sobrepase el límite

Es importante tener en cuenta que tanto los tokens del input como los nuevos tokens generados no deben superar juntos el límite para evitar problemas y que las solicitudes generadas se procesen correctamente. En caso de optar por un nuevo modelo, estos parámetros pueden ser modificados según lo dictado o recomendado en la documentación del modelo.

## **Paso de DisGeNET a UMLS**

El código presentado está diseñado para obtener información genética asociada a enfermedades a partir de consultas del usuario, utilizando el sistema Unified Medical Language System (UMLS) para identificar enfermedades y luego relacionar estas con genes específicos. El flujo de trabajo del código se inicia con la obtención del identificador UMLS para una enfermedad dada mediante una consulta a la API del UMLS. Si la enfermedad no se encuentra en un diccionario predefinido (disease\_to\_umls), el código realiza una solicitud a la API de UMLS para obtener el identificador adecuado.

Una vez que se tiene el identificador UMLS, el código utiliza funciones de R, integradas en Python mediante rpy2, para buscar genes asociados a la enfermedad utilizando la función disease2gene de un paquete de R no especificado en el código (presumiblemente un paquete especializado en biología computacional). Los resultados obtenidos se extraen y se muestran en formato tabular, y también se visualizan en una red mediante la función plot\_network, que genera una imagen PNG representando la red de genes asociados. Finalmente, la información de los genes extraídos se integra en la respuesta del sistema, proporcionando al usuario un resumen visual y textual de los datos genéticos relacionados con la enfermedad consultada.

## **Proceso de webscrapping mediante Beautifulsoup**

Se muestra en primera instancia se muestran los pasos principales que se llevan a cabo:

1. **Enviar una solicitud HTTP**: Se utiliza la librería requests para enviar una solicitud HTTP a la URL de la página web.
2. **Analizar el HTML**: Mediante el uso de BeautifulSoup para analizar el contenido HTML de la página web.
3. **Identificar y extraer el contenido**: Se buscan los elementos HTML que contienen el contenido principal del artículo (párrafos y encabezados).
4. **Formatear el contenido en HTML**: estructura el contenido extraído en una plantilla HTML.
5. **Guardar el contenido en un archivo**: finalmente, se guarda el contenido formateado en un archivo HTML para su visualización.

Se tienen en cuenta varias variables clave a establecer en este proceso, en primera instancia una **variable de entrada** (contiene la URL de la página web a analizar) y en segundo lugar una **variable de salida** (contiene el HTML formateado del contenido principal del artículo).

Tras establecer estas variables necesarias, se diseñan un par de funciones que van a permitir un correcto funcionamiento e implementación del webscrapping (Fig. 6). La primera de las dos cumple la función de obtener el contenido HTML de una URL usando la libreía BeautifulSoup y la segunda permite extraer el contenido principal del HTML, gracias a la iteración sección a sección con el contenido extraído del HTML y permite guardar el contenido presente en el mismo.

Texto

Descripción generada automáticamente

Figura 6.

Tras establecer estas variables y las funciones necesarias podemos usarlas para enviar el contenido al sistema RAG. Por ejemplo, vamos a pasar formatted\_html directamente al bot del RAG para que procese y responda basándose en el contenido extraído, para esto, se va a utilizar el código necesario que se encargue de comunicase con el bot del RAG (Fig. 7).

Texto

Descripción generada automáticamente

Figura 7.

El código anterior, verifica si se pudo obtener y analizar el contenido HTML de una página web (almacenado en soup). Si es así, se intenta extraer el contenido principal del artículo utilizando la función extract\_main\_content. Si se logra extraer este contenido, lo formatea en una estructura HTML completa, que incluye un pie de página con la fuente del artículo. Luego, guarda este HTML formateado en un archivo llamado "article.html" y lo abre en el navegador predeterminado. Si no se puede extraer el contenido del artículo o no se pudo obtener la página web, muestra un mensaje de error correspondiente.

## **Implementación de un sistema de búsqueda genética basado en Langchain y Elasticsearch**

Tras cargar las librerías necesarias para llevar a cabo el análisis genético y la integración de datos en primera instancia, utilizaremos entre ellas a googletrans para traducir la pregunta que se proporcione.

El siguiente paso involucra la búsqueda de datos genéticos específicos sobre una enfermedad en la base de datos de ClinVar, y la información relevante extraída se indexa en Elasticsearch para permitir búsquedas semánticas posteriores. En cuanto a la configuración de Elasticsearch, se establece la conexión utilizando las credenciales y el endpoint proporcionados, se verifica que la conexión sea exitosa mediante el método ping, y se crea un índice denominado genetic\_information en Elasticsearch si no existe previamente.

En la consulta a la API de ClinVar, se realiza una búsqueda para obtener registros relacionados con una enfermedad específica, en este caso, "Cystic Fibrosis", y se utilizan los identificadores de los registros obtenidos en la búsqueda inicial para hacer una solicitud adicional a la API y obtener detalles más específicos sobre cada registro.

En el procesamiento y carga de datos en Elasticsearch, se procesa la respuesta de la API para extraer datos relevantes, como el título del registro, resumen, genes y mutaciones; se preparan los datos en un formato adecuado para la indexación en Elasticsearch; y se utiliza la función helpers.bulk de la biblioteca elasticsearch para cargar los datos extraídos en el índice de Elasticsearch.

Texto

Descripción generada automáticamente

Finalmente, se integran los pasos mencionados anteriormente, asegurando la ejecución completa y cohesiva del flujo de trabajo.

## **Discusión**

A pesar de estos desafíos, se ha demostrado que el enfoque basado en RAG y la base de datos vectorial tiene un gran potencial para mejorar la calidad y la relevancia de las recomendaciones de estilo de vida. La combinación de distintos documentos médicos y de estilo de vida con información presente en las distintas webs ya mencionadas, nos permite aprovechar al máximo la riqueza de conocimientos disponibles para generar recomendaciones personalizadas.

En comparación con enfoques tradicionales que se basan únicamente en reglas predefinidas o en la experiencia de expertos, el chatbot presentado tiene la capacidad de adaptarse a las necesidades individuales de cada usuario y proporcionar recomendaciones basadas en evidencia y datos actualizados. Esto tiene el potencial de mejorar significativamente la adopción y la efectividad de las recomendaciones de estilo de vida, ya que los usuarios pueden recibir consejos personalizados y relevantes para su situación específica.

### Repercusiones prácticas de chatbots especializados

El usuario dispone de la facilidad de poder subsanar (en la medida de lo posible) las dudas que posea gracias a la capacidad de respuesta del chatbot. Este puede aportar información tanto general como especializada. Además de los documentos integrados y Wikipedia, el chatbot puede acceder a fuentes como Arxiv y DisGeNET, enriqueciendo así la información proporcionada.

La perspectiva y desarrollo del chatbot aportado por en el trabajo muestra la tendencia que se ha instaurado con respecto al desarrollo de más herramientas personalizadas para realizar consultas médicas, la gran mayoría con la participación de expertos en materias de salud (18). Estos chatbots especializados contrastan significativamente con ChatGPT, el bot más popular que encontramos hoy en día, el cual tiene un carácter más generalista y no puede funcionar al nivel de los chatbots médicos dedicados.

Esto se debe en parte a que las bases de datos y el entrenamiento de ChatGPT no se enfocan casi exclusivamente en informar y ayudar a los pacientes con dudas relacionadas con su salud. La mayoría de los conjuntos de datos de entrenamiento para los bots médicos son afinados y creados por profesionales de la salud, lo cual es crucial para su precisión y efectividad.

Esta situación puede generar preocupaciones, ya que muchos pacientes pueden recurrir a ChatGPT en busca de asistencia. Aunque esta herramienta tiene el potencial de educar y agilizar la atención, existe el riesgo de que proporcione diagnósticos o recomendaciones inexactas, lo que podría resultar perjudicial.

En cuanto al análisis de la literatura médica, los documentos frecuentemente implican dependencias sustanciales, como hipervínculos y referencias, permitiendo que el conocimiento se extienda a través de múltiples artículos. Para suplir esta carencia y establecer enlaces de conocimiento entre documentos, en ocasiones anteriores se ha realizado con un entrenamiento que abarcó dos dominios: el dominio general, utilizando artículos de Wikipedia con hipervínculos, y el dominio biomédico, empleando artículos de PubMed con enlaces de citas (19).

Los LLMs también pueden ser aprovechados para generar escenarios de casos o cuestionarios, ayudando a los estudiantes de medicina a practicar y refinar sus habilidades de diagnóstico y planificación de tratamientos dentro de un entorno seguro y controlado (20). La integración de LLMs en procesos de ludificación representa otra perspectiva cautivadora, ya que la mejora de herramientas para interactuar con el paciente podría llevar a mejorar el compromiso del paciente, proporcionar soporte personalizado y mejorar la gestión de enfermedades crónicas.

La perspectiva clave gira en torno a abordar las limitaciones de los LLMs, que incluyen desafíos como la desinformación, problemas de privacidad, sesgos en los datos de entrenamiento y el riesgo de uso indebido. El fenómeno de la alucinación puede propagar peligrosamente la desinformación médica o introducir sesgos que tienen el potencial de exacerbar las disparidades en salud (21). El objetivo general es garantizar la integridad del procesamiento del lenguaje natural (NLP). Este proceso integral se incorpora en cada etapa del ciclo de vida del desarrollo de NLP, con el fin de validar, verificar y hacer que los resultados sean confiables y explicables para los no expertos. Además, se subraya la importancia del despliegue ético, el aprendizaje imparcial y la equidad hacia los usuarios.

### Puntos fuertes y ventajas

Cabe destacar que los clínicos y el personal médico pasan varias horas cada día completando documentos, revisando comunicaciones electrónicas y realizando varias otras actividades que pueden resultar en agotamiento y en el desperdicio de recursos. El uso de estas herramientas de IA podría permitir a los clínicos centrar su atención en el cuidado de los pacientes. Por ejemplo, se pueden incorporar estas herramientas en la monitorización remota de pacientes, interpretando y explicando terminología médica compleja, lo que a su vez permite a los pacientes entender sus problemas médicos, las diversas opciones de tratamiento disponibles y tomar mayor responsabilidad sobre sus condiciones de salud (22).

Los investigadores también han indicado que la IA (y la tecnología asociada) se puede utilizar para promover el concepto de vida asistida ambiental, un medio que permitiría a los ancianos seguir viviendo en su hogar por un período prolongado. Hogares inteligentes equipados con diferentes sensores, alarmas y herramientas que pueden realizar análisis de comportamiento y robots de asistencia que pueden ayudar con diferentes actividades de la vida diaria o robots como RIBA que pueden ayudar a levantar y mover cosas pueden ayudar a los ancianos en casa. Las herramientas de IA también pueden ayudar con la estimulación cognitiva, lo que resulta en una mejor capacidad de atención, memoria y planificación. Esto es particularmente útil en los ancianos que experimentan deterioro en las habilidades cognitivas y enfrentan problemas para completar actividades o tareas complejas que requieren mantener la atención y utilizar habilidades de resolución de problemas (23).

### Implicaciones en la seguridad

Aunque el campo de la inteligencia artificial (IA) está creciendo y seguirá expandiéndose, la supervisión humana es extremadamente importante. Las organizaciones reguladoras, las instalaciones de atención médica y asociaciones profesionales deben colaborar y establecer sistemas para monitorear la progresión de las iniciativas de IA, las iniciativas de seguridad del paciente, los riesgos clínicos y cualquier error o evento adverso (24). No se puede negar la importancia de la contribución humana, la revisión y el papel de los profesionales humanos, incluso si las instalaciones están equipadas con robots quirúrgicos. Los requisitos personales de los pacientes juegan un papel crítico en la planificación de la atención. Un sistema de IA puede identificar un centro de atención específico basado en las necesidades físicas de los pacientes; sin embargo, también se deben considerar factores sociales, económicos y otros al asignar la atención. Solo es posible hacerlo si los proveedores de atención médica prestan atención a las necesidades personales de los pacientes.

Las leyes estrictas de privacidad, las regulaciones y los problemas de cumplimiento deben considerarse cuidadosamente cuando se incorporan sistemas de IA. No se puede comparar el sistema de salud con una compañía de teléfonos móviles u organizaciones como Amazon, por lo que el acceso a la información puede plantear desafíos y limitar la capacidad de la IA para ayudar con las prácticas médicas (25). Aunque la inclusión de la IA puede reducir costos y presión sobre los clínicos, también puede llevar a la eliminación de ciertos trabajos y plantear desafíos de equidad en el entorno de atención médica actual.

La IA en la medicina se basa en datos diagnósticos disponibles de una gran cantidad de casos clínicos que se recopilan a lo largo del tiempo. En ciertos casos clínicos o condiciones donde hay una disponibilidad limitada de datos, no se puede ignorar la posibilidad de diagnóstico erróneo o diagnóstico omitido. Por lo tanto, la intervención y experiencia humana son necesarias para ayudar con condiciones médicas complejas o raras (25). La incapacidad de la IA para ayudar con el diagnóstico y la planificación del tratamiento de ciertas condiciones puede llevar a la falta de aceptación tanto de los pacientes como de los proveedores.

Debido a que los sistemas de IA dependen de una red de datos, estos son propensos a amenazas de seguridad. Es importante considerar las posibilidades de ciberataques y poner en marcha mecanismos y procedimientos de seguridad apropiados para asegurar que los datos de los pacientes estén protegidos. Los beneficios de la aplicación de la IA en el diseño y la prestación de atención médica son ampliamente reconocidos, pero también debemos examinar los problemas éticos al usar datos de pacientes y clínicos. Cuatro problemas principales que deben abordarse incluyen:

* 1. El consentimiento informado para que los pacientes y proveedores utilicen los datos.
  2. Prácticas para asegurar la seguridad y transparencia.
  3. La equidad algorítmica y problemas de sesgo.
  4. Los estándares de privacidad.

Se necesita atención a estos factores para garantizar la implementación de un diseño de IA transparente y equitativo. La falta de transparencia de la IA se cita como una de las principales barreras para su implementación en prácticas clínicas y de salud en general (26).

El problema de la opacidad sobre cómo la IA llega a decisiones es extremadamente importante porque están en juego los problemas de salud y las vidas de los pacientes. Es crucial comprender cómo una entrada particular ha llevado a una salida específica y los diferentes factores que han contribuido a un diagnóstico particular. Esto no siempre es posible con la IA y los sistemas habilitados para IA. Para construir sistemas de IA confiables que puedan llevar a una adopción generalizada, se deben hacer esfuerzos para que las organizaciones cumplan con las leyes y regulaciones, las prácticas éticas y construyan sistemas seguros y confiables. Un informe reciente del Instituto Europeo de Innovación y Tecnología en Salud reconoció cómo la "IA detallada y ética" puede ayudar en la adopción de dichos sistemas en la práctica (27).

Esto puede ser facilitado por las asociaciones profesionales en todo el mundo. Más específicamente, organizaciones como la Federación Europea de Informática Médica, la Asociación Internacional de Informática Médica, la Asociación Americana de Informática Médica y el Instituto Nacional de Informática en Salud pueden jugar un papel importante en abordar los desafíos éticos mencionados anteriormente.

# Conclusiones

En conclusión, el chatbot basado en RAG y la base de datos vectorial que integra datos de NHANES con documentos médicos y de estilo de vida ha demostrado ser una herramienta prometedora para proporcionar recomendaciones personalizadas de estilo de vida a los usuarios. La combinación de técnicas de recuperación de información y generación de lenguaje natural nos permite aprovechar al máximo la riqueza de conocimientos disponibles y adaptarlos a las necesidades individuales de cada usuario.

Los resultados preliminares han sido alentadores, con altas puntuaciones de precisión y exhaustividad en la generación de recomendaciones relevantes, así como una retroalimentación positiva por parte de los usuarios. Sin embargo, también hemos identificado algunas limitaciones y desafíos que requieren atención adicional, como la integración efectiva de datos heterogéneos y la personalización basada en datos individuales limitados.

Para el trabajo futuro, nos proponemos abordar estas limitaciones y continuar mejorando el rendimiento y la usabilidad del chatbot. Algunas de las áreas clave de enfoque incluyen:

1. Explorar técnicas avanzadas de representación de conocimientos y razonamiento para mejorar la integración y la coherencia de los datos de NHANES con los documentos adicionales.
2. Desarrollar métodos de adquisición de datos más efectivos para recopilar información relevante de los usuarios y mejorar la personalización de las recomendaciones.
3. Realizar estudios de usuario a mayor escala para evaluar la efectividad y la aceptación del chatbot en diferentes poblaciones y contextos.
4. Investigar la integración de técnicas de aprendizaje por refuerzo para permitir que el chatbot aprenda y se adapte continuamente en función de las interacciones con los usuarios.
5. Explorar la posibilidad de extender el chatbot a otras áreas relacionadas con la salud y el bienestar, como la salud mental, la gestión de enfermedades crónicas y la prevención de enfermedades.

En resumen, nuestro chatbot basado en RAG y la base de datos vectorial que integra datos de NHANES con documentos médicos y de estilo de vida representa un avance significativo en la provisión de recomendaciones personalizadas de estilo de vida. Aunque aún hay trabajo por hacer, creemos que este enfoque tiene un gran potencial para mejorar la salud y el bienestar de las personas al brindarles acceso a información y consejos personalizados basados en evidencia. Esperamos continuar desarrollando y perfeccionando esta herramienta para ayudar a las personas a tomar decisiones informadas y adoptar estilos de vida más saludables.

Referencias bibliográficas

1. Egli A. ChatGPT, GPT-4, and Other Large Language Models: The Next Revolution for Clinical Microbiology? Clin Infect Dis. 11 de noviembre de 2023;77(9):1322-8.

2. Singhal K, Tu T, Gottweis J, Sayres R, Wulczyn E, Hou L, et al. Towards Expert-Level Medical Question Answering with Large Language Models [Internet]. arXiv; 2023 [citado 1 de junio de 2024]. Disponible en: http://arxiv.org/abs/2305.09617

3. Mökander J, Schuett J, Kirk HR, Floridi L. Auditing large language models: a three-layered approach. AI Ethics [Internet]. 30 de mayo de 2023 [citado 31 de marzo de 2024]; Disponible en: https://link.springer.com/10.1007/s43681-023-00289-2

4. Yu B. Evaluating Pre-Trained Language Models on Multi-Document Summarization for Literature Reviews.

5. Agapito G, Calabrese B, Guzzi PH, Cannataro M, Simeoni M, Care I, et al. DIETOS: A recommender system for adaptive diet monitoring and personalized food suggestion. En: 2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob) [Internet]. New York, NY: IEEE; 2016 [citado 1 de abril de 2024]. p. 1-8. Disponible en: http://ieeexplore.ieee.org/document/7763190/

6. Iwendi C, Khan S, Anajemba JH, Bashir AK, Noor F. Realizing an Efficient IoMT-Assisted Patient Diet Recommendation System Through Machine Learning Model. IEEE Access. 2020;8:28462-74.

7. Yang Z, Khatibi E, Nagesh N, Abbasian M, Azimi I, Jain R, et al. ChatDiet: Empowering Personalized Nutrition-Oriented Food Recommender Chatbots through an LLM-Augmented Framework [Internet]. arXiv; 2024 [citado 1 de abril de 2024]. Disponible en: http://arxiv.org/abs/2403.00781

8. Chang J, Wang S, Ling C, Qin Z, Zhao L. Gene-associated Disease Discovery Powered by Large Language Models [Internet]. arXiv; 2024 [citado 1 de abril de 2024]. Disponible en: http://arxiv.org/abs/2401.09490

9. Piñero J, Ramírez-Anguita JM, Saüch-Pitarch J, Ronzano F, Centeno E, Sanz F, et al. The DisGeNET knowledge platform for disease genomics: 2019 update. Nucleic Acids Res. 4 de noviembre de 2019;gkz1021.

10. LangChain [Internet]. [citado 1 de junio de 2024]. Disponible en: https://www.langchain.com/

11. Are WW. arXiv.org e-Print archive [Internet]. arXiv. [citado 1 de junio de 2024]. Disponible en: https://arxiv.org/

12. DisGeNET - a database of gene-disease associations [Internet]. [citado 1 de junio de 2024]. Disponible en: https://www.disgenet.org/

13. Wikipedia contributors. Wikipedia:Portada. En. Disponible en: https://es.wikipedia.org/w/index.php?title=Wikipedia:Portada&oldid=149134229

14. FastAPI [Internet]. [citado 1 de junio de 2024]. Disponible en: https://fastapi.tiangolo.com

15. Streamlit • A faster way to build and share data apps [Internet]. [citado 1 de junio de 2024]. Disponible en: https://streamlit.io

16. Groq builds the world’s fastest AI inference technology [Internet]. [citado 1 de junio de 2024]. Disponible en: https://groq.com

17. Welcome to flask — flask documentation (3.0.X) [Internet]. [citado 1 de junio de 2024]. Disponible en: https://flask.palletsprojects.com/en/3.0.x/

18. Chow JCL, Sanders L, Li K. Design of an Educational Chatbot Using Artificial Intelligence in Radiotherapy. AI. 2 de marzo de 2023;4(1):319-32.

19. Yasunaga M, Leskovec J, Liang P. LinkBERT: Pretraining Language Models with Document Links [Internet]. arXiv; 2022 [citado 2 de junio de 2024]. Disponible en: http://arxiv.org/abs/2203.15827

20. Khan RA, Jawaid M, Khan AR, Sajjad M. ChatGPT - Reshaping medical education and clinical management. Pak J Med Sci [Internet]. 7 de febrero de 2023 [citado 2 de junio de 2024];39(2). Disponible en: https://pjms.org.pk/index.php/pjms/article/view/7653

21. Zúñiga Salazar G, Zúñiga D, Vindel CL, Yoong AM, Hincapie S, Zúñiga AB, et al. Efficacy of AI Chats to Determine an Emergency: A Comparison Between OpenAI’s ChatGPT, Google Bard, and Microsoft Bing AI Chat. Cureus [Internet]. 18 de septiembre de 2023 [citado 2 de junio de 2024]; Disponible en: https://www.cureus.com/articles/183679-efficacy-of-ai-chats-to-determine-an-emergency-a-comparison-between-openais-chatgpt-google-bard-and-microsoft-bing-ai-chat

22. Morse S, Hagen J. Talkdesk talks about the benefits of being an early adopter of ChatGPT [Internet]. 2023 [citado 5 de junio de 2024]. Disponible en: https://www.healthcareitnews.com/news/talkdesk-talks-about-benefits-being-early-adopter-chatgpt

23. Bohr A, Memarzadeh K. The rise of artificial intelligence in healthcare applications. En: Artificial Intelligence in Healthcare [Internet]. Elsevier; 2020 [citado 5 de junio de 2024]. p. 25-60. Disponible en: https://linkinghub.elsevier.com/retrieve/pii/B9780128184387000022

24. Bajwa J, Munir U, Nori A, Williams B. Artificial intelligence in healthcare: transforming the practice of medicine. Future Healthc J. julio de 2021;8(2):e188-94.

25. Pros & cons of artificial intelligence in medicine [Internet]. 2024 [citado 5 de junio de 2024]. Disponible en: https://drexel.edu/cci/stories/artificial-intelligence-in-medicine-pros-and-cons/

26. He J, Baxter SL, Xu J, Xu J, Zhou X, Zhang K. The practical implementation of artificial intelligence technologies in medicine. Nat Med. enero de 2019;25(1):30-6.

27. Kiseleva A, Kotzinos D, De Hert P. Transparency of AI in Healthcare as a Multilayered System of Accountabilities: Between Legal Requirements and Technical Limitations. Front Artif Intell. 30 de mayo de 2022;5:879603.

Índice de acrónimos

API: Application Programming Interface

LLM: Large Language Model

PDF: Portable Document Format

CSV: Comma-separated value

JSON: JavaScript Object Notation

DOCX: Microsoft Word text document

FAISS: Facebook AI Similarity Search

RAG: Retrieval Augmented Generation

IA: Inteligencia Artificial

HTTP: Hypertext transfer protocol secure