



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



## TFG del Grado en Ingeniería Informática

Chatbot basado en *Large Language Models* y técnicas de *Retrieval-augmented Generation* para asistente de dudas sobre la realización del Trabajo Fin de Grado.



Presentado por José María Redondo Guerra  
en Universidad de Burgos — 14 de enero  
de 2024

Tutores: José Ignacio Santos Martín, Carlos  
López Nozal







UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



D. José Ignacio Santos Martín, profesor del departamento de Ingeniería de Organización, área de Organización de Empresas y D. Carlos López Nozal, profesor del departamento Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Exponen:

Que el alumno D. José María Redondo Guerra, con DNI 44906147G, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado “Chatbot basado en *Large Language Models* y técnicas de *Retrieval-augmented Generation* para asistente de dudas sobre la realización del Trabajo Fin de Grado”.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección de los que suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 14 de enero de 2024

Vº. Bº. del Tutor:

D. José Ignacio Santos Martín

Vº. Bº. del co-tutor:

D. Carlos López Nozal co-tutor





## Resumen

Este proyecto desarrolla un chatbot avanzado diseñado para proporcionar asistencia a estudiantes de Ingeniería Informática que se encuentran inmersos en la realización de sus **Trabajo de Fin de Grado (TFG)**. El chatbot se basa en **Large Language Models (LLM)**, como ChatGPT, y emplea técnicas de **Retrieval-augmented Generation (RAG)** para enriquecer su capacidad de respuesta y acceso a información relevante.

Este proyecto se caracteriza por su doble enfoque: en primer lugar, un análisis tecnológico exhaustivo de las distintas soluciones de implementación del chatbot utilizando LLM basados en **Inteligencia Artificial (IA)** generativa; en segundo lugar, su aplicación en un entorno específico, en este caso, la asistencia a estudiantes en la realización de sus **TFG** en la Universidad de Burgos. Se ha realizado la investigación y evaluación de soluciones tecnológicas probando tanto distintos modelos como la forma de acceder a los mismos.

Entre los modelos que fueron objeto de estudio, se optó por Mistral debido a su destacado rendimiento, la disponibilidad de una interfaz de programación de aplicaciones **API**, y su condición como empresa europea, lo que implica su sujeción a las leyes de protección de datos necesarias para este tipo de productos. Para la interacción con el Modelo de Lenguaje Grande **LLM** y la creación del **RAG**, se empleó LangChain, una elección que facilitó una implementación bastante sencilla, aunque se evidenciaron algunas limitaciones en su funcionamiento.

Esta solución tecnológica emergente exhibe resultados superiores en comparación con el chatbot basado en DialogFlow, incluso en sus etapas iniciales. Se anticipa que esta tecnología podría dominar el sector de los chatbots a medida que se perfeccionen aspectos como las alucinaciones, la validación de los **LLM** y las consideraciones relacionadas con la sostenibilidad ética y energética.

## Descriptores

*Chatbot, Large Language Models(LLM), Retrieval-augmented Generation(RAG), Procesamiento de lenguaje natural(NLP), Inteligencia artificial, Aprendizaje automático, Embeddings, Base de datos vectorial.*

## Abstract

This project focuses on the development of an advanced chatbot designed to provide assistance to Computer Engineering students who are immersed in the completion of their “Final Thesis” (**TFG**). The chatbot is based on *Large Language Models (LLM)*, such as ChatGPT, and uses *Retrieval-augmented Generation (RAG)* techniques to improve its responsiveness and access to relevant information.

This project is characterized by its twofold approach: first, an exhaustive technological analysis of the different chatbot implementation solutions using **LLM** based on “Generative Artificial Intelligence” (AI); second, its application in a specific environment, in this case, the assistance to students in the realization of their **TFG** at the University of Burgos. Research and evaluation of technological solutions has been carried out by testing different models and the way to access them.

Among the models under study, Mistral was chosen because of its outstanding performance, the availability of an **API**, and its status as a European company, which means that it is subject to the data protection laws required for this type of product. For the interaction with the **LLM** and the creation of the **RAG**, LangChain was used, a choice that facilitated a fairly straightforward implementation, although some limitations in its operation were evident.

This emerging technology solution exhibits superior results compared to the DialogFlow-based chatbot, even in its initial stages. It is anticipated that this technology could dominate the chatbot industry as aspects such as hallucinations, **LLM** validation, and ethical and energy sustainability considerations are refined.

## Keywords

Chatbot, Large Language Models(LLM), Retrieval-augmented Generation(RAG), Natural Language Processing(NLP), Artificial Intelligence, Machine Learning, Embeddings, Vector Database.

---

# Índice general

---

Índice general	iii
Índice de figuras	iv
Índice de tablas	v
<b>1. Introducción</b>	<b>1</b>
<b>2. Objetivos del proyecto</b>	<b>3</b>
<b>3. Conceptos teóricos</b>	<b>5</b>
3.1. <i>Large Language Models</i> . . . . .	6
3.2. <i>Retrieval-augmented Generation</i> . . . . .	32
3.3. <i>Prompt Engineering</i> . . . . .	38
3.4. TFG y Chatbot actual . . . . .	40
<b>4. Técnicas y herramientas</b>	<b>49</b>
4.1. Específicas de los LLM . . . . .	49
4.2. Prototipado, documentación y gestión . . . . .	61
4.3. Frontend . . . . .	67
<b>5. Aspectos relevantes del desarrollo del proyecto</b>	<b>71</b>
5.1. Selección de un LLM . . . . .	71
5.2. Fase temprana en la inteligencia artificial generativa . . . . .	75
5.3. Preprocesamiento de los documentos del TFG . . . . .	80
5.4. Validación y pruebas en el procesamiento del lenguaje natural	82
<b>6. Trabajos relacionados</b>	<b>85</b>

6.1.	Anteriores TFG de Chatbots . . . . .	85
6.2.	Estado del arte de los Chatbots basados en LLMs . . . . .	86
6.3.	Estado del arte de los Chatbots basados en LLMs . . . . .	87
6.4.	Comparativa con UBU-Chatbot . . . . .	88
<b>7.</b>	<b>Conclusiones y Líneas de trabajo futuras</b>	<b>97</b>
7.1.	El futuro de los LLM . . . . .	97
7.2.	Líneas de trabajo futuras . . . . .	98
7.3.	Reflexiones y conclusiones . . . . .	99

---

# Índice de figuras

---

3.1. Cronología de la evolución de los LLM . . . . .	8
3.2. Cronología de los modelos lingüísticos de gran tamaño (superior a 10B) de los últimos años . . . . .	11
3.3. Ilustración de un espacio de incrustación 2D tal que el vector que une “lobo” con “perro” es el mismo que el vector que une “tigre” con “gato”. . . . .	12
3.4. Arquitectura de los modelos de redes neuronales profundas <i>Transformers</i> . . . . .	15
3.5. Muestra de un texto y su equivalente en tokens usando el tokenizador de OpenAI para GPT-3.5 y GPT-4. . . . .	16
3.6. Ejemplo de codificación de pares de bytes aplicado a una secuencia de caracteres. . . . .	18
3.7. Cómo funciona self-attention: se calculan las puntuaciones de atención entre “estación” y cualquier otra palabra de la secuencia, y luego se utilizan para ponderar una suma de vectores de palabras que se convierte en el nuevo vector “estación”. . . . .	20
3.8. Clasificación de los LLM en tres categorías . . . . .	22
3.9. Evolución tecnológica de los modelos GPT a lo largo de los años. . . . .	23
3.10. Gráfico de la evolución de los modelos LLaMa . . . . .	24
3.11. Resultados en MMLU, Razonamiento de sentido común, Conocimiento del mundo y Comprensión lectora para Mistral 7B y Llama 2 (7B/13/70B). . . . .	28
3.12. Secuencia para la creación de un <i>Retrieval-augmented Generation</i> . . . . .	33
3.13. Esquema del funcionamiento de un <i>Retrieval-augmented Generation</i> . . . . .	33
3.14. Esquema del funcionamiento de creación de bases de datos vectoriales con Embeddings. . . . .	37

3.15. Proceso de gestión de TFG en la UBU. . . . .	41
3.16. UI del UBU-CHATBOT actual. . . . .	47
3.17. Limitaciones del UBU-CHATBOT actual relativas a la formulación de preguntas por parte de los estudiantes. . . . .	48
4.1. Esquema del <i>Framework</i> de Langchain con los distintos componentes y módulos. . . . .	52
4.2. Matriz de representación de los múltiples <i>Document Loaders</i> disponibles en LangChain. . . . .	55
4.3. Efecto de la <i>Quantization</i> en el tamaño de los LLM de LLaMa.	58
4.4. <i>Sprint Board</i> de Zube.io con los distintos estados de las <i>Issues</i> . .	64
5.1. Avances en el campo de la Inteligencia Artificial Generativa en los últimos meses. . . . .	76
5.2. Ejemplo de reporte de testeo de una posible configuración del RAG. . . . .	83
5.3. Ejemplo de la validación de Preguntas y Respuestas del chatbot y su respuesta generada. . . . .	84
6.1. Ejemplo de pregunta de tutoría administrativa, saludo, del chatbot de DialogFlow. . . . .	89
6.2. Ejemplo de pregunta de tutoría administrativa, saludo, del chatbot basado en LLM y RAG. . . . .	89
6.3. Ejemplo de pregunta de tutoría administrativa, profesores del tribunal, del chatbot de DialogFlow. . . . .	90
6.4. Ejemplo de pregunta de tutoría administrativa, profesores del tribunal, del chatbot basado en LLM y RAG. . . . .	91
6.5. Ejemplo de pregunta de tutoría administrativa, profesores del tribunal pregunta compleja, del chatbot de DialogFlow. . . .	92
6.6. Ejemplo de pregunta de tutoría administrativa, profesores del tribunal pregunta compleja, del chatbot basado en LLM y RAG. .	92
6.7. Ejemplo de pregunta de tutoría general del chatbot basado en LLM y RAG. . . . .	93
6.8. Ejemplo de pregunta de tutoría general del chatbot basado en LLM y RAG. . . . .	94
6.9. Ejemplo de pregunta de tutoría general del chatbot basado en LLM y RAG. . . . .	95
6.10. Ejemplo de pregunta de tutoría específicas del chatbot basado en LLM y RAG. . . . .	96

---

## Índice de tablas

---

3.1. Plataformas de desarrollo de Chatbots. . . . .	46
5.1. Comparativa entre distintos <i>Large Language Models</i> . . . . .	75



---

# 1. Introducción

---

Como parte del grado en Ingeniería Informática, es necesaria la realización de un **Trabajo de Fin de Grado (TFG)** que marca el cierre de la etapa de formación. Sin embargo, este proyecto no está exento de desafíos, y los estudiantes a menudo se enfrentan a una multitud de preguntas y obstáculos, desde la selección del tema adecuado hasta la presentación final del proyecto [?]. La complejidad de este proceso, combinada con la necesidad de optimizar los tiempos y la orientación, ha dado lugar a una necesidad de herramientas y recursos que asistan a los estudiantes en esta parte de su etapa académica [?].

En respuesta a esta demanda, este proyecto se centra en el desarrollo de un asistente virtual innovador: un chatbot basado en **Large Language Models (LLM)** que tiene como objetivo proporcionar respuestas precisas y adaptadas y orientación en tiempo real a los estudiantes que comienzan a realizar el **TFG** en Ingeniería Informática [?].

La base de este chatbot reside en su capacidad para comprender y procesar preguntas complejas, así como en su habilidad para generar respuestas coherentes y relevantes. Para mejorar aún más su eficacia y precisión, se implementarán técnicas de **Retrieval-augmented Generation (RAG)** que permitirán enriquecer las respuestas del chatbot al aprovechar la información contenida en documentos existentes sobre **TFG**. Información que será debidamente embebida en vectores(en el espacio del modelo) y almacenada en una base de datos vectorizada. Este enfoque combina la potencia de los **LLM** con la precisión de la recuperación de información **RAG**, creando así un asistente virtual altamente competente para este caso de uso.

Además, en aras de evaluar y verificar la eficacia de este nuevo chatbot, se llevará a cabo una comparación con un chatbot preexistente desarrollado

en un proyecto de fin de grado anterior. Este análisis nos permitirá identificar las mejoras y ventajas de nuestra nueva aproximación.

En resumen, este proyecto aborda la problemática de acceder a una gran cantidad de información que no es de dominio general a través de la combinación de **LLM** y técnicas **RAG**. Se espera no solo crear un chatbot que pueda ser utilizado para este caso de uso sino también generar un proceso que pueda ser aplicado a otros casos de uso similares en los que exista una gran cantidad de información específica y que no sea de dominio público.

---

## 2. Objetivos del proyecto

---

El objetivo fundamental de este proyecto radica en el desarrollo de un asistente virtual, específicamente un chatbot basado en *Large Language Models (LLM)*, como ChatGPT, Bard o LLama. Este chatbot se concibe como una herramienta avanzada que tiene como misión principal proporcionar orientación, respuestas a preguntas comunes y asistencia en las diferentes fases de la realización de un TFG en Ingeniería Informática. Más allá de la simple automatización de respuestas, este chatbot se diseña con la ambición de convertirse en un recurso confiable y eficaz que brinde apoyo en tiempo real a los estudiantes, facilitándoles la toma de decisiones informadas y la superación de obstáculos.

Sin embargo, la singularidad de este proyecto reside en su enfoque hacia la mejora continua de la precisión y relevancia de las respuestas proporcionadas por el chatbot. Para lograr esto, se implementarán técnicas de *Retrieval-augmented Generation (RAG)*. Estas técnicas permitirán que el chatbot enriquezca sus respuestas mediante la recuperación de información específica sobre **TFG** contenida en documentos previamente existentes. Estos documentos son PDFs con la regulación de la **UBU**, repositorios de **TFG** previos e información en la página web de la universidad entre otros. Esta combinación de **LLM** y **RAG** garantizará que las respuestas sean no solo coherentes y útiles, sino también contextualmente relevantes.

Además de diseñar y desarrollar este chatbot, este proyecto se propone un objetivo de comparación. Se buscará evaluar y contrastar el rendimiento y la eficacia de este nuevo chatbot con un chatbot preexistente que ha sido desarrollado en un proyecto de fin de grado anterior. A través de este análisis comparativo, se pretende identificar y documentar las mejoras y ventajas

que esta nueva aproximación aporta al proceso de realización de **TFG** en Ingeniería Informática.

En resumen, los objetivos de este proyecto son tres:

1. Diseño y desarrollo del chatbot basado en **LLM**: El primer objetivo es diseñar y desarrollar un chatbot que utilice *Large Language Models* para responder a las preguntas y dudas de los estudiantes en relación con la realización de sus **Trabajo de Fin de Grado** en Ingeniería Informática.
2. Integración de técnicas de **RAG** para mejora de la precisión: El segundo objetivo es integrar técnicas de *Retrieval-augmented Generation* en el chatbot. Estas técnicas se utilizarán para mejorar la precisión y relevancia de las respuestas del chatbot, aprovechando la información contenida en documentos existentes relacionados con los **TFG**.
3. Comparación de rendimiento con chatbot preexistente: El tercer objetivo implica comparar el rendimiento y la eficacia del chatbot desarrollado en este proyecto con un chatbot preexistente que fue creado en un proyecto de fin de grado anterior. Esto permitirá identificar las mejoras y ventajas de la nueva aproximación.

En un enfoque más general, mediante la creación de un chatbot que aprovecha las capacidades de los **LLM** y la implementación de técnicas **RAG**, se pretende explorar y ampliar las posibles aplicaciones de esta tecnología en diversos sectores. La capacidad de proporcionar respuestas altamente precisas y contextualmente relevantes, respaldadas por datos específicos, presenta un gran potencial que se extiende más allá de los límites de este proyecto académico. Estas aplicaciones pueden ser especialmente beneficiosas en sectores como empresas privadas, administración pública y otros contextos donde la información precisa y la interacción automatizada son fundamentales.

---

### 3. Conceptos teóricos

---

Este apartado de conceptos teóricos desempeña un papel importante al condensar los fundamentos esenciales de principios, teorías y términos subyacentes en el dominio de conocimiento relacionado con el proyecto. En este contexto, su propósito principal es brindar una visión panorámica de los conceptos teóricos cruciales que servirán como cimiento para la comprensión y avance de este **Trabajo de Fin de Grado**. A través de una exposición minuciosa de estos conceptos, se proporciona una base conceptual que permite una comprensión más profunda de su aplicación práctica en el proyecto. Asimismo, se busca definir una serie de términos y establecer una base de conocimiento compartida para las secciones subsiguientes.

Especialmente importante son los conceptos de los *Large Language Models* (**LLM**) y las técnicas de *Retrieval-augmented Generation* (**RAG**). Estas nociones son muy actuales y están en constante evolución en el ámbito del procesamiento de lenguaje natural y la inteligencia artificial, desempeñan un papel fundamental en la comprensión y avance del proyecto en consideración.

Los **LLM** representan un hito significativo en el campo de la generación de texto y la comprensión del lenguaje. Modelos como GPT-3 han demostrado una capacidad sin precedentes para comprender y generar texto de manera coherente y relevante, convirtiéndose en herramientas poderosas con diversas aplicaciones.

Las técnicas de **RAG** amplían aún más el potencial de los **LLM** al permitir el acceso a información específica en documentos o bases de conocimiento existentes. Esta capacidad de recuperación y generación mejorada se traduce en respuestas más precisas y contextualmente relevantes, lo que resulta particularmente valioso en situaciones que requieren asistencia y generación de contenido.

En el núcleo de los **LLM** y las técnicas de **RAG**, se encuentran otros conceptos fundamentales de embeddings y bases de datos vectoriales. Estos conceptos son la columna vertebral que impulsa la capacidad de estos modelos para entender y generar texto de manera efectiva. Los embeddings, representaciones vectoriales de palabras y frases, permiten a los **LLM** comprender y procesar el lenguaje natural, capturando la semántica y relaciones entre palabras. Por otro lado, las bases de datos vectoriales almacenan información en un espacio vectorial, lo que habilita la recuperación eficiente de información relevante. La interacción sinérgica de estos conceptos permite a los **LLM** y las técnicas **RAG** acceder a conocimiento específico y generar respuestas contextualmente enriquecidas, mejorando la precisión y relevancia en la comunicación y generación de contenido.

### **3.1. Large Language Models**

#### **¿Qué es un Large Language Models?**

En esencia, un modelo lingüístico de gran escala es un tipo de modelo de aprendizaje automático que puede comprender y generar lenguaje humano mediante redes neuronales profundas (*Deep Neural Networks*). La principal tarea de un modelo lingüístico es calcular la probabilidad de que una palabra siga a una entrada dada en una frase: por ejemplo, “El cielo es ....”, siendo la respuesta más probable “azul”. El modelo es capaz de predecir la siguiente palabra de una frase tras ser entrenado con un amplio conjunto de datos de texto (o *corpus*). Básicamente, aprende a reconocer distintos patrones en las palabras. De este proceso se obtiene un modelo lingüístico preentrenado.

Mediante la afinación del modelo (reentrenamiento con nuevos datos) se mejora la capacidad de hacer tareas muy diversas, como la traducción o la adquisición de conocimientos especializados en un campo concreto, como el Derecho o la Medicina. Este proceso se conoce como aprendizaje por transferencia, que permite a un modelo aplicar los conocimientos adquiridos de una tarea a otra.

Lo que hace que un modelo lingüístico sea “grande” es el tamaño de su arquitectura. Ésta, a su vez, se basa en la inteligencia artificial de las redes neuronales, muy parecidas al cerebro humano, donde las neuronas trabajan juntas para aprender de la información y procesarla. Además, los **LLM** constan de un gran número de parámetros (por ejemplo, **GPT** tiene más de 100.000 millones) entrenados en grandes cantidades de datos de texto sin etiquetar mediante aprendizaje autosupervisado o semisupervisado.

Con el primero, los modelos son capaces de aprender a partir de texto no anotado, lo que supone una gran ventaja si se tienen en cuenta los costosos inconvenientes de tener que depender de datos etiquetados manualmente.

Además, las redes más grandes y con más parámetros han demostrado un mejor rendimiento, con una mayor capacidad para retener información y reconocer patrones en comparación con sus homólogas más pequeñas. Cuanto mayor es el modelo, más información puede aprender durante el proceso de entrenamiento, lo que a su vez hace que sus predicciones sean más precisas. Aunque esto puede ser cierto en el sentido convencional, hay una salvedad: tanto las empresas de IA como los desarrolladores están encontrando formas de sortear los retos que plantean los excesivos costes computacionales y la energía necesaria para entrenar los LLM introduciendo modelos más pequeños y entrenados de forma más óptima.

Aunque los LLM se han entrenado principalmente para tareas sencillas, como predecir la siguiente palabra de una frase, es asombroso ver la cantidad de estructura y significado del lenguaje que han sido capaces de captar, por no mencionar el enorme número de datos que pueden recoger.

### **Historia y desarrollo de los LLM**

La historia y evolución de los LLM se remontan a varias décadas de investigación y desarrollo en el campo del procesamiento de lenguaje natural y la Inteligencia Artificial, como se ve en la figura 3.1. A continuación, se proporciona un resumen de los hitos más significativos en la historia de los LLM [?, ?, ?].

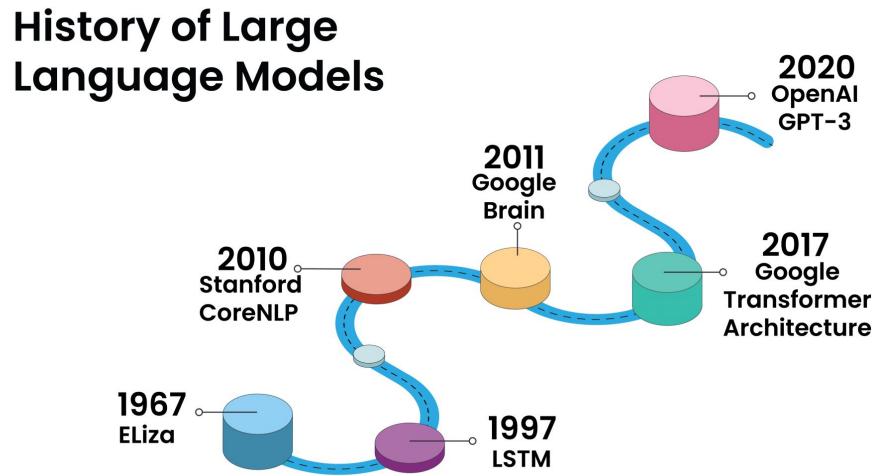


Figura 3.1: Cronología de la evolución de los LLM

**Década de 1950-1960 :** Los primeros pasos en la creación de modelos de lenguaje se remonta a los experimentos pioneros con redes neuronales y sistemas de procesamiento de información neuronal realizados en la década de 1950 con el propósito de permitir a las computadoras comprender y procesar el lenguaje natural. Colaboraciones entre investigadores de IBM y la Universidad de Georgetown dieron lugar a la creación de un sistema capaz de traducir automáticamente frases del ruso al inglés, lo que marcó un hito notable en la traducción automática. A partir de ese punto, la investigación en este campo experimentó un auge significativo. Durante esta misma época, se dieron los primeros pasos en el desarrollo de modelos de lenguaje, con investigadores dedicados a la creación de reglas gramaticales y algoritmos para analizar y generar texto. Sin embargo, estos enfoques iniciales se basaban en reglas manuales y presentaban limitaciones significativas en cuanto a su efectividad.

La idea de los **LLM** surgió con la creación de *Eliza* en los años 60, fue el primer chatbot del mundo, diseñado por el investigador del **MIT** Joseph Weizenbaum. *Eliza* marcó el inicio de la investigación sobre el **Procesamiento del Lenguaje Natural (PLN)** y sentó las bases para futuros **LLM** más complejos.

**Década de 1980-1990 :** Se produjeron avances en el **Procesamiento del Lenguaje Natural (PLN)** con la introducción de modelos estadísticos y técnicas de aprendizaje automático. Modelos como el modelo de lenguaje de Markov oculto (HMM) se convirtieron en populares para tareas de **PLN**. Una de las innovaciones más significativas fue la introducción de las redes de memoria a largo plazo (LSTM) en 1997, que permitieron crear redes neuronales más profundas y complejas, capaces de manejar cantidades de datos más significativas.

**Década de 2000-2010 :** Otro momento crucial fue la suite CoreNLP de Stanford, introducida en 2010. Esta suite ofrecía un conjunto de herramientas y algoritmos que ayudaban a los investigadores a abordar tareas de **PLN** complejas, como el análisis de sentimientos y el reconocimiento de entidades con nombre. Surgieron también modelos estadísticos más avanzados, como los modelos de lenguaje basados en **Máquinas de soporte vectorial (SVM)** y las **Cadenas de Markov Condicionales (CRF)**. Estos modelos mejoraron la capacidad de procesar y generar texto de manera más efectiva.

**Década de 2010-2020 :** Esta década marcó un hito significativo con la llegada de modelos basados en redes neuronales, especialmente modelos de lenguaje recurrente (RNN) y modelos de lenguaje basados en *Transformers*. En 2011, Google Brain hizo su debut, proporcionando a los investigadores un acceso invaluable a recursos informáticos de gran potencia y conjuntos de datos enriquecidos, además de ofrecer características avanzadas, como la incrustación de palabras. Esta innovación permitió a los sistemas de **Procesamiento del Lenguaje Natural** comprender el contexto de las palabras de manera más efectiva.

El trabajo pionero de Google Brain sentó las bases para avances significativos en el campo, incluyendo la aparición de los modelos *Transformer* en 2017. La arquitectura de los *Transformers* revolucionó la creación de **Large Language Models (LLM)** más grandes y sofisticados, ejemplificados por el **Generative Pre-trained Transformer (GPT)** de OpenAI. Uno de los modelos más influyentes es el GPT-1 desarrollado por OpenAI en 2018.

GPT-2, una versión más grande y avanzada de GPT-1, causó revuelo en la comunidad de inteligencia artificial debido a su capacidad para generar texto coherente y de alta calidad. OpenAI inicialmente decidió no publicar GPT-2 debido a preocupaciones sobre el uso malicioso.

Fue en 2019 cuando los investigadores de Google presentaron BERT, el modelo bidireccional de 340 millones de parámetros (el tercer modelo

más grande de su clase) que podía determinar el contexto permitiéndole adaptarse a diversas tareas. Al preentrenar a BERT en una amplia variedad de datos no estructurados mediante aprendizaje autosupervisado, el modelo pudo comprender las relaciones entre las palabras. En poco tiempo, BERT se convirtió en la herramienta de referencia para las tareas de procesamiento del lenguaje natural. De hecho, BERT estaba detrás de todas las consultas en inglés realizadas a través de Google Search.

**2020 en adelante** : GPT-3, lanzado por OpenAI, marcó un avance significativo en el campo de los **LLM**. Con 175 mil millones de parámetros, GPT-3 demostró una sorprendente capacidad para comprender y generar texto de manera coherente. Se convirtió en un modelo base para muchas aplicaciones de procesamiento de lenguaje natural y impulsó el movimiento basado “*Generative AI*” al gran público, especialmente a través de la *interface* Chat-GPT.

Desde GPT-3, la investigación en **LLM** ha continuado avanzando. Se han desarrollado modelos aún más grandes y efectivos, y se han aplicado a una amplia gama de aplicaciones, desde asistentes virtuales hasta traducción automática y generación creativa de texto o imagen.

La versión más reciente hasta la fecha es GPT-4, que presenta mejoras significativas, como la capacidad de utilizar visión por ordenador para interpretar datos visuales (a diferencia de ChatGPT, que utiliza GPT-3.5). GPT-4 acepta como entrada tanto texto, como imágenes.

Y lo que es más, el último avance es la *steerability* (direcciónabilidad), que permite a los usuarios de **GPT** personalizar la estructura de su salida para satisfacer sus necesidades específicas. Básicamente, la direcciónabilidad alude a la capacidad de controlar o modificar el comportamiento de un modelo lingüístico, lo que implica hacer que el **LLM** adopte distintos roles, siga instrucciones del usuario o hable con un tono determinado. La direcciónabilidad permite al usuario cambiar el comportamiento de un **LLM** a voluntad y ordenarle que escriba con un estilo o una voz diferentes. Las posibilidades son infinitas.

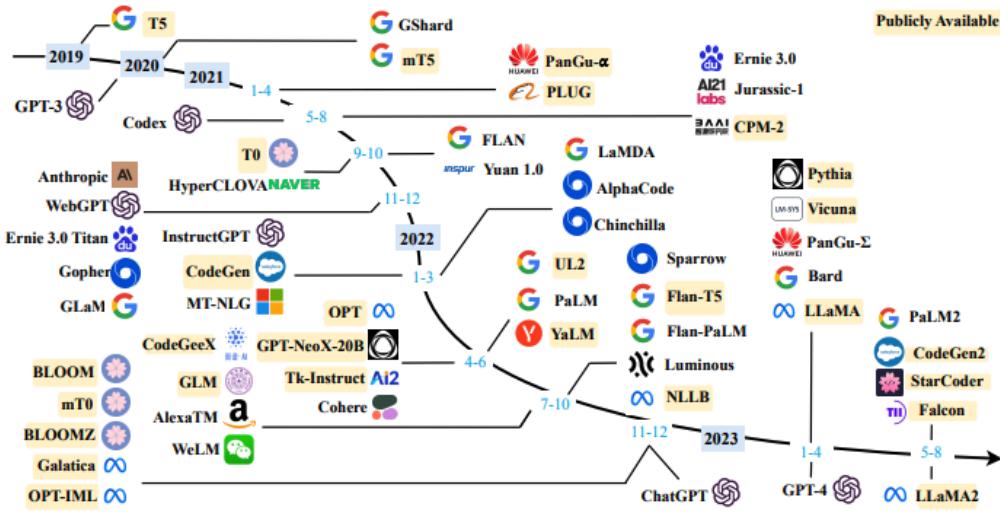


Figura 3.2: Cronología de los modelos lingüísticos de gran tamaño (superior a 10B) de los últimos años

Es importante destacar que la evolución de los **LLM** ha sido impulsada en gran medida por el aumento en la disponibilidad de datos de entrenamiento, el desarrollo de arquitecturas de redes neuronales más avanzadas y la mejora en el hardware de cómputo. Estos avances han permitido a los **LLM** alcanzar un nivel de comprensión y generación de texto que antes era impensable. Esto ha causado que en los últimos años se hayan lanzado multitud de productos y modelos como se ve en la figura 3.2.

## Word2Vec

Un precursor de los actuales *Large Language Models* que merece la pena destacar es Word2Vec. Word2Vec fue un algoritmo revolucionario en el **Procesamiento del Lenguaje Natural (PLN)** que se emplea para aprender representaciones vectoriales densas de palabras a partir de grandes cantidades de texto. Desarrollado por un equipo de investigadores de Google en 2013 [?], este enfoque no supervisado ha tenido un impacto significativo en el campo del **PLN**. Construyeron un modelo para incrustar palabras en un espacio vectorial, un problema que ya contaba con una larga historia académica en aquel momento, que comenzaba en la década de 1980. Su modelo utilizaba un objetivo de optimización diseñado para convertir las relaciones de correlación entre palabras en relaciones de distancia en el espacio de incrustación: se asociaba un vector a cada palabra de un vocabulario, y los

vectores se optimizaban para que el producto punto (proximidad del coseno) entre vectores que representaban palabras que coincidían con frecuencia estuviera más cerca de 1, mientras que el producto punto entre vectores que representaban palabras que rara vez coincidían estuviera más cerca de 0. Descubrieron que el espacio de incrustación resultante era un espacio vectorial.

El espacio de incrustación resultante hacía mucho más que captar la similitud semántica. Presentaba alguna forma de aprendizaje emergente: era capaz de realizar “aritmética de palabras”, algo para lo que no había sido entrenado, como se ve en la figura 3.3. Existía un vector en el espacio que podía añadirse a cualquier sustantivo masculino para obtener un punto cercano a su equivalente femenino. Por ejemplo,  $V(\text{rey}) - V(\text{hombre}) + V(\text{mujer}) = V(\text{reina})$ . Un “vector de género”. Esta capacidad de realizar operaciones matemáticas en el espacio de incrustación reveló una comprensión subyacente de las relaciones semánticas. Parecía haber docenas de vectores mágicos de este tipo: un vector plural, un vector para pasar de nombres de animales salvajes a su equivalente más cercano en mascotas, y muchos otros. Estos descubrimientos abrieron nuevas perspectivas en el campo del procesamiento de lenguaje natural y subrayaron la potencia de Word2Vec en la representación de palabras [?].

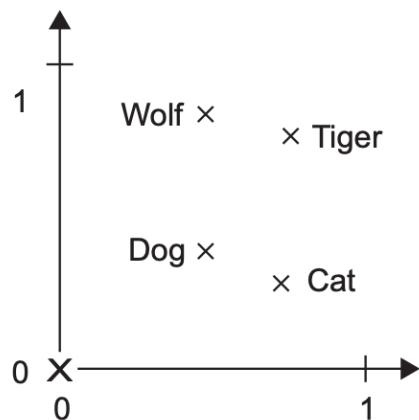


Figura 3.3: Ilustración de un espacio de incrustación 2D tal que el vector que une “lobo” con “perro” es el mismo que el vector que une “tigre” con “gato”.

Word2Vec se implementa en dos arquitecturas principales: *Continuous Bag of Words (CBOW)* y *Skip-gram*. El modelo **CBOW** se utiliza para

predecir una palabra objetivo basada en un contexto circundante, mientras que el modelo *Skip-gram* realiza predicciones inversas, es decir, predice palabras de contexto a partir de una palabra dada. Estas arquitecturas han demostrado ser altamente efectivas en la generación de representaciones vectoriales de palabras que capturan significados y relaciones con precisión.

Las representaciones de palabras aprendidas con Word2Vec se han convertido en una pieza fundamental en tareas de transferencia de conocimiento en **PLN**, lo que ha impulsado su amplia adopción en la comunidad de investigación y desarrollo. A través de su capacidad para reducir la dimensionalidad y su habilidad para mejorar el rendimiento en tareas de procesamiento de lenguaje, Word2Vec ha dejado una huella perdurable en la forma en que las computadoras comprenden y utilizan el lenguaje natural en una variedad de aplicaciones.

## Cómo funciona un LLM

Un *Large Language Models* (**LLM**) es un tipo de modelo lingüístico que destaca por su capacidad de comprensión y generación de lenguaje de propósito general. Los **LLM** adquieren estas capacidades utilizando cantidades masivas de datos, llamados *corpus*, para ajustar miles de millones de parámetros durante el entrenamiento y consumiendo grandes recursos computacionales durante su formación y funcionamiento [?]. Los **LLM** son redes neuronales artificiales, principalmente *transformers* [?], y se preentrenan utilizando aprendizaje autosupervisado y aprendizaje semisupervisado.

Como modelos autorregresivos del lenguaje, funcionan tomando un texto de entrada y prediciendo repetidamente el siguiente token o palabra [?]. Hasta 2020, el *fine-tuning* era la única forma de adaptar un modelo para que pudiera realizar tareas específicas. Sin embargo, los modelos de mayor tamaño, como el GPT-3, pueden diseñarse con *prompt-engineering* para lograr resultados similares. Se cree que adquieren conocimientos incorporados sobre sintaxis, semántica y “ontología” inherentes a los del lenguaje humano, pero también imprecisiones y sesgos presentes en los en el mismo.

### **Transformers**

Los *Transformers* son una arquitectura de *Deep Neural Networks* (DNN) que ha revolucionado el campo del **Procesamiento del Lenguaje Natural (PLN)** y ha impulsado el desarrollo de los *Large Language Models* (**LLM**). Estas arquitecturas se destacan por su capacidad para modelar relaciones y

dependencias de largo alcance en datos secuenciales, como texto, de manera altamente eficiente [?, ?].

La clave de los *Transformers* radica en su estructura de auto-atención, que permite que el modelo procese secuencias de entrada de manera paralela y capture relaciones entre palabras en diferentes posiciones de la secuencia [?] como se puede ver en la figura 3.4. A continuación, se describe cómo funcionan los *Transformers* aplicados a los LLM:

1. **Codificación de entrada:** La secuencia de entrada (por ejemplo, una oración o un párrafo) se descompone en una serie de vectores de palabras o tokens. Cada palabra se representa como un vector de números reales.
2. **Capas de atención:** El corazón de la arquitectura de los *Transformer* es la capa de atención. Esta capa calcula la atención entre todas las palabras en la secuencia de entrada. La atención es una medida de cuánta importancia se le asigna a cada palabra en función de su relación con otras palabras en la secuencia. Esta atención se calcula a través de una función de similitud, que pondera las conexiones entre las palabras.
3. **Codificación posicional:** Aunque las capas de atención capturan relaciones entre palabras, los *Transformers* también necesitan información sobre la posición de las palabras en la secuencia. Para lograr esto, se agrega información de codificación posicional a los vectores de palabras.
4. **Apilamiento de capas:** Los *Transformers* constan de múltiples capas de atención. Cada capa refina las representaciones de las palabras y las relaciones entre ellas. Esto se hace de manera repetitiva para capturar relaciones cada vez más complejas y contextos más amplios.
5. **Decodificación:** Una vez que la entrada se ha procesado a través de las capas de atención, el modelo puede generar una salida. En el caso de los LLM, esta salida es una secuencia de palabras que forman una respuesta coherente a una pregunta o una generación de texto.
6. **Entrenamiento:** Los *Transformers* se entranan utilizando grandes conjuntos de datos que contienen pares de entrada y salida. El modelo ajusta sus parámetros para minimizar la diferencia entre las respuestas generadas y las respuestas esperadas en el conjunto de datos de entrenamiento.

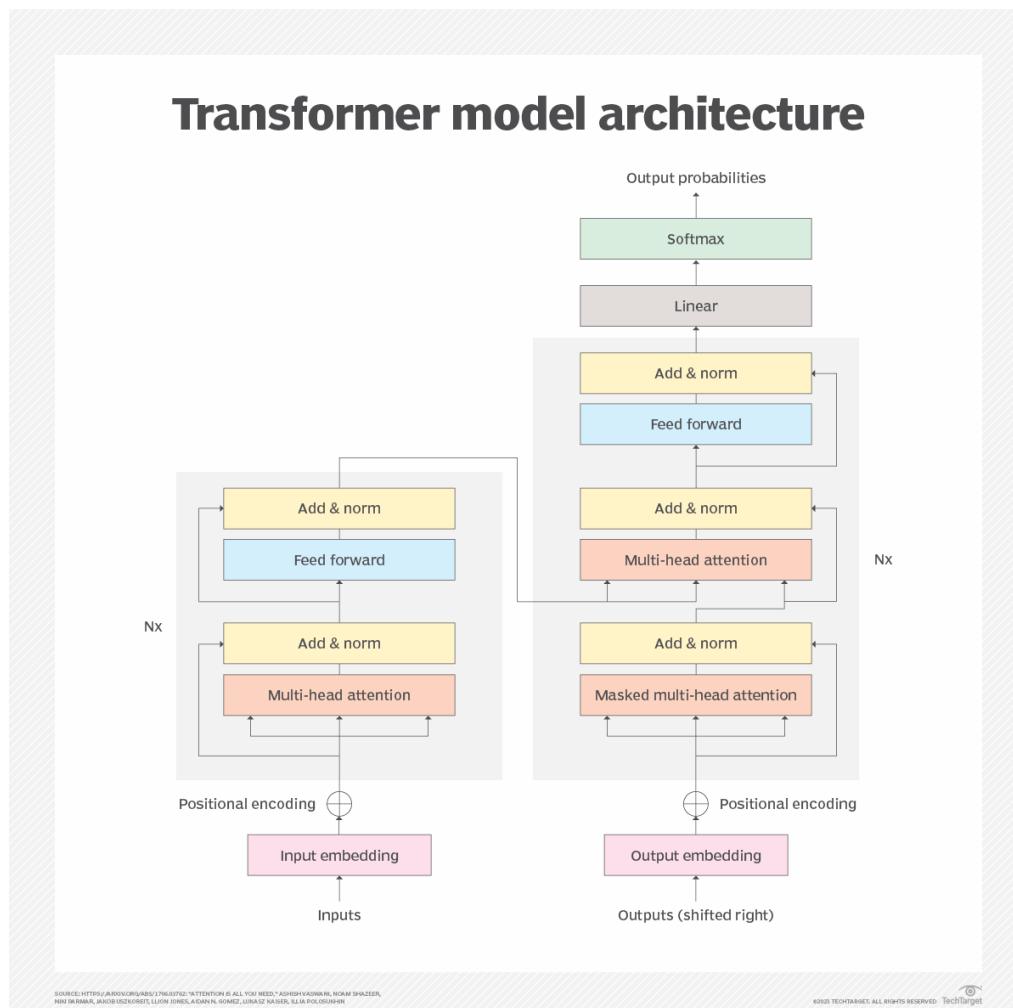


Figura 3.4: Arquitectura de los modelos de redes neuronales profundas *Transformers*.

### Procesamiento de datos de entrenamiento - tokenización

Los *Large Language Models* procesan el texto utilizando tokens, que son secuencias comunes de caracteres que se encuentran en un conjunto de texto. Los modelos aprenden a entender las relaciones estadísticas entre estos tokens y destacan a la hora de producir el siguiente token en una secuencia de tokens [?]. Un ejemplo se puede ver en la figura 3.5.

**GPT-3.5 & GPT-4    GPT-3 (Legacy)**

Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing the underlying bytes: 

Sequences of characters commonly found next to each other may be grouped together: 1234567890

**Tokens      Characters**  
57      252

Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing the underlying bytes: 

Sequences of characters commonly found next to each other may be grouped together: 1234567890

**TEXT      TOKEN IDS**

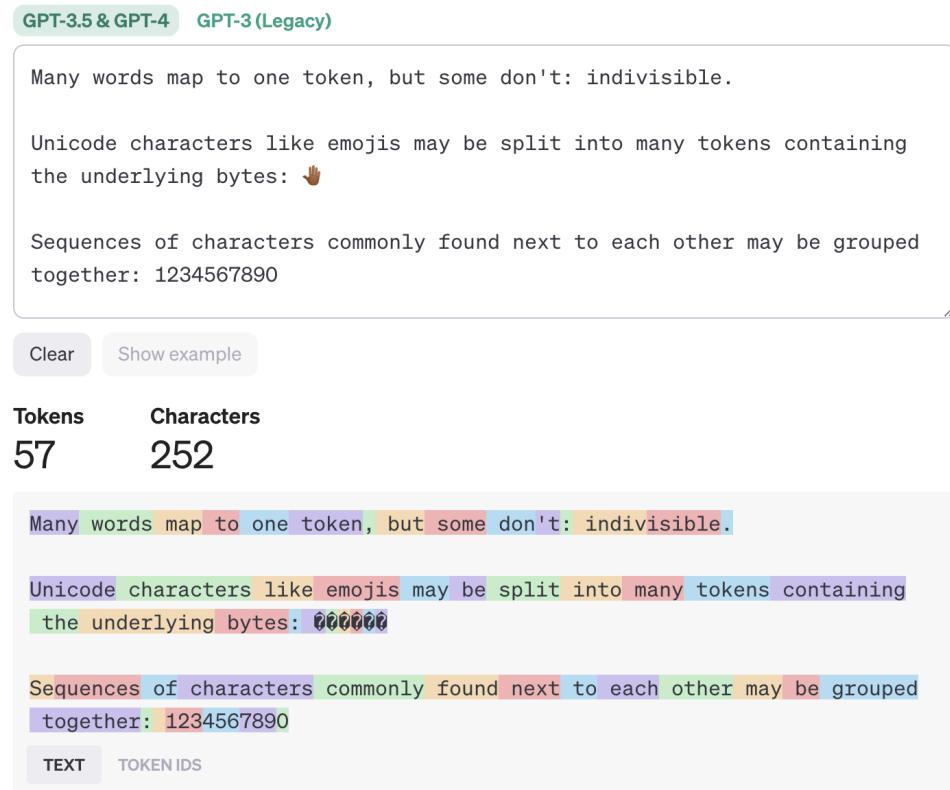


Figura 3.5: Muestra de un texto y su equivalente en tokens usando el tokenizador de OpenAI para GPT-3.5 y GPT-4.

Existen dos conceptos que están en la base de la tokenización de los **LLM**: y los N-grama y la Codificación de pares de bytes.

N-grama es una serie de n letras adyacentes (incluidos los signos de puntuación y los espacios en blanco), sílabas o, rara vez, palabras enteras que se encuentran en un conjunto de datos lingüísticos; o fonemas adyacentes extraídos de un conjunto de datos de grabación del habla, o pares de bases adyacentes extraídos de un genoma. Se recogen de un *corpus* de texto o de habla. Si se utilizan prefijos numéricos latinos, el n-grama de tamaño 1 se denomina “unigrama”, el de tamaño 2 “bigrama”, etc. Si, en lugar de los latinos, se utilizan además los números cardinales ingleses, entonces se denominan “four-gram”, “five-gram”, etc.

La codificación de pares de bytes (también conocida como codificación de diagramas) es un algoritmo, descrito por primera vez en 1994 por Philip Gage [?]. Su modificación destaca por ser el tokenizador de modelos de

lenguaje de gran tamaño con capacidad para combinar tanto tokens que codifican caracteres únicos (incluidos dígitos únicos o signos de puntuación únicos) como aquellos que codifican palabras completas (incluso las palabras compuestas más largas). Esta modificación, en el primer paso, supone que todos los caracteres únicos son un conjunto inicial de n-gramas de 1 carácter de longitud (es decir, tokens iniciales). A continuación, el par más frecuente de caracteres adyacentes se fusiona sucesivamente en un nuevo n-grama de 2 caracteres de longitud y todas las instancias del par se sustituyen por este nuevo token. Esto se repite hasta obtener un vocabulario del tamaño prescrito. Tenga en cuenta que siempre se pueden construir palabras nuevas a partir de los tokens del vocabulario final y de los caracteres del conjunto inicial.

Todos los tokens únicos encontrados en un *corpus* se enumeran en un vocabulario de tokens cuyo tamaño, en el caso de GPT-3, es de 50257.

La diferencia entre el algoritmo modificado y el original es que el algoritmo original no fusiona el par más frecuente de bytes de datos, sino que los sustituye por un nuevo byte que no estaba contenido en el conjunto de datos inicial. Véase el ejemplo de la figura 3.6. Para reconstruir el conjunto de datos inicial se necesita una tabla de búsqueda de las sustituciones. El algoritmo es eficaz para la tokenización porque no requiere grandes gastos de cálculo y sigue siendo coherente y fiable.

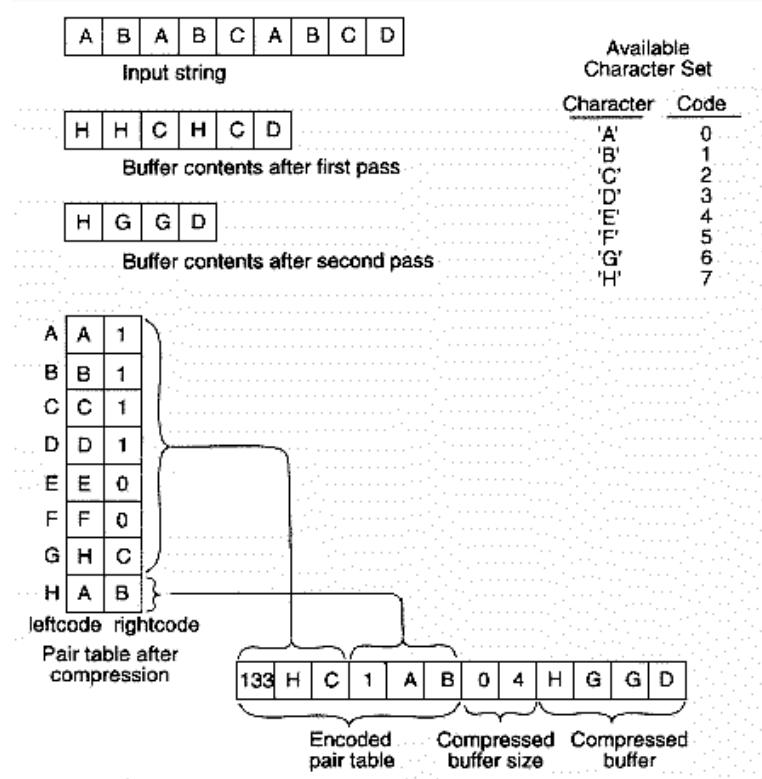


Figura 3.6: Ejemplo de codificación de pares de bytes aplicado a una secuencia de caracteres.

Un vocabulario de tokens basado en las frecuencias extraídas principalmente del inglés, utiliza el menor número posible de tokens para una palabra media en inglés. Sin embargo, una palabra media en otro idioma codificada por un tokenizador optimizado para el inglés se divide en una cantidad subóptima de tokens. Como regla general, un token suele corresponder a unos 4 caracteres de texto en inglés común. Esto equivale aproximadamente a tres cuartas partes de una palabra (por tanto, 100 tokens = 75 palabras).

Adicionalmente a la tokenización, el procesamiento de datos intenta crear datos de entrenamiento de alta calidad. La eliminación de pasajes tóxicos del conjunto de datos, el descarte de datos de baja calidad y la desduplicación son ejemplos de limpieza de conjuntos de datos de entrenamiento. Los conjuntos de datos limpios (de alta calidad) resultantes contenían hasta 17 billones de palabras en 2022, frente a los 985 millones de palabras utilizados en 2018 para GPT-1 y los 3.300 millones de palabras utilizados para BERT. No

obstante, se espera que los datos futuros estén cada vez más “contaminados” por los propios contenidos generados por **LLM**.

### ***Self-Attention***

En el contexto de la memoria del proyecto, es relevante abordar la aparente contradicción entre los ***Large Language Models (LLM)***, que son modelos autorregresivos entrenados para predecir la siguiente palabra en función de la secuencia de palabras anteriores, y el objetivo de Word2Vec de maximizar el producto punto entre tokens co-ocurrentes.

La conexión entre estos dos enfoques se establece a través del componente fundamental de la arquitectura *Transformer*: la autoatención(*self-attention*). La autoatención es un mecanismo clave para aprender un nuevo espacio de incrustación de tokens. Funciona mediante la recombinación lineal de incrustaciones de tokens del espacio anterior, dando mayor peso a los tokens que están más cerca en términos de su producto punto, es decir, aquellos que están más correlacionados.

Con el tiempo, este proceso conduce a la creación de un espacio en el cual las relaciones de correlación entre los tokens se convierten en relaciones de proximidad de incrustación, medida en términos de distancia coseno. Los Transformers, al aprender una serie de espacios de incrustación cada vez más refinados, logran dos propiedades cruciales:

**Continuidad Semántica:** Los espacios de incrustación son semánticamente continuos, lo que significa que un pequeño cambio en el espacio de incrustación apenas afecta el significado humano de los tokens correspondientes. Esta propiedad también se verifica en el espacio de Word2Vec.

**Interpolación Semántica:** Los espacios de incrustación son semánticamente interpolativos. Tomar el punto intermedio entre dos puntos de un espacio de incrustación da como resultado un punto que representa el “significado intermedio” entre los tokens correspondientes. Esto se logra al construir cada nuevo espacio de incrustación interpolando entre vectores del espacio anterior.

Esta dinámica no difiere mucho del proceso de aprendizaje en el cerebro, donde las relaciones de correlación entre eventos de disparo neuronal se convierten en relaciones de proximidad en la red cerebral, siguiendo el principio clave del aprendizaje Hebbiano:“neuronas que disparan juntas,

conectan junta". Tanto *Transformers* como Word2Vec son, de alguna manera, mapas de un espacio de información, buscando representar de manera eficiente y semánticamente significativa las relaciones entre los elementos del lenguaje. Se puede ver una representación gráfica de este proceso en la figura 3.7.

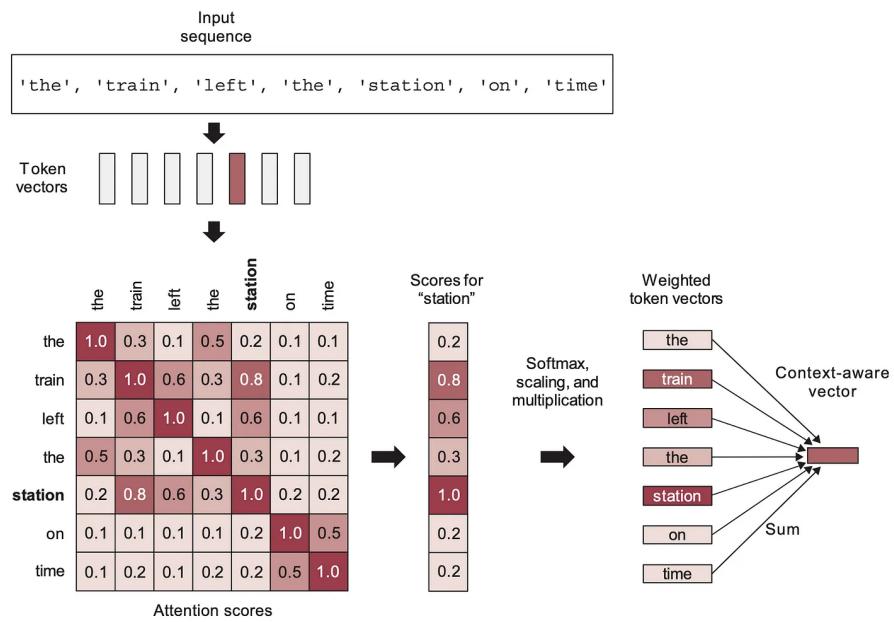


Figura 3.7: Cómo funciona self-attention: se calculan las puntuaciones de atención entre “estación” y cualquier otra palabra de la secuencia, y luego se utilizan para ponderar una suma de vectores de palabras que se convierte en el nuevo vector “estación”.

### Los LLM como bases de datos de programas

Se puede conceptualizar un *Large Language Models* (LLM) como algo análogo a una base de datos: almacena información que puedes recuperar mediante solicitudes. Sin embargo, existen dos diferencias fundamentales entre los LLM y las bases de datos convencionales.

La primera diferencia radica en que un LLM representa un tipo de base de datos continua e interpolativa. En lugar de almacenar datos como un conjunto discreto de entradas, la información se guarda como un espacio vectorial, es decir, una curva. La semántica de esta curva es continua, permitiéndote desplazarte a lo largo de ella para explorar puntos cercanos y

relacionados. Además, se puede interpolar en la curva entre distintos puntos de datos para encontrar puntos intermedios. Esto implica que se puede recuperar de la base de datos más información de la que originalmente se introdujo, aunque no todo será necesariamente preciso ni significativo. La interpolación puede conducir a la generalización, pero también a posibles alucinaciones.

La segunda diferencia es que un **LLM** no solo contiene datos en el sentido tradicional. Aunque ciertamente almacena una amplia gama de datos, como hechos, lugares, personas, fechas y relaciones, es, quizás principalmente, una base de datos de programas.

Estos programas en un **LLM** no se asemejan exactamente a los programas deterministas que se pueden encontrar en lenguajes de programación como Python, compuestos por secuencias de instrucciones simbólicas que procesan datos paso a paso. En cambio, los programas en un **LLM** son funciones altamente no lineales que mapean el espacio de incrustación latente en sí mismo. Son análogos a los vectores de Word2Vec, pero considerablemente más complejos en su naturaleza y capacidad.

## Tipos de LLM

Los *Large Language Models* pueden clasificarse a grandes rasgos en tres tipos: modelos de preentrenamiento, modelos de *Fine-tuning* y modelos multimodales [?], ver figura 3.8.

**Los modelos de preentrenamiento**, como GPT-3/GPT-3.5, T5 y XL-Net, se entrena con grandes cantidades de datos, lo que les permite aprender una amplia gama de patrones y estructuras lingüísticas. Estos modelos destacan en la generación de textos coherentes y gramaticalmente correctos sobre una gran variedad de temas. Se utilizan como punto de partida para seguir entrenándolos y perfeccionándolos para tareas específicas.

**Los modelos de *Fine-tuning***, como BERT, RoBERTa y ALBERT, se entrena previamente en un gran conjunto de datos y luego se ajustan en un conjunto de datos más pequeño para una tarea específica. Estos modelos son muy eficaces para tareas como el análisis de sentimientos, la respuesta a preguntas y la clasificación de textos. Suelen utilizarse en aplicaciones industriales que requieren modelos lingüísticos específicos para cada tarea.

**Los modelos multimodales** como CLIP y DALL-E combinan texto con otras modalidades como imágenes o vídeo para crear modelos lingüísticos más robustos. Estos modelos pueden entender las relaciones entre imágenes y texto, lo que les permite generar descripciones textuales de imágenes o incluso generar imágenes a partir de descripciones textuales.

Cada tipo de **LLM** tiene sus puntos fuertes y débiles, y la elección de cuál utilizar depende del caso de uso específico.

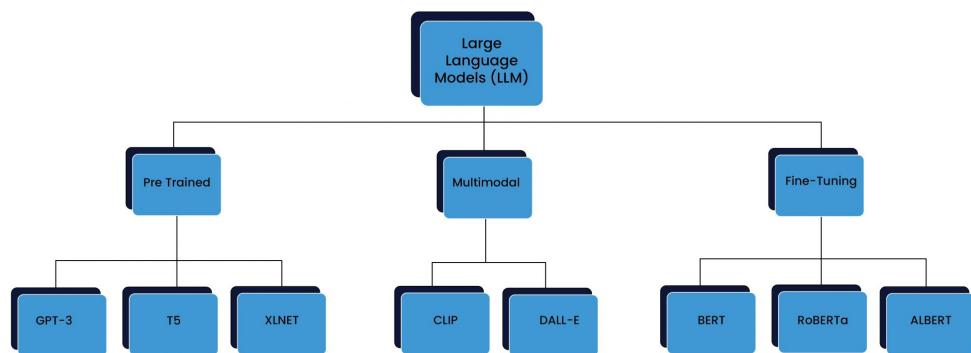


Figura 3.8: Clasificación de los LLM en tres categorías

## LLM más extendidos

### OpenAI - ChatGPT

OpenAI es pionera en el campo de la **Inteligencia Artificial**, es una de las empresas mas influyentes en el avance de los límites del procesamiento del lenguaje similar al humano. OpenAI ha lanzado numerosos modelos lingüísticos, figura 3.9, incluida toda la familia **GPT**, como GPT-3 y GPT-4, que impulsan su producto ChatGPT, y que han revolucionado la forma de trabajar de desarrolladores, investigadores y entusiastas de todo el mundo. En el área de los grandes modelos lingüísticos, es imposible pasar por alto el significativo impacto y el espíritu pionero de OpenAI, que sigue a día de hoy marcando el futuro de la inteligencia artificial.

Los modelos OpenAI han acaparado una gran atención por sus impresionantes características y su rendimiento de vanguardia. Estos modelos

poseen notables capacidades de comprensión y generación de lenguaje natural. Destacan en una amplia gama de tareas relacionadas con el lenguaje, como completar textos, traducir o responder preguntas, entre otras.

La familia de modelos **GPT**, incluidos gpt-4 y gpt-3.5-turbo, se ha entrenado con datos de Internet, códigos, instrucciones y comentarios humanos, con más de cien mil millones de parámetros, lo que garantiza la calidad de los modelos. Los modelos de OpenAI están diseñados para ser versátiles y atender a una amplia gama de casos de uso, incluida la generación de imágenes. Se puede acceder a ellos a través de una **API**, lo que permite a los desarrolladores integrar los modelos en sus aplicaciones. OpenAI ofrece distintas opciones de uso, entre ellas el ajuste fino, que permite a los usuarios adaptar los modelos a tareas o dominios específicos aportando datos de entrenamiento personalizados.

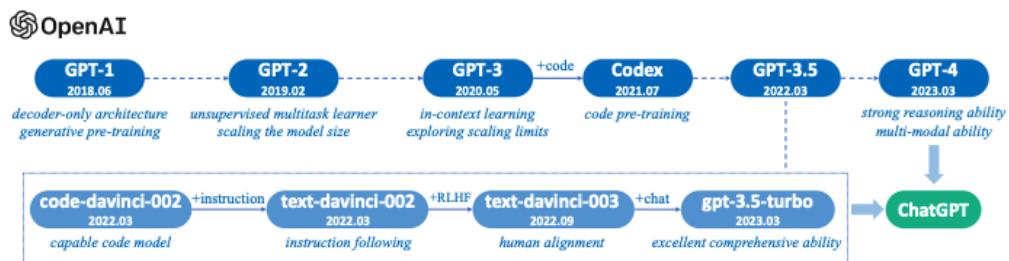


Figura 3.9: Evolución tecnológica de los modelos GPT a lo largo de los años.

OpenAI ha estado a la vanguardia del avance de los modelos de **Procesamiento del Lenguaje Natural (PLN)**, siendo pionera en el desarrollo del **Reinforcement Learning from Human Feedback (RLHF)** como una poderosa técnica para moldear el comportamiento de sus modelos en contextos de chat. El **RLHF** consiste en entrenar los modelos de IA combinando la retroalimentación generada por humanos con métodos de aprendizaje por refuerzo. De este modo, los modelos de OpenAI aprenden de las interacciones con los humanos para mejorar sus respuestas. Gracias al **RLHF**, OpenAI ha logrado avances significativos en la mejora de la fiabilidad, utilidad y seguridad de sus modelos, proporcionando en última instancia a los usuarios respuestas más precisas y adecuadas al contexto.

### Meta - LLaMa

Meta AI avanza significativamente en la promoción de la ciencia abierta con el lanzamiento de LLaMA (*Large Language Model Meta AI*). Este

modelo de lenguaje grande de última generación está diseñado para facilitar el progreso de los investigadores en el campo de la IA.

Los modelos más pequeños pero de alto rendimiento de LLaMA ofrecen accesibilidad a la comunidad de investigación, permitiendo a los investigadores sin recursos extensos explorar y estudiar estos modelos, democratizando así el acceso en este campo de rápido desarrollo. Estos modelos base, entrenados con grandes cantidades de datos no etiquetados, requieren menos potencia informática y recursos, lo que los hace ideales para el ajuste fino y la experimentación en diversas tareas.

LLaMA es una colección de modelos de lenguaje grandes que abarcan un amplio rango de parámetros, desde 7B hasta 65B. A través de un entrenamiento meticuloso con trillones de tokens extraídos exclusivamente de conjuntos de datos públicamente disponibles, los desarrolladores de LLaMA demuestran la posibilidad de lograr un rendimiento de vanguardia sin necesidad de fuentes de datos propietarias o inaccesibles. Especialmente, LLaMA-13B muestra un rendimiento superior en comparación con el renombrado GPT-3 (175B) en varios puntos de referencia, mientras que LLaMA-65B compite de manera impresionante con modelos de primer nivel como PaLM-540B [?].

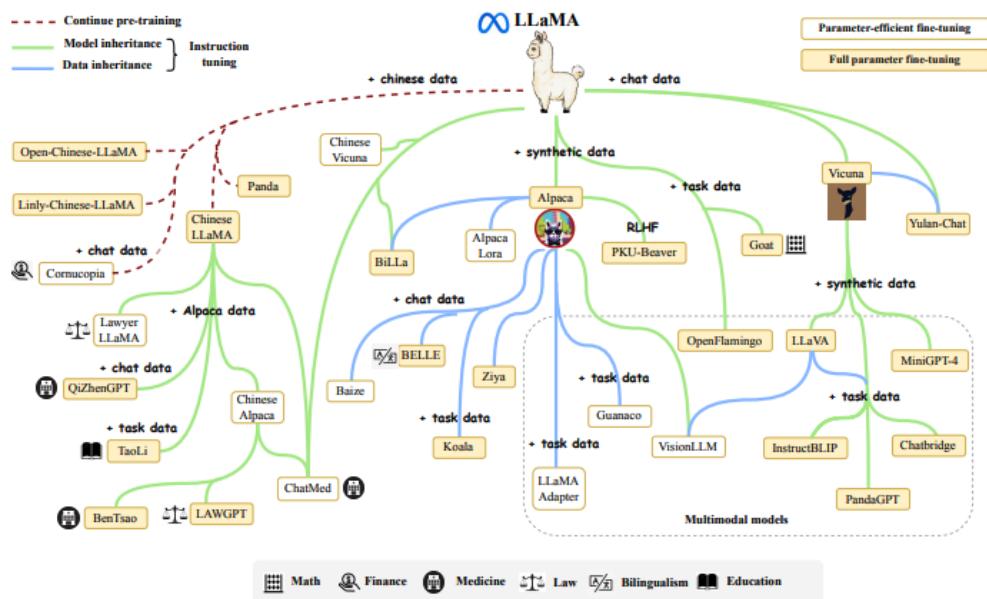


Figura 3.10: Gráfico de la evolución de los modelos LLaMa

Los modelos LLaMA aprovechan la arquitectura de *transformers*, que se ha convertido en el estándar de la industria para la modelización del lenguaje desde 2018, ver figura 3.10. En lugar de aumentar únicamente el número de parámetros, los desarrolladores de LLaMA priorizaron escalar el rendimiento del modelo al expandir significativamente el volumen de datos de entrenamiento. Su razonamiento se basa en la comprensión de que el costo principal de los grandes modelos de lenguaje radica en la inferencia durante el uso del modelo, más que en los gastos computacionales de entrenamiento. En consecuencia, LLaMA se entrenó con impresionantes 1.4 billones de tokens, obtenidos minuciosamente de datos públicamente disponibles. Este extenso conjunto de datos de entrenamiento capacita a LLaMA para destacar en la comprensión de patrones de lenguaje complejos y generar respuestas contextualmente apropiadas.

### Google - Bard

Google AI presentó BARD, un avanzado *Large Language Models (LLM)* en forma de chatbot, desarrollado y entrenado con un extenso conjunto de datos de texto y código. BARD exhibe su destreza en la generación de texto, la traducción multilingüe, la creación de código, la diversificación de contenido y al proporcionar respuestas informativas a preguntas.

Lo que diferencia a BARD es su capacidad para acceder a datos del mundo real a través de Google Search, lo que amplía sus capacidades de comprensión y respuesta. Google, a la vanguardia de la investigación en *LLM*, ha introducido modelos innovadores como BERT, T5 y PaLM. Estos modelos aprovechan arquitecturas basadas en *transformers*, lo que permite una comprensión contextual profunda del lenguaje y su aplicación versátil en tareas de *Procesamiento del Lenguaje Natural*.

BERT, un hito temprano, utiliza *transformers* bidireccionales para la comprensión contextual del texto, preentrenado en vastos datos no etiquetados y ajustado para tareas específicas. T5, adoptando un enfoque de aprendizaje por transferencia de texto a texto, demuestra su versatilidad en diversas tareas de *Procesamiento del Lenguaje Natural* sin necesidad de entrenamiento específico para cada tarea. PaLM se centra en capturar estructuras sintácticas y semánticas en las oraciones, utilizando características lingüísticas como árboles de análisis para relaciones sintácticas y etiquetado semántico de roles para identificación de funciones. Con la capacidad de escalar hasta 540 mil millones de parámetros, PaLM logra un rendimiento innovador.

Los modelos de lenguaje de Google han demostrado de manera consistente capacidades avanzadas y un rendimiento excepcional al abordar diversos desafíos en el **PLN**. El camino de Google en la investigación de **LLM**, comenzando con la arquitectura *Transformer*, ha allanado el camino para modelos cada vez más sofisticados, estableciendo estándares en la comprensión y generación de lenguaje.

### **Anthropic - Claude**

Anthropic es una organización que busca abordar algunos de los desafíos más profundos en **Inteligencia Artificial** y dar forma al desarrollo de sistemas de **IA** avanzados. Con un enfoque en la robustez, la seguridad y la alineación de valores, Anthropic tiene como objetivo abordar consideraciones éticas y sociales críticas en torno a la **IA**.

Claude, el producto estrella de Anthropic, es un **Large Language Model** de vanguardia que se encuentra en la vanguardia de la investigación en **Procesamiento del Lenguaje Natural**. Este modelo, nombrado en honor al legendario matemático Claude Shannon, representa un avance significativo en las capacidades del lenguaje de la **IA**.

El modelo Claude de Anthropic es un potente **LLM** diseñado para procesar grandes volúmenes de texto y realizar una amplia gama de tareas. Con Claude, los usuarios pueden gestionar fácilmente diversas formas de datos textuales, como documentos, correos electrónicos, preguntas frecuentes, transcripciones de chat y registros. El modelo ofrece una multitud de capacidades, como edición, reescritura, resumen, clasificación, extracción de datos estructurados y servicios de preguntas y respuestas basados en el contenido.

La familia de modelos de Anthropic, que incluye a Claude y Claude-instant, ha sido entrenada con datos de Internet, códigos, instrucciones y retroalimentación humana, lo que garantiza la calidad de los modelos.

Además del procesamiento de texto, Claude puede participar en conversaciones naturales, asumiendo diversos roles en un diálogo. Al especificar el rol y proporcionar una sección de preguntas frecuentes, los usuarios pueden tener interacciones fluidas y contextualmente relevantes con Claude. Ya sea un diálogo en busca de información o un escenario de juego de roles, Claude puede adaptarse y responder de manera naturalista.

Anthropic destaca algunas de las características sobresalientes de Claude, como “un extenso conocimiento general perfeccionado a partir de su vasto corpus de entrenamiento, con antecedentes detallados en conocimientos

técnicos, científicos y culturales. Claude puede hablar una variedad de idiomas comunes, así como lenguajes de programación” [?].

Además, Claude ofrece capacidades de automatización, lo que permite a los usuarios optimizar sus flujos de trabajo. El modelo puede ejecutar varias instrucciones y escenarios lógicos, incluido el formateo de salidas según requisitos específicos, siguiendo instrucciones condicionales y realizando una serie de evaluaciones lógicas. Esto permite a los usuarios automatizar tareas repetitivas y aprovechar la eficiencia de Claude para mejorar la productividad. Recientemente, se introdujo una nueva versión de Claude, que ofrece un impresionante límite de 100,000 tokens de contexto. Con esta capacidad ampliada, ahora se pueden incorporar sin esfuerzo libros completos o documentos extensos, abriendo emocionantes posibilidades para usuarios que buscan información integral o detalladas sugerencias creativas.

El modelo Claude de Anthropic introduce una función conocida como “inteligencia artificial constitucional”, que implica un proceso de dos fases: aprendizaje supervisado y aprendizaje por refuerzo. Aborda los posibles riesgos y daños asociados con sistemas de **Inteligencia Artificial** que utilizan retroalimentación. Al incorporar los principios del aprendizaje constitucional, tiene como objetivo controlar el comportamiento de la **IA** de manera más precisa.

### Mistral

Mistral AI, con sede en París y cofundada por ex trabajadores de Google DeepMind y Meta, anunció su primer modelo de lenguaje grande, Mistral 7B, recientemente. Esta startup logró asegurar una financiación inicial récord incluso antes de lanzar un producto. El primer modelo de Mistral AI con 7 mil millones de parámetros supera el rendimiento de Llama 2 13B en todas las pruebas y supera a Llama 1 34B en muchos aspectos métricos [?], ver figura 3.11.

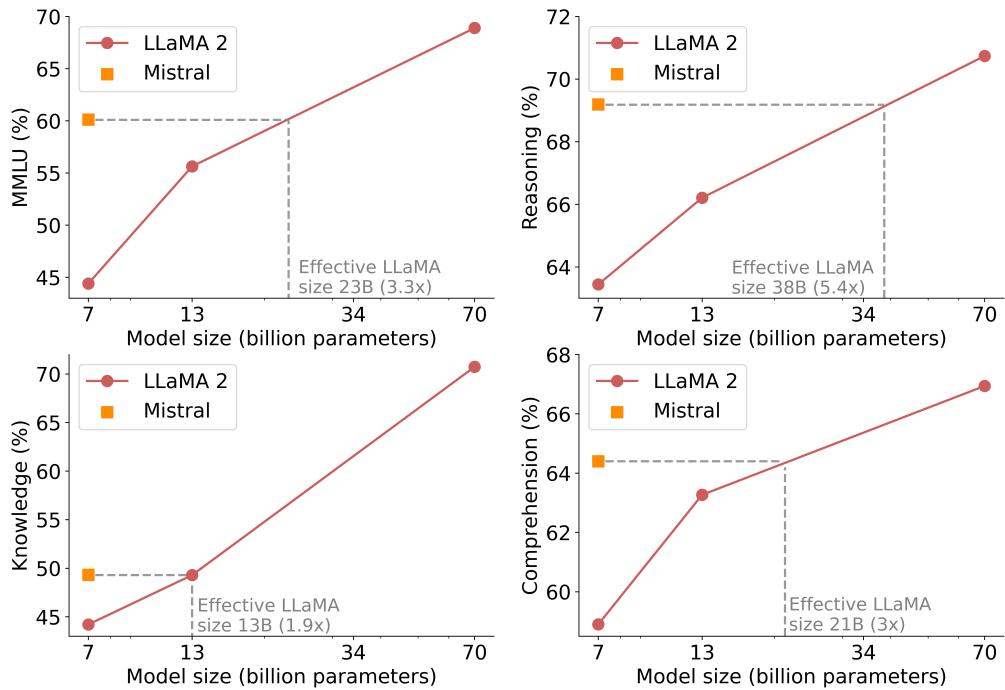


Figura 3.11: Resultados en MMLU, Razonamiento de sentido común, Conocimiento del mundo y Comprensión lectora para Mistral 7B y Llama 2 (7B/13B/70B).

En comparación con otros modelos como Llama 2, Mistral 7B ofrece capacidades similares o mejores pero con menos carga computacional. Mientras que modelos fundacionales como GPT-4 pueden lograr más, pero teniendo un costo más alto.

En lo que respecta a las tareas de codificación, Mistral 7B compite eficazmente con CodeLlama 7B. Además, es lo suficientemente compacto, con 13.4 GB, como para ejecutarse en máquinas estándar.

Además, Mistral 7B Instruct, ajustado específicamente para conjuntos de datos instructivos en Hugging Face, ha mostrado un gran rendimiento. Supera a otros modelos de 7B en MT-Bench y se destaca junto a los modelos de chat de 13B.

El nacimiento de *Large Language Models* de código abierto como Mistral 7B señala un cambio fundamental en la industria de la **Inteligencia Artificial**, haciendo que modelos de lenguaje de alta calidad sean accesibles para un público más amplio. Enfoques innovadores como el de Mistral AI,

como *Grouped-query attention* y *Sliding Window Attention*, prometen un rendimiento eficiente sin comprometer la calidad [?].

Aunque la naturaleza descentralizada de Mistral presenta ciertos desafíos, su flexibilidad y licencia de código abierto subrayan el potencial de democratizar la **Inteligencia Artificial**. A medida que el panorama evoluciona, el enfoque inevitablemente se centrará en equilibrar el poder de estos modelos con consideraciones éticas y mecanismos de seguridad.

El equipo de Mistral tiene como objetivo lanzar modelos aún más grandes próximamente. Si estos nuevos modelos igualan el rendimiento del 7B, Mistral podría ascender rápidamente como y jugar un papel destacado en la industria.

## Otros LLM

Existe un gran número de **LLM** y no dejan de salir nuevos modelos casi a diario. A mayores de los anteriormente comentados caben destacar también los siguientes.

Salesforce ha desarrollado el modelo *Conditional Transformer Language Model* (CTRL), un logro destacado en el procesamiento del lenguaje natural con 1.6 mil millones de parámetros. CTRL permite un control preciso sobre la generación de texto, con la capacidad de atribuir fuentes al texto generado, facilitando la comprensión de las influencias en la salida del modelo. Su entrenamiento con más de 50 códigos de control proporciona a los usuarios un manejo detallado del contenido y estilo del texto generado, mejorando la interacción humano-IA y mostrando potencial para mejorar otras aplicaciones de **Procesamiento del Lenguaje Natural**.

Dolly, desarrollado por Databricks, es un impresionante **Large Language Models (LLM)** diseñado para uso comercial y basado en el modelo pythia-12b. Dolly 2.0, una versión de código abierto, destaca por su capacidad de seguir instrucciones con precisión y ofrece interactividad similar a ChatGPT. Basado en un conjunto de datos de alta calidad, Dolly 2.0 es completamente accesible a la personalización, lo que lo hace que se pueda usar comercialmente sin necesidad de **API**, marcando un enfoque flexible y transparente. Además, Dolly no busca competir con modelos generativos de vanguardia, y Databricks proporciona el conjunto de datos de entrenamiento para permitir su uso y expansión por parte de la comunidad.

Cohere es un modelo de lenguaje grande desarrollado por una startup canadiense del mismo nombre. Este **LLM** de código abierto se entrena con un conjunto de datos diverso e inclusivo, lo que lo convierte en un experto en el manejo de numerosos idiomas y acentos. Además, los modelos de Cohere

se entranan con un corpus de texto grande y diverso, lo que los hace más eficaces para manejar una amplia gama de tareas.

## Limitaciones de los LLM

Junto con los increíbles avances en **Inteligencia Artificial**, modelos de aprendizaje automático y, en general, modelos de lenguaje, todavía existen numerosos desafíos por superar. La desinformación, el *malware*, el contenido discriminatorio, el plagio y la información simplemente falsa pueden conducir a resultados no deseados o peligrosos; esto pone en cuestión como de confiables son estos modelos.

Además, cuando se introducen inadvertidamente sesgos en productos basados en **LLM**, como GPT-4, pueden dar la impresión de estar “seguros pero incorrectos” en algunos temas. Es un poco como escuchar hablar a alguien sobre algo que no sabe. Superar estas limitaciones es clave para construir la confianza pública en esta nueva tecnología. Aquí es donde entra en juego **RLHF**, del que se hablará mas adelante, para ayudar a controlar o dirigir sistemas de **Inteligencia Artificial** a gran escala.

Los principales problemas que deben abordarse son:

**Preocupaciones éticas y de privacidad:** Actualmente, no existen muchas leyes o salvaguardas que regulen el uso de **LLM** y, dado que los grandes conjuntos de datos contienen mucha información confidencial o sensible (robo de datos personales, infracciones de derechos de autor y propiedad intelectual, entre otros), esto plantea la cuestión de problemas éticos, de privacidad e incluso psicológicos entre los usuarios que buscan respuestas en foros generados por **Inteligencia Artificial**.

**Sesgo y prejuicio:** Dado que los **LLM** se entranan en diferentes fuentes, pueden devolver inconscientemente el sesgo presente en esas fuentes. Los sesgos, incluidos los culturales, raciales, de género, y otros, están presentes en los datos de entrenamiento. Estos prejuicios pueden tener consecuencias reales, como decisiones de contratación de personal, atención médica o resultados financieros.

**Costos ambientales y computacionales:** Entrenar un **Large Language Model** requiere una enorme cantidad de cálculo informático, lo que afecta el consumo de energía y las emisiones de carbono. Además, es costoso. Muchas empresas, especialmente las más pequeñas, simplemente no pueden permitírselo.

Para contrarrestar estos efectos potencialmente dañinos, los investigadores se centran en diseñar **LLM** en torno a tres pilares principales: utilidad, veracidad e inocuidad. Si un **LLM** puede mantener los tres principios, se considera “alineado” (*aligned*), un término que tiene elementos de subjetividad.

### ***Reinforcement Learning from Human Feedback***

***Reinforcement Learning from Human Feedback (RLHF)*** puede hacer frente a los retos mencionados anteriormente, lo que lleva a preguntarse: ¿puede una máquina aprender valores humanos?

En el fondo, **RLHF** implementa la opinión humana para generar un conjunto de datos de preferencias humanas que determina la función de recompensa para un resultado deseado. La opinión humana puede obtenerse de varias maneras [?]:

**Orden de preferencia:** Las personas clasifican los productos por orden de preferencia.

**Demostraciones:** Los seres humanos escriben las respuestas preferidas a las indicaciones.

**Correcciones:** Los humanos editan la salida de un modelo para corregir comportamientos desfavorables.

**Entradas en lenguaje natural:** Los humanos proporcionan descripciones o críticas de los resultados en lenguaje. Una vez creado un modelo de recompensa, se utiliza para entrenar un modelo de referencia con la ayuda del aprendizaje por refuerzo, que aprovecha el modelo de recompensa para construir una política de valores humanos que el ***Large Language Models*** utiliza a continuación para producir respuestas. ChatGPT es un buen ejemplo de cómo un gran modelo lingüístico utiliza **RLHF** para producir respuestas mejores, más seguras y más atractivas.

**RLHF** constituye un gran avance en el ámbito de los modelos lingüísticos, ya que proporciona una experiencia de usuario más controlada y fiable. Pero hay una contrapartida: **RLHF** introduce los sesgos de quienes han contribuido al conjunto de datos de preferencias utilizados para entrenar el modelo de recompensa. Así, aunque ChatGPT está orientado hacia respuestas útiles, honestas y seguras, sigue estando sujeto a las interpretaciones de los anotadores de estas respuestas. Aunque **RLHF** mejora la coherencia (lo que

es estupendo para el uso de **LLM** en los motores de búsqueda), lo hace a expensas de la creatividad y la diversidad de ideas.

### 3.2. *Retrieval-augmented Generation*

Los **LLM** han demostrado su capacidad para comprender el contexto y ofrecer respuestas precisas a diversas tareas de **Procesamiento del Lenguaje Natural**, como la síntesis o las preguntas y respuestas, cuando se les solicita. Aunque son capaces de ofrecer muy buenas respuestas a preguntas sobre información con la que fueron entrenados, tienden a “alucinar” cuando el tema trata sobre información que desconocen, es decir, que no estaba incluida en sus datos de entrenamiento.

*Retrieval-augmented Generation* es una técnica avanzada en el campo del **PLN** que combina dos enfoques clave: recuperación y generación de texto. Esta técnica se utiliza para mejorar la generación de texto automática y garantizar que las respuestas generadas sean precisas, relevantes y contextualmente adecuadas [?].

En un sistema de **RAG**, el proceso se divide en dos etapas, ver figura 3.12:

1. Recuperación (*Retrieval*): En esta etapa, el sistema busca información relevante en grandes conjuntos de datos o bases de conocimiento. Utiliza métodos de recuperación de información para encontrar documentos o fragmentos de texto que contienen información relacionada con la consulta o el contexto actual.
2. Generación (*Generation*): Una vez que se ha recuperado la información relevante, el sistema de generación de texto (a menudo basado en un **LLM**, como **GPT**) utiliza esta información para generar respuestas coherentes y contextualmente apropiadas.

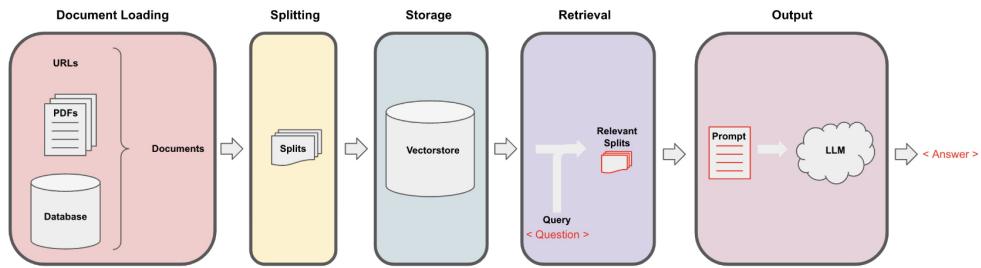


Figura 3.12: Secuencia para la creación de un *Retrieval-augmented Generation*.

La combinación de estas dos etapas, como se muestra en la figura 3.13 permite que la técnica **RAG** proporcione respuestas que no solo se basen en el conocimiento preexistente [?], sino que también sean sensibles al contexto específico de la consulta o la tarea. Esto propicia respuestas más precisas y relevantes en comparación con enfoques puramente generativos [?, ?].

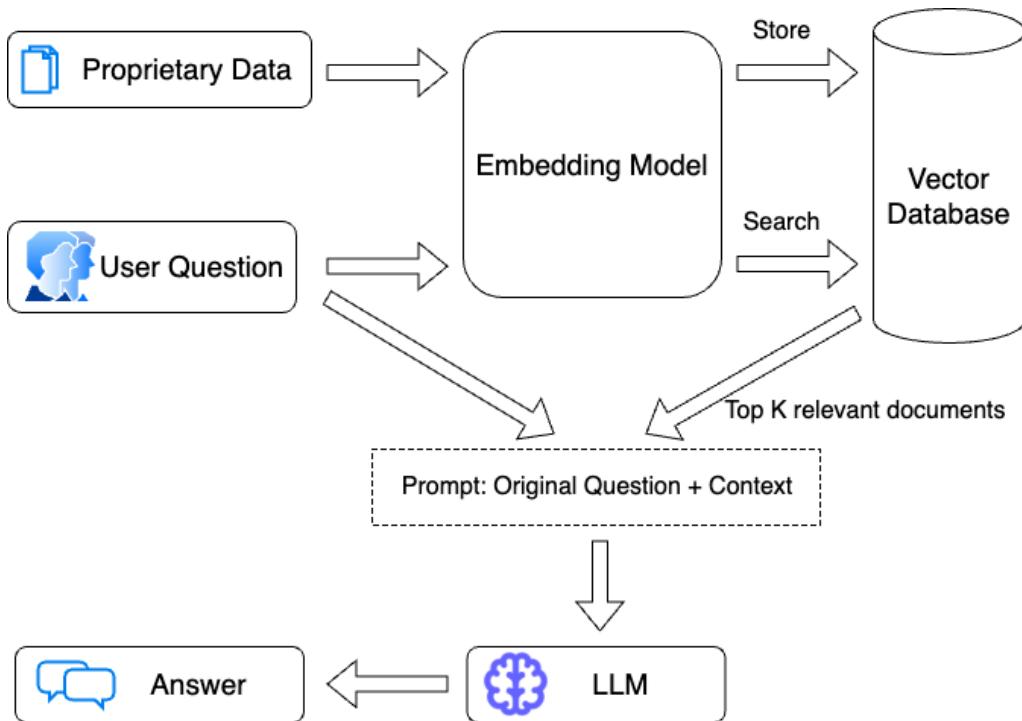


Figura 3.13: Esquema del funcionamiento de un *Retrieval-augmented Generation*.

**RAG** se utiliza en una variedad de aplicaciones, incluyendo chatbots, sistemas de respuesta automática, motores de búsqueda mejorados y generación de contenido automático, donde la capacidad de acceder y utilizar información específica es esencial para brindar respuestas más precisas.

## Preprocesamiento de datos

El preprocesamiento de datos es una parte fundamental del desarrollo de *Large Language Models* y *Retrieval-augmented Generation*. Aunque los detalles específicos del preprocesamiento pueden variar según la tarea y la arquitectura del modelo, algunos pasos comunes incluyen:

- **Tokenización:** Los modelos de lenguaje trabajan con unidades más pequeñas llamadas tokens. Tokenizar un texto implica dividirlo en estas unidades, que podrían ser palabras, subpalabras o incluso caracteres.
- **Normalización:** Esto implica convertir el texto a un formato estándar, como convertir todas las letras a minúsculas. Esto ayuda a que el modelo no trate las mismas palabras en diferentes formas como entidades separadas.
- **Eliminación de stopwords:** Para algunos modelos, puede ser beneficioso eliminar palabras comunes que no aportan mucha información (como "y", "o", "el", etc.) para reducir el ruido en los datos.
- **Lidiar con datos no estructurados:** Si los datos contienen elementos no textuales, como imágenes o tablas, se debe tener un proceso para manejarlos o convertirlos en un formato que el modelo pueda entender.
- **Segmentación de texto:** Para tareas específicas, como la recuperación de respuesta, puede ser útil dividir el texto en segmentos más pequeños para facilitar la búsqueda y la recuperación de información relevante.
- **Manejo de datos desbalanceados:** Si los datos están desbalanceados (por ejemplo, se tienen muchas más instancias de una clase que de otra), es posible que se desean aplicar técnicas para abordar este desequilibrio.

El preprocesamiento puede variar según la tarea y el modelo específico que se está utilizando. Algunos modelos, como *Generative Pre-trained Transformer (GPT)*, han demostrado ser bastante robustos y pueden manejar

datos en bruto con un preprocesamiento mínimo, mientras que otros modelos pueden requerir una preparación más cuidadosa de los datos.

Además, para **RAG**, también hay un enfoque importante en la creación de conjuntos de datos que vinculen preguntas con respuestas relevantes para el entrenamiento efectivo del modelo.

La información contenida en los datos del **FAQ** para el **TFG** presenta una estructura que no es óptima para el Chatbot basado en **LLM**. Se cuenta con diversas fuentes de datos que deben ser integradas en el **RAG**. Esta problemática se abordará con mas profundidad en la sección **5.3**.

## ***Embeddings***

En el **Procesamiento del Lenguaje Natural (PLN)**, un embedding de palabra es una representación de una palabra que se utiliza en el análisis de texto. Normalmente, la representación es un vector de valores reales que codifica el significado de la palabra de tal manera que se espera que las palabras que están más cercanas en el espacio vectorial sean similares en significado. Los embeddings de palabra se obtienen utilizando técnicas de modelado del lenguaje y aprendizaje de características, donde las palabras o frases del vocabulario se asignan a vectores de números reales.

Existen diversos métodos para generar este mapeo, que incluyen redes neuronales, reducción de dimensionalidad en la matriz de co-ocurrencia de palabras, modelos probabilísticos, métodos basados en bases de conocimiento explicables y representación explícita en términos del contexto en el que aparecen las palabras [?].

Estos embeddings de palabra y frase, cuando se utilizan como representación de entrada subyacente, han demostrado mejorar el rendimiento en tareas de **PLN** como el análisis sintáctico y el análisis de sentimientos. El desarrollo histórico de este enfoque se remonta a la década de 1950, con la idea de que “una palabra se caracteriza por la compañía que mantiene”. A lo largo del tiempo, se han desarrollado modelos de espacio semántico para representar el conocimiento basado en la distribución de propiedades en grandes conjuntos de datos lingüísticos.

La popularidad de los embeddings de palabra creció significativamente con los avances en modelos neuronales en la década de 2010, y herramientas como word2vec de Google de la que ya se ha hablado, desempeñaron un papel crucial al acelerar el entrenamiento de modelos de espacio vectorial. A medida que la tecnología y el hardware mejoraron, se realizaron avances

teóricos y prácticos, lo que llevó a la aplicación práctica de los embeddings de palabra en diversas áreas de la investigación y la aplicación práctica.

## Bases de datos Vectoriales

Una base de datos vectorial es un tipo de base de datos que almacena datos como vectores de alta dimensionalidad, que son representaciones matemáticas de características o atributos. Cada vector tiene un número específico de dimensiones, que pueden variar desde decenas hasta miles, dependiendo de la complejidad y la granularidad de los datos. Los vectores suelen generarse aplicando algún tipo de función de transformación o incrustación a los datos sin procesar, como texto, imágenes, audio, vídeo y otros. La función de incrustación puede basarse en diversos métodos, como modelos de aprendizaje automático, incrustaciones de palabras o algoritmos de extracción de características [?].

La principal ventaja de una base de datos vectorial es que permite realizar búsquedas y recuperación de datos rápida y precisa basada en la distancia o similitud de sus vectores. Esto significa que, en lugar de utilizar métodos tradicionales para consultar bases de datos basadas en coincidencias exactas o criterios predefinidos, se puede utilizar una base de datos vectorial para encontrar los datos más similares o relevantes según su significado semántico o contextual.

Por ejemplo, se puede utilizar una base de datos vectorial para:

- Encontrar imágenes similares a una imagen dada según su contenido visual y estilo.
- Encontrar documentos similares a un documento dado según su tema y sentimiento.
- Encontrar productos similares a un producto dado según sus características y calificaciones.

Para realizar búsquedas y recuperación de similitudes en una base de datos vectorial, se debe utilizar un vector de consulta que represente la información o criterios deseados. El vector de consulta puede derivarse del mismo tipo de datos que los vectores almacenados (por ejemplo, usar una imagen como consulta para una base de datos de imágenes) o de diferentes tipos de datos (por ejemplo, usar texto como consulta para una base de datos de imágenes). Luego, se debe utilizar una medida de similitud que calcule

qué tan cercanos o distantes están dos vectores en el espacio vectorial, ver figura 3.14. La medida de similitud puede basarse en diversas métricas, como similitud del coseno, distancia euclídea, distancia de Hamming, índice de Jaccard.

## Vector Store

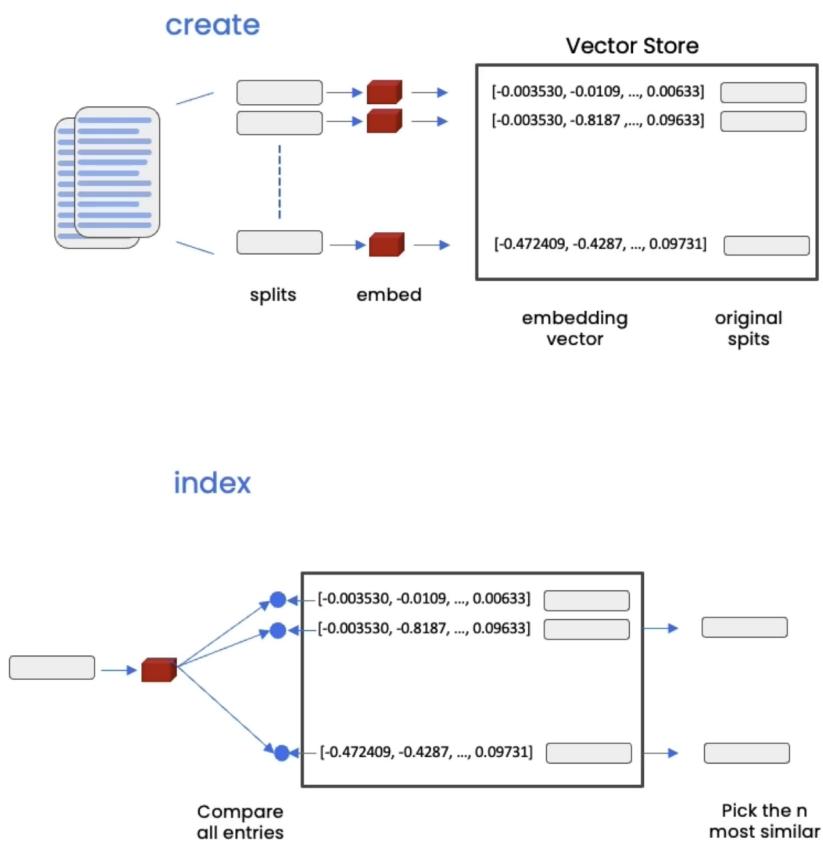


Figura 3.14: Esquema del funcionamiento de creación de bases de datos vectoriales con Embeddings.

El resultado de la búsqueda y recuperación de similitudes suele ser una lista clasificada de vectores que tienen las puntuaciones de similitud más altas con el vector de consulta. Luego se puede acceder a los datos originales, sin tratar, de cada vector.

Las bases de datos vectoriales tienen muchos casos de uso en diferentes dominios y aplicaciones que involucran **Procesamiento del Lenguaje Natural**, visión por computadora, sistemas de recomendación y otras áreas que requieren comprensión semántica y coincidencia de datos.

Un caso de uso para almacenar información en una base de datos vectorial es potenciar a los **Large Language Models** para generar texto más relevante y coherente basado en un complemento de **Inteligencia Artificial**. Sin embargo, los **LLM** a menudo enfrentan desafíos como generar información inexacta o irrelevante, carecer de consistencia o sentido común, repetirse o contradecirse, ser sesgados u ofensivos. Para superar estos desafíos, se puede utilizar una base de datos vectorial que almacene información sobre diferentes temas, palabras clave, hechos, opiniones y/o fuentes relacionadas con el dominio o género deseado. Luego, se puede usar un **Large Language Model** y pasar información desde la base de datos vectorial con el complemento de **Inteligencia Artificial** para generar contenido más informativo y atractivo que se ajuste a la intención y estilo.

### 3.3. *Prompt Engineering*

Para obtener información de un **LLM**, hay que hacer una pregunta. Si un **LLM** es como una base de datos de millones de programas vectoriales, entonces un *prompt* es como una consulta de búsqueda en esa base de datos. Una parte de la consulta puede interpretarse como una “clave de programa”, el índice del programa que se desea recuperar, y otra como una entrada de programa.

Esta “base de datos de programas” es continua e interpolativa, no es un conjunto discreto de programas. Esto significa que una instrucción ligeramente diferente, como “Reformula este texto en el estilo de x”, aún habría apuntado a una ubicación muy similar en el espacio de programas, dando como resultado un programa que se comportaría de manera bastante cercana pero no idéntica.

Hay miles de variaciones que se podrían haber utilizado, cada una dando como resultado un programa similar pero ligeramente diferente. Y es por eso que se necesita el *Prompt Engineering*. No hay una razón a priori por la cual una primera instrucción, ingenua, daría como resultado el programa óptimo para la tarea. El **LLM** no va a entender lo que se quiso decir y luego realizarlo de la mejor manera posible, simplemente recuperará el programa al que apunta la instrucción, entre muchas ubicaciones posibles en las que podrías haber aterrizado.

El *Prompt Engineering* es el proceso de buscar en el espacio de programas para encontrar el programa que empíricamente parece funcionar mejor para una tarea objetivo. No es diferente de probar diferentes palabras clave al hacer una búsqueda en Google de un software.

Si los **LLM** realmente entendieran lo que se les dice, no habría necesidad de este proceso de búsqueda, ya que la cantidad de información transmitida sobre la tarea objetivo no cambiaría, independientemente de que se use por ejemplo la instrucción con la palabra “reescribe” en lugar de “reformula”. Nunca se debe asumir que el **LLM** “lo entiende” desde el principio. Se debe de tener en cuenta que la instrucción es solo una dirección en un mundo casi infinito de programas, todo capturado como un subproducto de organizar tokens en un espacio vectorial a través de un objetivo de optimización autoregresivo.

### ***Chain-of-thought***

Existen multitud de técnicas de *Prompt Engineering* pero una de las mas conocidas es ***Chain-of-thought*** (**CoT**). La técnica de ***Chain-of-thought prompting*** es una estrategia que permite a los **LLM** abordar un problema como una serie de pasos intermedios antes de proporcionar una respuesta final. Mejora la capacidad de razonamiento al inducir al modelo a responder un problema de múltiples pasos con pasos de razonamiento que imitan una cadena de pensamiento. Esta técnica permite a los **LLM** superar dificultades en tareas de razonamiento que requieren pensamiento lógico y múltiples pasos para resolver, como preguntas de aritmética o razonamiento del sentido común.

Por ejemplo, ante la pregunta “Q: La cafetería tenía 23 manzanas. Si usaron 20 para hacer el almuerzo y compraron 6 más, ¿cuántas manzanas tienen?”, un *prompt CoT* podría inducir al **LLM** a responder “A: La cafetería tenía originalmente 23 manzanas. Usaron 20 para hacer el almuerzo. Así que tenían  $23 - 20 = 3$ . Compraron 6 manzanas más, así que tienen  $3 + 6 = 9$ . La respuesta es 9” [?].

Q: {question}

A: Let's think step by step.

Inicialmente, cada *prompt CoT* incluía algunos ejemplos de preguntas y respuestas (Q&A). Esto lo convertía en una técnica de *prompting* “few-shot”. Sin embargo, se ha demostrado que simplemente agregar las palabras

“Pensemos paso a paso” también es efectivo, lo que convierte a CoT en una técnica de *prompting* “zero-shot”. Esto permite una mejor escalabilidad, ya que el usuario ya no necesita formular muchos ejemplos de preguntas y respuestas específicos de CoT.

### **Few-shot-prompting vs Zero-shot-prompting**

El *few-shot-prompting* y *zero-shot-prompting* son enfoques específicos en el ámbito de la generación de lenguaje natural y el aprendizaje con modelos de lenguaje, especialmente en contextos como GPT u otros modelos similares. Aquí hay una breve descripción de ambos [?].

- **Few-shot-prompting:** El *few-shot-prompting* implica proporcionar al modelo solo un pequeño número de ejemplos de entrada, llamados “pocas muestras” (few-shot), junto con un *prompt* o indicación específica que guía al modelo sobre la tarea que debe realizar.

Este enfoque es útil cuando se quiere dirigir el modelo para realizar una tarea particular, pero no se dispone de grandes conjuntos de datos etiquetados. Las pocas muestras y el *prompt* juntos actúan como una guía para la generación de respuestas o contenido coherente.

- **Zero-shot-prompting:** El *zero-shot-prompting* va un paso más allá al permitir que el modelo realice tareas sin haber visto ejemplos específicos de esa tarea durante el entrenamiento. Se basa en la capacidad del modelo para generalizar y comprender tareas sin información previa.

Es beneficioso en escenarios donde se necesitan respuestas o generación de texto para tareas no vistas anteriormente. La capacidad de realizar tareas *zero-shot* amplía la aplicabilidad del modelo a una variedad más amplia de situaciones.

Ambos enfoques son parte de la búsqueda continua de mejorar la flexibilidad y la aplicabilidad de los LLM, permitiéndoles realizar tareas específicas con una entrada mínima o nula para esas tareas durante el entrenamiento.

## **3.4. TFG y Chatbot actual**

### **Trabajo Fin de Grado**

Un Trabajo de Fin de Grado (TFG) en el contexto de un grado de Ingeniería Informática en España se refiere a un proyecto académico que los

estudiantes deben completar al finalizar su carrera universitaria. El **TFG** es un componente obligatorio y esencial para obtener el título de Grado en Ingeniería Informática [?]. Un esquema del proceso seguido durante el **TFG** en la **UBU** se puede ver en la figura 3.15.

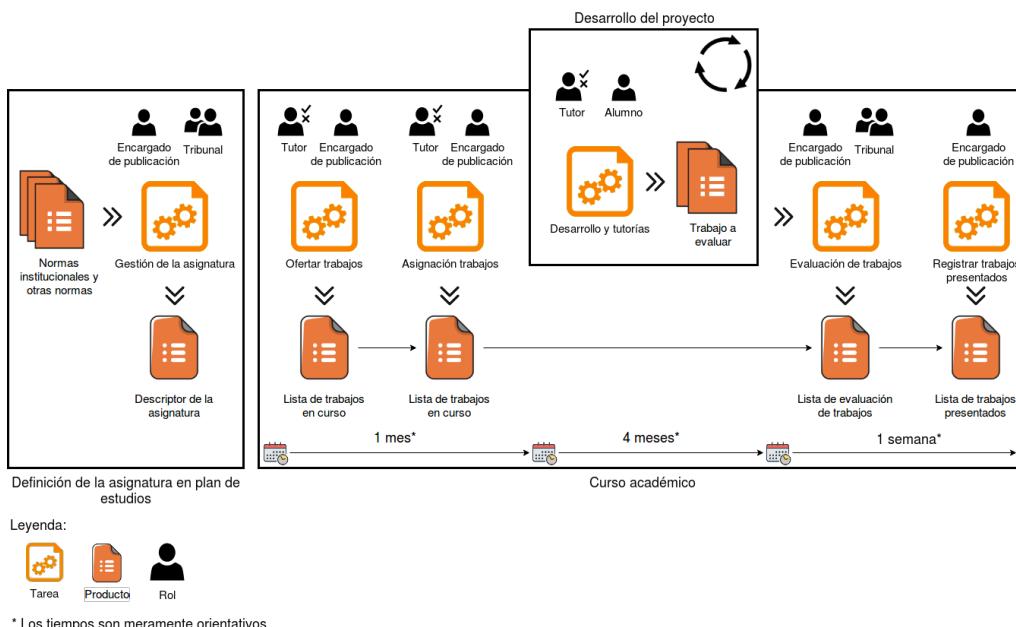


Figura 3.15: Proceso de gestión de TFG en la UBU.

El **Trabajo de Fin de Grado (TFG)** constituye un requisito para obtener el grado de Ingeniería Informática, y su finalización es necesaria para cumplir con los requisitos de graduación. En la **Universidad de Burgos (UBU)**, el **TFG** se aborda como un proyecto individual, proporcionando a los estudiantes la oportunidad de aplicar de forma práctica los conocimientos adquiridos durante su carrera.

Los temas del **TFG** son diversos y pueden abarcar una amplia gama de áreas dentro de la Ingeniería Informática, como desarrollo de software, diseño de sistemas, análisis de algoritmos, **Inteligencia Artificial**, ciberseguridad y redes, entre otros. Los estudiantes suelen contar con la orientación de uno o más tutores académicos, con experiencia en el área del proyecto.

En la documentación y presentación del **TFG**, donde los estudiantes deben elaborar informes detallados sobre el proceso de desarrollo, los resultados y las conclusiones. Además, se espera que realicen una presentación oral para defender y explicar su trabajo ante un tribunal académico.

La evaluación del **TFG** se realiza considerando diversos aspectos, como la calidad técnica del proyecto, la claridad de la documentación, la presentación oral y la capacidad del estudiante para aplicar los conocimientos adquiridos. En algunos casos, se espera que el **TFG** aporte alguna forma de contribución original al campo de la Ingeniería Informática, ya sea a través de la implementación de una solución innovadora o de la investigación en un área específica.

## Chatbots

Un chatbot es una aplicación de software diseñada para interactuar con usuarios a través de un chat, ya sea de texto o de voz, mediante respuestas automáticas. Estos fueron creados para simular el comportamiento de un agente humano, un desafío significativo en el campo de la **Inteligencia Artificial**, como demuestra el famoso test de Turing.

El test de Turing, propuesto por Alan Turing en 1950, establece que un sistema inteligente que imita a un humano debe engañar al menos al treinta por ciento de los jueces haciéndoles creer que es humano. Aunque Turing predijo que esto no ocurriría hasta el año 2000, el hito se alcanzó en 2014 cuando Eugene Goostman, un chatbot, superó la prueba al engañar al 33 % de los jueces.

Estos programas son ampliamente utilizados en diversos contextos, como sitios web, aplicaciones móviles, redes sociales y servicios de mensajería instantánea. Los chatbots pueden desempeñar roles variados, desde brindar información y asistencia al cliente hasta realizar tareas específicas como reservar vuelos, realizar compras en línea o proporcionar recomendaciones personalizadas.

En esencia, un chatbot actúa como un intermediario entre los usuarios y los sistemas subyacentes, facilitando la interacción y mejorando la accesibilidad a la información o servicios. Su popularidad ha crecido considerablemente debido a la conveniencia que ofrecen en la obtención rápida de respuestas y en la automatización de ciertas tareas.

En la práctica, los chatbots se diseñan principalmente con objetivos comerciales, como reducir costos laborales, aumentar la capacidad de trabajo atendiendo a miles de usuarios simultáneamente, y proporcionar respuestas más rápidas, especialmente en consultas a bases de datos. La **Inteligencia Artificial**, incluyendo los chatbots, está transformando el ámbito laboral, planteando desafíos y oportunidades. Según estudios como el del McKinsey

Global Institute, para 2030, la **Inteligencia Artificial** podría sustituir el 30 % de los empleos humanos, impactando significativamente la fuerza laboral.

## Herramientas y técnicas para la creación de Chatbots

Hay varias opciones y herramientas disponibles para la creación de chatbots, que van desde plataformas basadas en código hasta servicios de desarrollo de chatbots sin programación. Aquí hay algunas opciones populares:

### Dialogflow

1. Descripción: Dialogflow, de Google Cloud, es una plataforma de desarrollo de chatbots basada en la nube que utiliza técnicas de procesamiento del lenguaje natural (NLP).
2. Características:
  - a) Integra fácilmente con otros servicios de Google Cloud.
  - b) Permite la creación de chatbots para diversos canales, como web, aplicaciones móviles y asistentes de voz.
  - c) Ofrece una interfaz gráfica para diseñar flujos de conversación.

### Microsoft Bot Framework

1. Descripción: Desarrollado por Microsoft, Bot Framework proporciona herramientas y servicios para crear chatbots para aplicaciones, sitios web y servicios.
2. Características:
  - a) Admite múltiples lenguajes de programación, incluidos C# y Node.js.
  - b) Ofrece integración con servicios de inteligencia artificial de Microsoft, como Azure Cognitive Services.

### IBM Watson Assistant

1. Descripción: Watson Assistant, de IBM, utiliza **IA** para crear chatbots que pueden comprender y responder preguntas de manera natural.

2. Características:

- a) Utiliza aprendizaje automático para mejorar las respuestas con el tiempo.
- b) Permite la integración con otras soluciones de IBM Cloud.

**Botpress**

1. Descripción: Botpress es una plataforma de código abierto para la creación, gestión y despliegue de chatbots.

2. Características:

- a) Permite la personalización avanzada con acceso al código fuente.
- b) Admite la creación de chatbots en múltiples idiomas.

**Rasa**

1. Descripción: Rasa es una plataforma de código abierto para la creación de chatbots basados en **Inteligencia Artificial**.

2. Características:

- a) Ofrece capacidades avanzadas de procesamiento del lenguaje natural.
- b) Permite la creación de chatbots contextualmente inteligentes.

**Chatbot.com**

1. Descripción: Chatbot.com es una plataforma de desarrollo de chatbots sin código que permite a los usuarios crear chatbots de manera intuitiva.

2. Características:

- a) Interfaz de arrastrar y soltar para el diseño del flujo de conversación.
- b) Integración con plataformas de mensajería populares.

## Wit.ai

1. Descripción: Wit.ai, propiedad de Facebook, es una plataforma de procesamiento del lenguaje natural que permite crear chatbots con capacidades de comprensión de lenguaje.
2. Características:
  - a) Proporciona modelos preentrenados para simplificar el desarrollo.
  - b) Ofrece integración con aplicaciones, sitios web y dispositivos.

## LLMs (Large Language Models)

1. Descripción: Los *Large Language Models (LLM)* son modelos de **Inteligencia Artificial** que se entranan en grandes cantidades de datos de texto para comprender y generar lenguaje humano de manera avanzada. Estos modelos utilizan arquitecturas de redes neuronales profundas para aprender patrones complejos y representaciones semánticas del lenguaje.
2. Características:
  - a) Capacidad de Comprensión: Los **LLM** tienen una notable capacidad para comprender contextos complejos y generar respuestas coherentes.
  - b) Aprendizaje Continuo: Pueden ser entrenados en conjuntos de datos actualizados para mantenerse al día con la evolución del lenguaje y la información.
  - c) Adaptabilidad: Son adaptables a una variedad de tareas, desde la generación de texto creativo hasta la respuesta a preguntas específicas.
  - d) Complejidad del Lenguaje: Pueden manejar la complejidad del lenguaje natural, incluyendo matices, ambigüedades y diferentes estilos de expresión.
  - e) Generación de Contenido: Son capaces de generar contenido nuevo y coherente en función del contexto proporcionado.

La combinación de **LLM** y **RAG** ofrece un enfoque poderoso para la creación de chatbots, ya que permite aprovechar la capacidad de generación avanzada de lenguaje de los **LLM** mientras mejora la precisión y relevancia mediante la recuperación de conocimiento contextual. Este

Plataforma	Castellano	UI Based	App Integration	Gratis
Dialogflow	X	X	X	X
Microsoft Bot Framework	X	X	X	
IBM Watson	X	X	X	X
Botpress	X		X	
Rasa	X			X
Chatbot.com	X	X	X	
Wit.ai	X	X	X	
LLM+RAG	X			X

Tabla 3.1: Plataformas de desarrollo de Chatbots.

enfoque es especialmente útil en situaciones donde se necesita acceder a información específica para proporcionar respuestas más precisas.

Estas opciones varían en complejidad, flexibilidad y requisitos de programación. La elección de la plataforma dependerá de factores como los objetivos del chatbot, las habilidades de programación disponibles y la preferencia en términos de servicios en la nube.

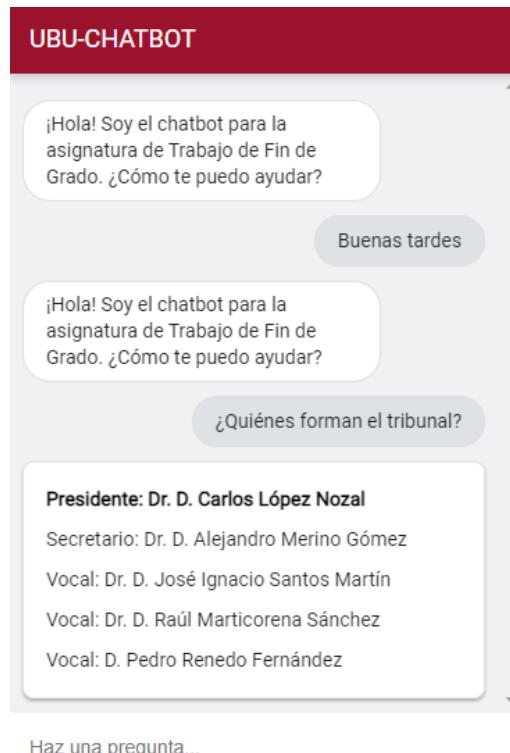
## Tabla comparativa

En la tabla 3.1 se muestra la comparativa entre las distintas plataformas y herramientas para la creación de chatbots.

## Chatbot actual

El chatbot para la resolución de dudas que actualmente está disponible en la plataforma UBUVirtual es el resultado del TFG realizado por Alfredo Asensio Vázquez en 2021 y titulado “Desarrollo y explotación de un chatbot de FAQs sobre una asignatura de Trabajo Fin de Grado” [?], ver figura 3.16.

El chatbot actual logra resolver eficazmente más del 50 % de las preguntas formuladas por los alumnos, lo que resultó en una notable reducción de consultas al personal docente de la asignatura a través de otros medios. Es destacable que este logro se alcanzó sin incurrir en ningún coste adicional.



Haz una pregunta...

Figura 3.16: UI del UBU-CHATBOT actual.

La herramienta elegida para la realización del chatbot fue Dialogflow. Dialogflow ha demostrado ser una solución óptima para el proyecto, proporcionando un **Procesamiento del Lenguaje Natural (PLN)** potente, gratuito y con una curva de aprendizaje mínima. La efectividad de las conversaciones depende en gran medida del correcto planteamiento de preguntas por parte de los estudiantes. Aquellos que formulen preguntas precisas y concisas obtendrán mejores resultados, y gracias a que los usuarios son estudiantes de Ingeniería Informática con conocimientos sobre las limitaciones de la **Inteligencia Artificial**, este problema no se consideró crítico.

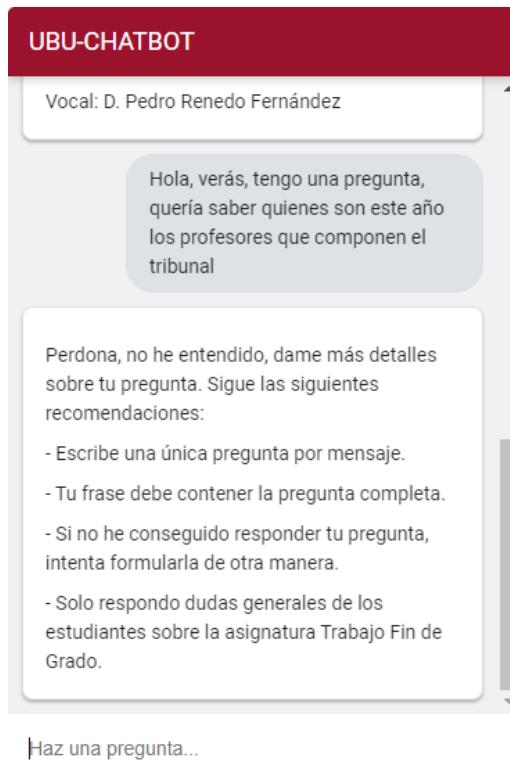


Figura 3.17: Limitaciones del UBU-CHATBOT actual relativas a la formulación de preguntas por parte de los estudiantes.

La adición de indicaciones sobre cómo formular preguntas en caso de mensajes no reconocidos mejoró la experiencia del usuario, pero sigue siendo una limitación para el uso del chatbot, ver figura 3.17. Los chatbots son una herramienta en crecimiento, impulsados por la potencia creciente de los *Large Language Models*. En los últimos meses se ha mejorado enormemente las posibilidades y las herramientas disponibles.

Otra carencia del chatbot actual es que en su base de datos de preguntas y respuestas no utiliza el histórico de datos de los **TFG** de otros cursos. Solo se disponen de un reducido grupo de ejemplos que están embebidos en las preguntas y respuestas generales.

---

## 4. Técnicas y herramientas

---

En esta sección de la memoria, se presentan las técnicas y las herramientas de desarrollo que han sido empleadas en la ejecución de este proyecto. Debido a que el campo de los LLM se encuentra actualmente en una fase de desarrollo temprana y en constante cambio, se han barajado distintas alternativas para la realización de este proyecto. El carácter investigador y exploratorio del tfg hace que se hayan tenido que desechar caminos por encontrar limitaciones o mejores soluciones disponibles.

Durante la etapa de prototipado y la fase inicial de investigación, se emplearon herramientas distintas a las utilizadas en la versión definitiva del proyecto. Las herramientas utilizadas más destacadas se describen en los siguientes apartados.

### 4.1. Específicas de los LLM

A pesar de que la inteligencia artificial generativa es un campo relativamente reciente, la explosión de los LLM ha dado lugar a uno numero considerable de herramientas y técnicas en constante cambio y desarrollo.

Para la interacción del LLM se han barajado en distintos momentos alternativas y ampliaciones a LangChain como el uso LlamaIndex o GPT4all. Se ha optada finalmente por combinar Langchain con Hugging Face, ambas herramientas se explican más adelante en detalle, aunque también se explican algunas herramientas para el uso de LLM instalados localmente.

## LangChain

LangChain es un *framework* de código abierto que permite a los desarrolladores de software que trabajan con **Inteligencia Artificial (IA)** y su subconjunto de aprendizaje automático, combinar **Large Language Models** con otros componentes externos para desarrollar aplicaciones impulsadas por modelos **LLM**. El objetivo de LangChain es vincular modelos de **LLM**, como GPT-3.5 y GPT-4 de OpenAI, con una variedad de fuentes de datos externas para crear y aprovechar los beneficios de las aplicaciones de **Procesamiento del Lenguaje Natural (PLN)**.

Desarrolladores, ingenieros de software y científicos de datos con experiencia en los lenguajes de programación Python, JavaScript o TypeScript pueden utilizar los paquetes de LangChain ofrecidos en esos idiomas. LangChain se lanzó como un proyecto de código abierto por los cofundadores Harrison Chase y Ankush Gola en 2022; la versión inicial se lanzó ese mismo año [?].

### ¿Por qué es importante LangChain?

LangChain es un *framework* que simplifica el proceso de creación de interfaces de aplicaciones de inteligencia artificial generativa. Los desarrolladores que trabajan en este tipo de interfaces utilizan diversas herramientas para crear aplicaciones avanzadas de **PLN**; LangChain agiliza este proceso. Por ejemplo, los **LLM** deben acceder a grandes volúmenes de big data, por lo que LangChain organiza estas grandes cantidades de datos para que se puedan acceder fácilmente.

Además, los modelos **Generative Pre-trained Transformer (GPT)** generalmente se entrena en datos hasta su liberación al público. Por ejemplo, ChatGPT se lanzó al público a finales de 2022, pero su base de conocimientos se limitaba a datos de 2021 y anteriores. LangChain puede conectar modelos de **IA** a fuentes de datos para darles conocimiento de datos recientes sin limitaciones.

### ¿Cuáles son las características de LangChain?

LangChain se compone de los siguientes módulos, ver figura 4.1, que aseguran que los múltiples componentes necesarios para crear una aplicación efectiva de **PLN** puedan funcionar sin problemas:

1. **Interacción del modelo:** También llamado entrada/salida del modelo, este módulo permite que LangChain interactúe con cualquier

modelo de lenguaje y realice tareas como gestionar las entradas al modelo y extraer información de sus salidas. Conexión y recuperación de datos. Los datos a los que acceden los **LLM** pueden transformarse, almacenarse en bases de datos y recuperarse de esas bases de datos mediante consultas con este módulo.

2. **Cadenas:** Al construir aplicaciones más complejas con LangChain, se pueden requerir otros componentes o incluso más de un **LLM**. Este módulo enlaza múltiples **LLM** con otros componentes o **LLM**, conocido como una cadena de **LLM**.
3. **Agentes:** El módulo de agentes permite que los **LLM** decidan los mejores pasos o acciones a tomar para resolver problemas. Lo hace orquestando una serie de comandos complejos a los **LLM** y otras herramientas para que respondan a solicitudes específicas.
4. **Memoria:** El módulo de memoria ayuda a un **LLM** a recordar el contexto de sus interacciones con los usuarios. Se puede agregar memoria a corto y largo plazo a un modelo, según el uso específico.

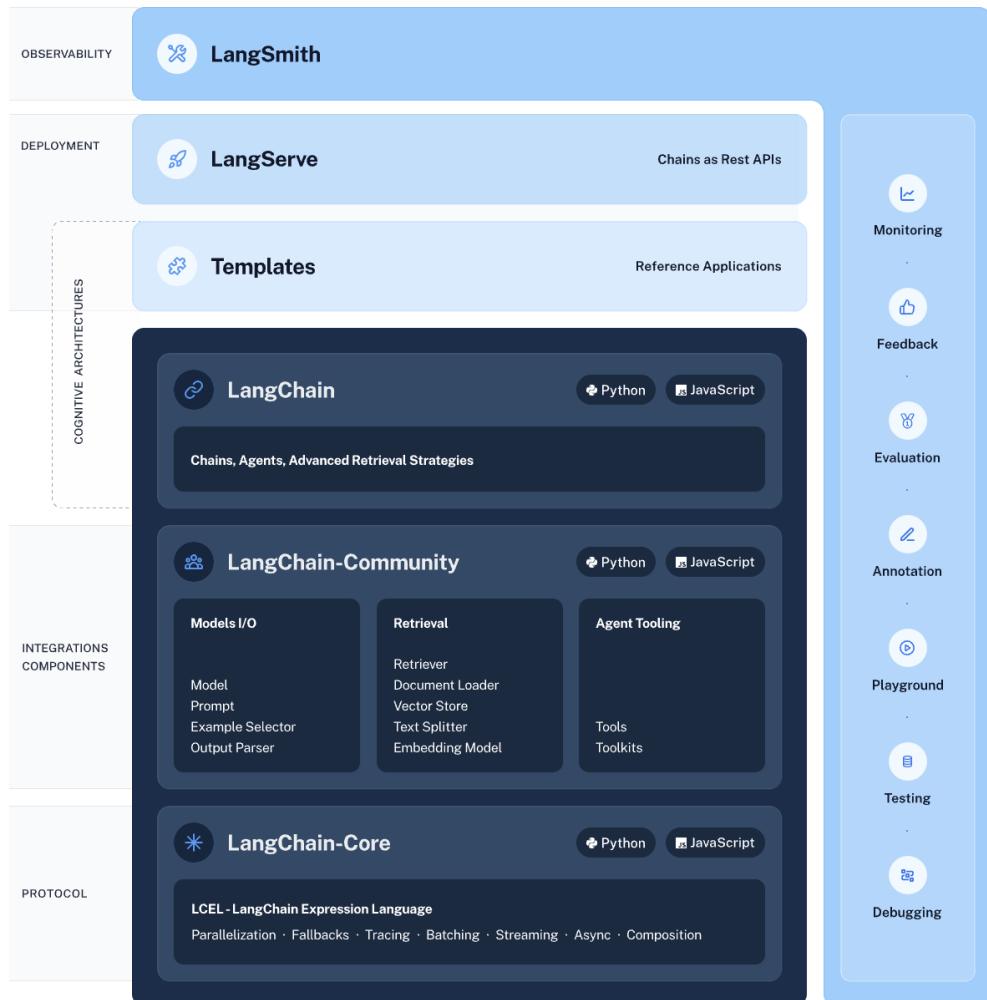


Figura 4.1: Esquema del *Framework* de Langchain con los distintos componentes y módulos.

### ¿Cuáles son las integraciones de LangChain?

LangChain generalmente construye aplicaciones utilizando integraciones con proveedores de **LLM** y fuentes externas donde se pueden encontrar y almacenar datos. Por ejemplo, LangChain puede construir chatbots o sistemas de preguntas y respuestas integrando un **LLM**, como los de Hugging Face, Cohere y OpenAI, con fuentes o almacenes de datos como Apify Actors, Google Search y Wikipedia. Esto permite que una aplicación tome texto de entrada del usuario, lo procese y recupere las mejores respuestas de

cualquiera de estas fuentes. En este sentido, las integraciones de LangChain utilizan la tecnología de **Procesamiento del Lenguaje Natural** más actualizada para construir aplicaciones efectivas.

Otras integraciones potenciales incluyen plataformas de almacenamiento en la nube, como Amazon Web Services, Google Cloud y Microsoft Azure, así como bases de datos de vectores. Una base de datos de vectores puede almacenar grandes volúmenes de datos de alta dimensión, como vídeos, imágenes y texto extenso, como representaciones matemáticas que facilitan la consulta y búsqueda de esos elementos de datos. Pinecone es un ejemplo de base de datos de vectores que se puede integrar con LangChain.

### ¿Cómo crear prompts en LangChain?

Los prompts sirven como entrada al **LLM** que le indica que devuelva una respuesta, que suele ser una respuesta a una consulta. Esta respuesta también se denomina salida. Un prompt debe diseñarse y ejecutarse correctamente para aumentar la probabilidad de obtener una respuesta bien escrita y precisa de un modelo de lenguaje. Es por eso que la ingeniería de prompts es una ciencia emergente que ha recibido más atención en los últimos años.

Los prompts pueden generarse fácilmente en implementaciones de LangChain utilizando una plantilla de prompt, que se utilizará como instrucciones para el **LLM** subyacente. Las plantillas de prompts pueden variar en especificidad. Pueden diseñarse para plantear preguntas simples a un modelo de lenguaje. También se pueden utilizar para proporcionar un conjunto de instrucciones explícitas a un modelo de lenguaje con suficiente detalle y ejemplos para recuperar una respuesta de alta calidad.

LangChain generalmente requiere al menos una integración. OpenAI es un ejemplo destacado. Para usar las interfaces de programación de aplicaciones **LLM** de OpenAI, un desarrollador debe crear una cuenta en el sitio web de OpenAI y recuperar la clave de acceso a la **API**. Luego, utilizando el siguiente fragmento de código, instale el paquete Python de OpenAI e ingrese la clave para acceder a las **API**.

### ¿Cómo desarrollar aplicaciones en LangChain?

LangChain está diseñado para desarrollar aplicaciones con funcionalidad de modelos de lenguaje. Hay diferentes formas de hacer esto, pero el proceso generalmente implica algunos pasos clave.

El desarrollador debe definir primero un caso de uso específico para la aplicación. Esto también implica determinar su alcance, incluidos los requisitos como cualquier integración, componente y **LLM** necesario.

LangChain permite a los desarrolladores modificar su código para crear funcionalidades personalizadas que satisfagan las necesidades del caso de uso y den forma al comportamiento de la aplicación. Aunque es cierto que solo se puede modificar hasta un cierto punto. Es importante elegir el **LLM** adecuado para el trabajo y también ajustarlo finamente para cumplir con las necesidades del caso de uso.

### ***Document Loaders***

Los *document loaders* en el contexto de *Large Language Models* como LLaMa.cpp o LangChain se refieren a componentes o módulos diseñados para cargar documentos o datos en la memoria del modelo. Estos documentos actúan como contexto o información de referencia que el modelo puede utilizar durante el proceso de inferencia para comprender y responder de manera más precisa a las consultas o preguntas que se le presentan. Básicamente es la función que se describe en el método **RAG**.

En términos generales, la carga de documentos es esencial para proporcionar contexto y conocimiento al modelo, mejorando así su capacidad para generar respuestas significativas. El proceso de carga de documentos implica tomar información desde diversas fuentes, como bases de datos, páginas web, archivos de texto, etc., y convertirla en un formato que el modelo pueda entender y utilizar. En la figura 4.2 se pueden ver algunos de los *document loaders* que existen en LangChain.

## Document Loaders

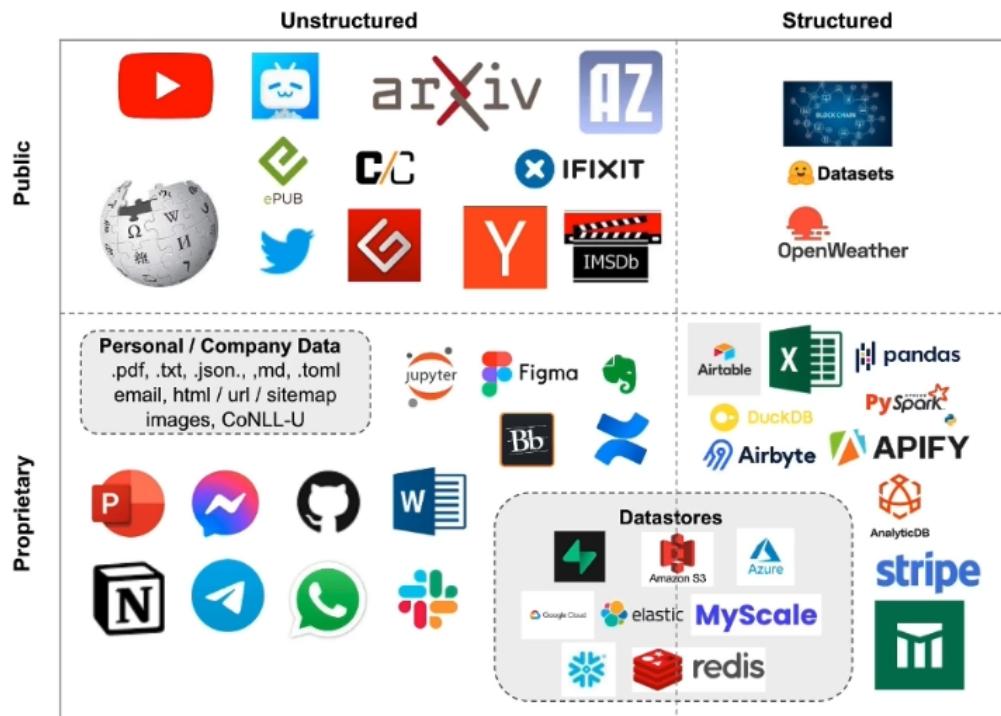


Figura 4.2: Matriz de representación de los múltiples *Document Loaders* disponibles en LangChain.

En resumen, los *document loaders* son parte fundamental del proceso de preparación de datos para *Large Language Models*, asegurando que tengan acceso a la información relevante que les permita realizar tareas específicas de manera efectiva.

### ¿Para qué se ha usado LangChain en este TFG?

Langchain ha sido la herramienta mas utilizada en este TFG. Se ha usado tanto en la fase de prototipado con modelos locales en Kaggle, como en la la versión de producción final que utiliza una API.

No solo se ha usado en las distintas versiones del chatbot, sino que se ha usado en todos los módulos y de forma general ha gestionado todos o casi todos los proceso del chatbot. Se usa desde la creación de la configuración

del **LLM** hasta la gestión de las llamadas a la **API**. También se utiliza en la creación de la base de datos vectorial y la recuperación de la información con búsquedas de similitud.

## Hugging Face

Hugging Face es una empresa y plataforma que se especializa en modelos de lenguaje natural y **Deep Neural Networks**. Ofrecen una amplia gama de recursos y herramientas destinados a facilitar el desarrollo, entrenamiento y despliegue de modelos de **Procesamiento del Lenguaje Natural (PLN)**.

Algunos aspectos destacados de Hugging Face incluyen [?]:

- **Modelos preentrenados:** Hugging Face proporciona acceso a una variedad de modelos de lenguaje natural preentrenados de última generación. Esto incluye modelos como BERT, **GPT**, Mistral, LLa-Ma, RoBERTa, y muchos otros, que han demostrado un rendimiento excepcional en diversas tareas de procesamiento del lenguaje natural.
- **Transformers Library:** La Transformers Library de Hugging Face es una biblioteca de código abierto que facilita el uso, entrenamiento y ajuste fino de modelos de transformer para tareas específicas. Proporciona una interfaz consistente para varios modelos y se utiliza ampliamente en la comunidad de aprendizaje profundo para **PLN**.
- **Hugging Face Hub:** Es una plataforma en línea que permite a los desarrolladores compartir, explorar y utilizar modelos de lenguaje natural de Hugging Face. Facilita la colaboración y el intercambio de modelos entrenados por la comunidad.
- **Pipeline API:** Hugging Face ofrece una **API** de canalización (Pipeline API) que simplifica el uso de modelos complejos para tareas específicas. Esto hace que sea fácil utilizar modelos de **PLN** preentrenados para clasificación de texto, traducción, resumen y más.
- **Comunidad de usuarios:** La plataforma fomenta la colaboración y la contribución de la comunidad al código y los modelos. Los usuarios pueden contribuir con modelos, compartir implementaciones y participar en discusiones relacionadas con el procesamiento del lenguaje natural y la **Inteligencia Artificial**.

Hugging Face se ha convertido en un recurso integral para la comunidad de aprendizaje profundo y **PLN**, proporcionando modelos avanzados,

bibliotecas de código abierto y una plataforma para compartir y colaborar en proyectos relacionados con el Procesamiento del Lenguaje Natural.

En este TFG se ha usado HuggingFace como fuente de modelos locales, que se han descargado en el ordenador para hacer pruebas en la fase de prototipado. Como también de proveedor de la API del modelo Mistral, que se usa en la solución final como alternativa *Open Source* de OpenAI.

### ***Quantization y modelos locales***

El tener que depender de API gratuitas de *Open Source* ha sido una constante limitación en el proyecto. Si bien estos modelos y herramientas ofrecen bastantes posibilidades, también tienen grandes limitaciones sobre todo en comparación a la API de OpenAI.

Por este motivo se ha investigado y probado la instalación local de LLM de Open Source como LLaMa2 ha través de GPT4all y llama-cpp-python. Se han usado durante la fase de prototipado tanto Mistral, como otros LLM. La versión final con Mistral no se hace con un modelo local sino a través de una API. Los modelos en local, ya sea en Kaggle o en el ordenador han resultado suficientemente estables, por lo que se han descartado para el modelo de explotación del chatbot, pero si han resultado cruciales en la fase de prototipado e investigación.

### **Quantization**

GGML es una biblioteca Tensor para *machine learning* que se presenta como una biblioteca en C++. Su función principal es permitir la ejecución de modelos de *Large Language Models* en la CPU o en combinación con la GPU. Un aspecto distintivo de GGML es su definición de un formato binario para la distribución de LLM. Además, GGML emplea una técnica llamada *Quantization* que posibilita la ejecución de modelos de LLM en hardware de consumo [?].

Los pesos de los LLM son números de punto flotante (decimales). Al igual que se necesita más espacio para representar un número entero grande (por ejemplo, 1000) en comparación con un entero pequeño (por ejemplo, 1), se requiere más espacio para representar un número de punto flotante de alta precisión (por ejemplo, 0.0001) en comparación con un número de punto flotante de baja precisión (por ejemplo, 0.1). El proceso de *Quantization* de un modelo de lenguaje grande implica reducir la precisión con la que se representan los pesos para disminuir los recursos necesarios para utilizar el modelo. GGML admite varias estrategias de *Quantization* (por ejemplo,

*Quantization* de 4 bits, 5 bits y 8 bits), cada una de las cuales ofrece diferentes compromisos entre eficiencia y rendimiento.

Para utilizar eficazmente los modelos, es esencial tener en cuenta los requisitos de memoria y disco. Dado que los modelos se cargan completamente en la memoria, se necesita suficiente espacio en disco para almacenarlos y RAM suficiente para cargarlos durante la ejecución. En el caso del modelo de 65 mil millones de parámetros, incluso después de la *Quantization*, se recomienda tener al menos 40 gigabytes de RAM disponible. Cabe destacar que los requisitos de memoria y disco son actualmente equivalentes.



Model	Original size	Quantized size (4-bit)
7B	13 GB	3.9 GB
13B	24 GB	7.8 GB
30B	60 GB	19.5 GB
65B	120 GB	38.5 GB

Figura 4.3: Efecto de la *Quantization* en el tamaño de los LLM de LLaMa.

La *Quantization* desempeña un papel crucial en el manejo de estas demandas de recursos. A menos que se disponga de recursos computacionales excepcionales, la *Quantization* permite utilizar los modelos en configuraciones de hardware más modestas al reducir la precisión de los parámetros del modelo y optimizar el uso de la memoria. Esto garantiza que la ejecución de los modelos siga siendo factible y eficiente para una gama más amplia de configuraciones. Se puede ver en la figura 4.3, como se reduce el tamaño original de los modelos al ser utilizada esta técnica.

La *Quantization* se ha usado en este TFG para instalar modelos en Kaggle y para instalar modelo en locales en el ordenador a través de Llama-cpp-python. Finalmente se han descartado los modelos locales por que requieren muchos recursos de los que no se dispone. Como ejemplo se han realizado

pruebas en un MacBook Air de 2021 y en la generación de una respuesta se ha tardado mas de 20 minutos. Para poder usar estos modelos locales es conveniente usar recursos Cloud como AWS o Azure que pueden proveer de suficiente capacidad de computo para que estos **LLM** se ejecuten de forma fluida.

### Llama-cpp-python

LLama.cpp fue desarrollado por Georgi Gerganov. Implementa la arquitectura LLaMa de Meta de manera eficiente en C/C++ y es una de las comunidades de código abierto más dinámicas en torno a la inferencia de *Large Language Models* con más de 390 contribuyentes, 43,000+ estrellas en el repositorio oficial de GitHub y 930+ versiones [?].

El diseño de Llama.cpp como una biblioteca C++ centrada en la **CPU** significa menos complejidad y una integración perfecta en otros entornos de programación. Esta amplia compatibilidad aceleró su adopción en diversas plataformas. Actuando como un repositorio para características críticas de bajo nivel, Llama.cpp refleja el enfoque de LangChain para capacidades de alto nivel, simplificando el proceso de desarrollo aunque con posibles desafíos de escalabilidad futura.

Se centra en una única arquitectura de modelo, permitiendo mejoras precisas y efectivas. Su compromiso con los modelos Llama a través de formatos como GGML y GGUF ha llevado a ganancias de eficiencia sustanciales.

El núcleo de LLama.cpp son los modelos Llama originales, que también se basan en la arquitectura de *transformers*. Los autores de Llama aprovechan varias mejoras que posteriormente se propusieron y utilizan diferentes modelos como PaLM.

Esta librería se ha usado para la instalación e modelos locales en la fase de prototipado. No se encuentra en la versión final del chatbot, ya que la velocidad de ejecución en un ordenador comercial es lenta y es necesaria una compleja instalación.

### GPT4All

GPT4All es un ecosistema diseñado para entrenar e implementar *Large Language Models*. Notablemente, estos modelos están destinados a ejecutarse localmente en **CPU** de consumo, lo que los hace accesibles y eficientes para una amplia gama de usuarios. El objetivo principal de GPT4All es servir como un modelo de lenguaje ajustado finamente al estilo de un asistente,

ofreciendo un alto nivel de personalización. Se alienta a los usuarios, ya sean individuos o empresas, a utilizar, distribuir y construir libremente sobre los modelos de GPT4All [?].

Características Clave:

- **Implementación Local:** Los modelos de GPT4All están optimizados para ejecutarse en **CPU** de consumo, lo que permite la implementación local sin la necesidad de recursos computacionales extensivos.
- **Personalización:** El ecosistema enfatiza la personalización, permitiendo a los usuarios adaptar los modelos de lenguaje según sus necesidades y preferencias específicas.
- **Ecosistema de Código Abierto:** GPT4All está respaldado por un software de ecosistema de código abierto. Este enfoque abierto fomenta la colaboración, la transparencia y las contribuciones de la comunidad.
- **Tamaño del Archivo:** Los modelos de GPT4All se distribuyen como archivos descargables que van desde 3 GB hasta 8 GB, lo que facilita su descarga e integración en los sistemas de los usuarios.
- **Nomic AI:** El ecosistema de software es respaldado y mantenido por Nomic AI, que desempeña un papel crucial en garantizar la calidad y seguridad de los modelos. Nomic AI también lidera los esfuerzos para simplificar el proceso de entrenamiento e implementación para usuarios que desean crear sus propios modelos de lenguaje amplio.

Como alternativa a HuggingFace para obtener modelos para uso en ordenadores locales se ha probado con GPT4All. Esta opción tiene mucho potencial, pero para ello sería necesario mas potencia de procesamiento o nuevos modelos mas adaptados a limitados recursos.

## FAISS

FAISS, que significa *Facebook AI Similarity Search* (Búsqueda de Similitud de Inteligencia Artificial de Facebook) [?], es una biblioteca desarrollada por Facebook(ahora META) para realizar búsquedas eficientes de similitud en conjuntos grandes de datos. Se utiliza comúnmente para realizar búsquedas de vecinos más cercanos en conjuntos de datos de vectores de alta dimensionalidad, como los que se encuentran en tareas de aprendizaje automático y **Procesamiento del Lenguaje Natural**.

Está optimizado para manejar grandes cantidades de datos y realizar búsquedas de vecinos más cercanos de manera eficiente. Puede trabajar con vectores de diferentes dimensiones y tipos de datos, haciendo que sea versátil para aplicaciones en una variedad de dominios.

También aprovecha técnicas de implementación eficientes y puede aprovechar la capacidad de procesamiento paralelo de hardware, como **GPU**, para acelerar las operaciones de búsqueda de similitud. Ofrece métodos eficientes para la indexación de vectores, lo que facilita la búsqueda rápida en grandes conjuntos de datos.

Se utiliza a menudo en conjunto con bibliotecas de aprendizaje profundo, como PyTorch, y puede integrarse fácilmente en flujos de trabajo de aprendizaje automático. También se encuentra disponible en bibliotecas de **LLM** como por ejemplo Langchain.

Implementa algoritmos especializados para la búsqueda eficiente de vecinos más cercanos en espacios de alta dimensión, lo que la hace destacar en la creación de bases de datos vectoriales, especialmente si se usan las técnicas **RAG**.

En resumen, FAISS es una herramienta fundamental para tareas que involucran la búsqueda eficiente de similitud en grandes conjuntos de datos, lo que lo convierte en una opción popular en el campo de la recuperación de información, bases de datos vectoriales y la minería de datos. En este **TFG** ha sido la herramienta elegida para generar la base de datos vectorial con los datos de los **FAQ**. También es necesario para el proceso **RAG** que extrae la información relevante de esa base de datos vectorial en base a una búsqueda de similitud con la pregunta realizada.

## 4.2. Prototipado, documentación y gestión

### Kaggle

Kaggle es una plataforma en línea que aloja competiciones de ciencia de datos. Fundada en 2010, Kaggle proporciona un entorno donde científicos de datos y profesionales del aprendizaje automático pueden encontrar conjuntos de datos, participar en competiciones, colaborar en proyectos y mejorar sus habilidades en análisis de datos y modelado predictivo.

Kaggle, una plataforma integral para *data science*, ofrece competiciones regulares donde los participantes resuelven problemas variados, desde reconocimiento de imágenes hasta predicción de precios. Además, proporciona

conjuntos de datos públicos para prácticas y proyectos en diversas áreas. Los Kernels son entornos en línea para escribir, ejecutar y compartir código, ideales para la exploración de datos y la creación de modelos. La comunidad activa de Kaggle, respaldada por foros, facilita la discusión y el intercambio de conocimientos. La plataforma también ofrece recursos educativos, tutoriales y cursos para mejorar las habilidades en ciencia de datos y aprendizaje automático.

Kaggle ha crecido hasta convertirse en una de las plataformas más importantes para la comunidad de ciencia de datos y el uso de modelos, proporcionando oportunidades para la colaboración, la competencia y el aprendizaje continuo.

Para el prototipado se ha hecho uso de esta herramienta de forma intensiva. Esta elección se ha debido a que los modelos de Mistral y LLaMa 2 de Meta estaban disponibles en la plataforma y existe una amplia comunidad de usuarios. Se han usado modelos locales en estos *notebooks* y se ha usado el enlace con el repositorio del proyecto en GitHub para mantener un control de las distintas versiones que se creaban en las primeras fases de este **TFG**.

## GitHub

GitHub ha sido la herramienta de repositorio de versiones para el proyecto. Aunque se han valorado otras opciones como Gitlab, se ha optado por GitHub al ser una herramienta conocida de amplio uso en el grado de ingeniería informática en la **UBU**.

GitHub es una plataforma de desarrollo de software basada en web que utiliza el sistema de control de versiones Git. Lanzada en 2008, se ha convertido en una de las plataformas más populares para el alojamiento de proyectos de desarrollo colaborativo. Algunos aspectos clave de GitHub incluyen:

Permite a los desarrolladores alojar sus proyectos y controlar las versiones de su código utilizando Git. Los repositorios pueden ser públicos (accesibles para todos) o privados (restringidos a un conjunto de colaboradores). Facilita la colaboración entre desarrolladores. Varios colaboradores pueden trabajar en un proyecto, realizar cambios y fusionar sus contribuciones de manera eficiente.

Utiliza Git para el control de versiones, lo que permite realizar un seguimiento de los cambios en el código a lo largo del tiempo. Los desarrolladores pueden revertir a versiones anteriores, ramificar su código para trabajar en nuevas características y fusionar cambios de diferentes ramas.

GitHub proporciona herramientas para realizar un seguimiento de problemas (*bugs*, mejoras, tareas, etc.) y para proponer cambios en el código mediante solicitudes de extracción. Puede integrarse con servicios de despliegue continuo, lo que facilita la implementación automática de cambios en un entorno de producción después de pasar las pruebas necesarias.

El enlace del repositorio de GitHub es <https://github.com/jrg1013/chatbot>. En él se pueden encontrar las *Issues* del Proyecto, los *Milestones* y sus correspondientes *Branches*. Se han realizado *commits* con mejoras incrementales y desde Kaggle. También se han realizado versiones de la aplicación en distintos momentos del proyecto y se ha configurado los archivos *Readme* y *License* para facilitar la reutilización.

## Zube.io

Zube es una potente herramienta de gestión de proyectos y colaboración que ha facilitado la gestión ágil del tfg basada en sprints. Al principio se probaron otras posibles alternativas más integradas en GitHub, como Zenhub o Projects, pero al final se optó por Zube.io debido a los siguientes motivos:

1. **Tablero Kanban vs. Tablero Sprint:** Zube admite Kanban y Sprint, lo que permite a los usuarios elegir uno de ellos para la gestión de proyectos, y ambos métodos pueden utilizarse al mismo tiempo. La principal diferencia entre Kanban y Sprint es que Sprint tiene una línea de tiempo predefinida, generalmente de 2 semanas. Ver figura 4.4 para ver un ejemplo de uno de los Sprints del proyecto. Durante este periodo, se lleva a cabo una discusión diaria para alinear el estado del proyecto y abordar cualquier caso urgente. Cada 2 semanas se completan algunas características para que se prueben o revisen antes de pasar al siguiente sprint. Kanban no tiene el concepto de línea de tiempo, pero es útil para proyectos a largo plazo.

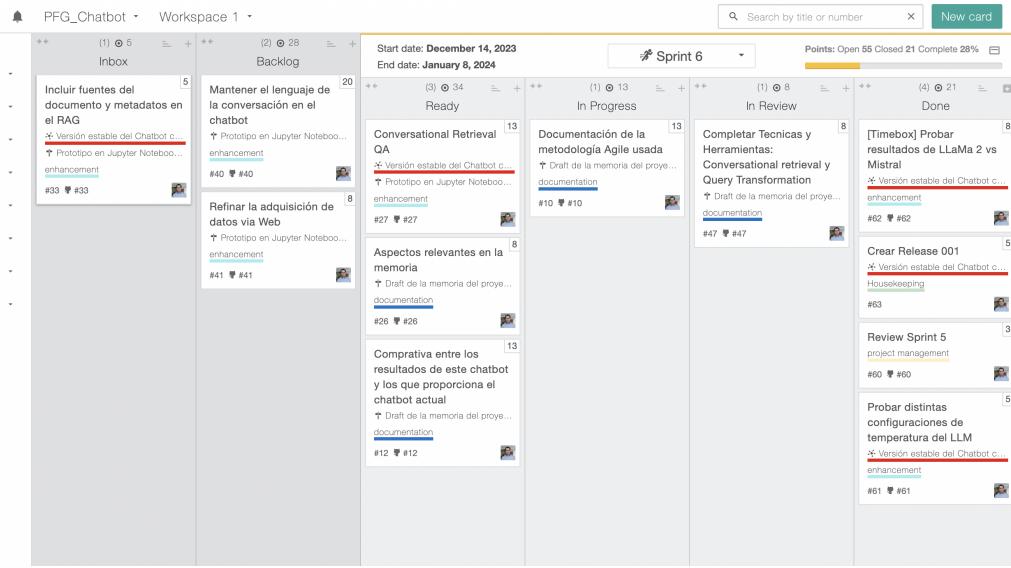


Figura 4.4: *Sprint Board* de Zube.io con los distintos estados de las *Issues*.

2. **Conexión con GitHub:** Una de las mejores características de Zube es que puede conectarse con GitHub, una popular plataforma de gestión de código. Esta conexión permite que los desarrolladores gestionen su código en GitHub, y los Project Managers (PM) pueden verificar el estado en Zube, facilitando una estrecha colaboración sin interferir en el trabajo del otro.
3. **Etiquetas:** Se pueden crear y editar etiquetas tanto en Zube como en GitHub. Aunque es una función simple, las etiquetas son útiles para verificar el estado de las funciones y las tarjetas. Las etiquetas con colores facilitan su distinción en el montón de tarjetas, brindando una ayuda visual.
4. ***Issue Manager:*** Es una función que ahorra tiempo al mover varias tarjetas a la vez, especialmente al finalizar un sprint. Permite filtrar según cada componente establecido en las tarjetas, actualizar sus parámetros o moverlas directamente a secciones o sprints específicos.

En resumen, Zube simplifica la gestión de proyectos Agile o Scrum al ofrecer opciones flexibles de Kanban, conexión con GitHub, etiquetas visuales y un eficiente *Issue Manager*.

## Overleaf

Para la realización de la documentación de la memoria, se ha optado por usar Overleaf y la plantilla LaTeX del **TFG**.

Overleaf es una plataforma en línea diseñada para simplificar la creación, edición y colaboración en documentos científicos y técnicos, especialmente aquellos redactados en LaTeX. Su entorno de escritura colaborativa permite a varios usuarios editar simultáneamente un mismo documento, facilitando la colaboración en proyectos de investigación, artículos científicos, tesis, entre otros. Al utilizar una interfaz basada en web, Overleaf elimina la necesidad de instalar software adicional, permitiendo la edición directa de documentos LaTeX sin requerir una instalación local.

La plataforma ofrece diversas plantillas predefinidas para diferentes tipos de documentos académicos y científicos, junto con herramientas integradas para el formato automático según las convenciones de estilo de LaTeX. Además, proporciona un entorno de compilación en línea que genera el documento final en formato PDF, sin requerir la instalación de compiladores LaTeX en las máquinas locales de los usuarios.

Overleaf facilita la organización de documentos en proyectos, lo que simplifica la gestión de múltiples archivos y carpetas relacionadas. Además, incorpora un sistema de control de versiones que permite realizar un seguimiento de los cambios realizados en el documento a lo largo del tiempo. En resumen, Overleaf se presenta como una herramienta integral para la redacción colaborativa y la creación eficiente de documentos científicos y técnicos.

Para la realización de textos largos, como el del **TFG**, el tiempo de compilación de la versión gratuita no es suficiente. Por ello se ha recurrido a la colaboración con las cuentas de pago, que tiene más funciones, aparte del tiempo de compilación extendido.

## ChatGPT

Como se ha explicado en detalle en la sección 3.1, ChatGPT de OpenAI opera mediante la generación de texto basada en patrones y contextos proporcionados. Esta herramienta, lanzada hace alrededor de un año, ha generado un impacto significativo al transformar la forma en que las personas trabajan y llevan a cabo sus actividades diarias, contribuyendo al rápido avance de la inteligencia artificial generativa.

En el contexto de este proyecto, ChatGPT ha demostrado ser una herramienta versátil y eficaz, desempeñando un papel fundamental en diversas tareas. Se ha utilizado para reformular párrafos de manera más concisa, realizar correcciones ortográficas y gramaticales, resumir secciones extensas y organizar ideas de manera más estructurada.

Es importante destacar cómo la incorporación de ChatGPT ha optimizado significativamente el proceso de desarrollo y redacción, ofreciendo soluciones rápidas y precisas. Su capacidad para comprender contextos complejos y generar texto coherente ha mejorado la eficiencia y calidad del trabajo realizado en este proyecto. Este uso ejemplifica cómo las tecnologías de **Inteligencia Artificial** están transformando y mejorando los flujos de trabajo en diversos campos. De forma análoga se espera que el trabajo realizado en este **TFG** basado en **RAG** pueda mejorar la eficiencia de otros estudiantes.

## Visual Studio Code

Visual Studio Code (VSCode) es un **Entorno de desarrollo integrado (IDE)** que ha ganado popularidad debido a su ligereza, rapidez y versatilidad. Aquí hay algunas características clave de Visual Studio Code:

- **Gratis y de código abierto:** VSCode es gratuito y de código abierto, lo que significa que puedes personalizarlo según tus necesidades y contribuir a su desarrollo.
- **Multiplataforma:** Está disponible para Windows, macOS y Linux, lo que facilita su uso en diferentes sistemas operativos.
- **Lenguajes de programación:** Admite una amplia variedad de lenguajes de programación, proporcionando resultado de sintaxis, completado de código, y otras características específicas del lenguaje. En nuestro caso Python ha sido el lenguaje elegido y está perfectamente integrado en el **IDE**.
- **Extensiones:** Una de las características más potentes de VSCode es su ecosistema de extensiones. Puedes instalar extensiones para agregar funcionalidades específicas, como soporte para nuevos lenguajes, herramientas de depuración, integración con sistemas de control de versiones, entre otras. Se han usado extensiones para GitHub, Python y *coding rules*.

- **Herramientas de depuración integradas:** Ofrece herramientas de depuración integradas para varios lenguajes, lo que facilita el proceso de encontrar y corregir errores en tu código.
- **Control de versiones:** Viene con soporte integrado para sistemas de control de versiones como Git, lo que facilita el seguimiento de cambios en tu código. Se ha usado para gestionar las *Branches* y los *commits* con GitHub directamente.
- **Integración con terminal:** Tiene una terminal integrada que te permite ejecutar comandos directamente desde el entorno de desarrollo. Ya que en este proyecto es necesario usar comandos desde la terminal para el entrenamiento, validación y ejecución, esta función es extremadamente útil.
- **Snippets:** Admite *snippets* de código, que son fragmentos reutilizables de código que puedes insertar con facilidad.
- **Personalización:** Puedes personalizar la apariencia y el comportamiento de VSCode según tus preferencias.
- **Integración con servicios en la nube:** Se puede integrar servicios en la nube directamente desde VSCode, lo que facilita el desarrollo en entornos basados en la nube. En este caso se ha usado principalmente los enlaces con GitHub y Kaggle.

Visual Studio Code es una opción popular y versátil para desarrolladores debido a su flexibilidad, extensibilidad y las numerosas características que ofrece para facilitar el desarrollo de software.

## 4.3. Frontend

Para la creación de la **UI** se ha optado por separar en la medida de lo posible el *backend* y *frontend*. Para ello se ha investigado el uso de FastAPI y de una biblioteca de Python llamada Streamlit.

### FastAPI

FastAPI es un *framework* para el desarrollo rápido de **API** con Python 3.7 (o superior). Diseñado para ser fácil de usar, rápido y eficiente, FastAPI destaca por su sintaxis declarativa, generación automática de documentación interactiva (compatible con Swagger y ReDoc), y la capacidad de aprovechar

al máximo las características de la programación asíncrona. Es una elección muy popular para el desarrollo de RESTful APIs y aplicaciones web.

Una de las principales ventajas de las REST API radica en que el protocolo REST separa el almacenamiento de datos (backend) y la interfaz de usuario (frontend) del servidor, lo que permite que el cliente y el servidor sean independientes entre sí. Esta separación es fundamental para la arquitectura REST y aporta beneficios significativos al desarrollo de aplicaciones.

La independencia entre el backend y el frontend facilita la escalabilidad y la flexibilidad del sistema. Al dividir las responsabilidades de manera clara, los equipos de desarrollo pueden trabajar de manera más eficiente y concurrente en las distintas capas de la aplicación. Por ejemplo, el equipo encargado del desarrollo del backend puede realizar mejoras o modificaciones en la lógica de negocio y en la gestión de datos sin afectar directamente a la interfaz de usuario.

Esta separación también favorece la reutilización de componentes y la portabilidad del software. Dado que el backend y el frontend operan de manera independiente, es posible implementar cambios o actualizaciones en una capa sin afectar a la otra, siempre y cuando se respeten los contratos definidos por la REST API. Esto simplifica el mantenimiento y permite la integración de nuevas funcionalidades sin perturbar el funcionamiento existente.

FastAPI es una opción robusta y moderna para el desarrollo de API en Python, destacándose por su enfoque rápido, sintaxis clara y generación automática de documentación. Gran parte de esta funcionalidad descrita anteriormente, se encuentra ya implementado en la librería de Streamlit que ha sido la opción elegida finalmente. FAST API hacía una separación mas marcada del frontend y el backend pero la simplicidad para la creación de la UI ha sido el factor determinante para la elección de Streamlit.

## Streamlit

Streamlit es una biblioteca de código abierto en Python que se utiliza para crear aplicaciones web interactivas para *data science* y *machine learning* de manera rápida y sencilla. Su objetivo principal es permitir a los usuarios transformar datos en aplicaciones web interactivas con tan solo unas pocas líneas de código.

Con Streamlit, los especialistas de datos y desarrolladores pueden crear fácilmente interfaces de usuario atractivas para sus modelos, visualizaciones

y análisis de datos sin necesidad de conocimientos extensos en desarrollo web. La biblioteca se integra bien con bibliotecas populares de Python como Pandas, Matplotlib y Plotly, lo que facilita la creación de aplicaciones web a partir de código existente [?].

Streamlit simplifica el proceso de desarrollo de aplicaciones web al manejar automáticamente la actualización de la interfaz de usuario en respuesta a los cambios en los datos subyacentes. Esto permite a los usuarios centrarse más en el análisis de datos y la creación de visualizaciones, sin tener que preocuparse demasiado por los detalles de implementación de la interfaz web.

En resumen, Streamlit es una herramienta valiosa para aquellos que desean hacer una *interface* para análisis de datos y modelos de aprendizaje automático de manera efectiva a través de aplicaciones web interactivas con una curva de aprendizaje mínima.



---

## 5. Aspectos relevantes del desarrollo del proyecto

---

En este apartado se muestran los aspectos mas relevantes de este **TFG**. Se hace especial hincapié en los decisiones y apartados que mayor influencia han tenido en el proyecto. Por el carácter de investigación del trabajo y por lo temprano del desarrollo de esta tecnología, parte de este apartado está dedicado a la problemática que surge al emplear técnicas y herramientas que en constante cambio.

### 5.1. Selección de un LLM

Los *Large Language Models (LLM)* evolucionan con rapidez y continuamente aparecen nuevos modelos. Esto supone un reto: ¿Cómo seleccionar el **LLM** más adecuado para este proyecto? En este apartado se analizan las consideraciones prácticas que han guiado el proceso de toma de decisiones.

#### Licencias y uso comercial

Una consideración crucial a la hora de elegir un **LLM** es la concesión de licencias. Posiblemente el mas conocido y evolucionado es el **GPT** de OpenAI, pero es un modelo que puede tener un coste considerable. En cada *query* se ha de pagar una pequeña cantidad por token. Para el objeto de este proyecto es más adecuado el uso de un **LLM** de *open-source* o por lo menos con una licencia comunitaria para investigación y educación.

Aunque muchos modelos abiertos tienen restricciones de uso comercial, existen modelos disponibles para aplicaciones comerciales. Por ejemplo,

la familia de modelos MPT de MosaicML se publica bajo licencias que permiten su uso comercial. Se puede obtener más información sobre las distintas licencias en la *Open Source Initiative* y a través de la plataforma HugginFace.

## **Factores prácticos para la velocidad de inferencia y la precisión**

Los factores prácticos desempeñan un papel crucial a la hora de determinar la idoneidad de un **LLM** para el proyecto. Evaluar la velocidad de inferencia (el tiempo que tarda un **LLM** en procesar y generar resultados) es esencial, sobre todo cuando se trata de grandes cantidades de datos no estructurados. Una inferencia lenta puede dificultar la extracción de información de nuestro chatbot. Optar por modelos optimizados para una inferencia más rápida o capaces de manejar volúmenes de entrada sustanciales puede resultar ventajoso pero también será un modelo más pesado. Además, si se requiere una gran precisión en el análisis de sentimientos, la selección de un **LLM** con precisión y análisis de grano fino resulta crucial, y la velocidad de inferencia pasa a ser una consideración secundaria.

## **El impacto de la longitud del contexto y el tamaño del modelo**

Tener en cuenta la longitud del contexto y el tamaño del modelo es crucial a la hora de evaluar los **LLM**. Mientras que muchos **LLM** tienen limitaciones en la longitud de entrada, los modelos abiertos más recientes como Salesforce X-Gen admiten longitudes de contexto más largas, lo que permite entradas más completas y resultados deseados. El tamaño del modelo también influye en los requisitos de infraestructura, ya que los modelos más pequeños (menos de siete mil millones de parámetros) son más fáciles de implementar en hardware básico, lo que agiliza la implementación práctica. Por el contrario, algunos **LLM** ofrecen flexibilidad en el procesamiento de entradas más cortas, pero compensan con parámetros más grandes, atendiendo a casos de uso en los que la precisión dentro de un contexto restringido es primordial. Los **LLM** con contextos más largos y modelos de mayor tamaño tienden a ser más potentes, pero también tienen mayores exigencias computacionales.

## Específicos para una tarea o de uso general

Cuando se trata de *Large Language Models (LLM)*, a menudo está la disyuntiva de elegir entre **LLM** específicos para una tarea o **LLM** multitarea de propósito general que utilizan *prompts*. Mientras que estos últimos ofrecen versatilidad, los **LLM** para tareas específicas suelen ser más prácticos y eficientes para caso concretos de uso. Estos modelos especializados se entrenan y ajustan específicamente para una tarea concreta, lo que mejora el rendimiento y la precisión.

Otro aspecto relacionado es el idioma de entremetimiento del **LLM**. Los modelos más extendidos como LLaMa, OpenAI O Mistral, son modelos entrenados principalmente con grandes volúmenes de datos en inglés. Algunos modelos están específicamente entrenados y diseñados para responder en otros idiomas, como Águila [?] para español, o multilenguaje como PolyLM [?].

## Pruebas y evaluación

Las pruebas y evaluaciones exhaustivas son cruciales para determinar la fiabilidad de los **LLM**. Un método eficaz consiste en crear un conjunto de pruebas con ejemplos etiquetados manualmente. Una anotación fiable garantiza mediciones precisas. La comparación de los resultados de los modelos **LLM** con las referencias etiquetadas ayuda a calcular las métricas de precisión.

Se hablará mas acerca de este aspecto en una sección posterior, ya que la solución de validación del chatbot dependerá mas del framework de acceso, LangChain, que del **LLM** elegido.

## La revolución de los modelos abiertos

Recientemente, la adopción de modelos abiertos ha ido en aumento debido a factores como la preocupación por la privacidad de los datos y la rentabilidad. Los modelos abiertos, formados a partir de datos públicos, resuelven los problemas de privacidad asociados a los modelos cerrados. Además, a menudo ofrecen opciones más asequibles. Puede explorar recursos de **LLM** abiertos en Huggingface y consultar la tabla de clasificación de **LLM** de *LlamaIndex* [?].

## Consideraciones sobre los costes de implementación

Al seleccionar un **LLM**, el coste de implementación es también importante, no solo el coste por uso. El tamaño del modelo, los requisitos informáticos y la configuración de la infraestructura influyen en el coste total. Para la escalabilidad, se pueden considerar técnicas de optimización de modelos como la cuantificación(*quantification*), la aceleración de hardware o los servicios cloud para reducir costes. Lograr un equilibrio entre el rendimiento y la asequibilidad de la plataforma es esencial.

Tras realizar algunas pruebas con modelos locales y cuantificación, se optó por usar mejor la **API** de HuggingFace. Aunque se tengan algunos retrasos en el establecimiento de la conexión y en momentos puntuales el tiempo de respuesta sea más lento de lo deseado, se considera que es la mejor opción. La **API** permite usar el Chatbot por terceros de forma más sencilla, sin tener que instalar modelos locales y depender de las características de la **CPU** del equipo local.

## La necesidad de adaptarse al rápido ritmo del cambio

Casi cada semana se producen cambios en los **LLM**. Es un momento apasionante pero igualmente supone un reto para desarrollar versiones estables del chatbot. Se seleccionará un **LLM** que se adapte al caso de uso, y se usarán interfaces de acceso como LangChain que permitan reemplazar el **LLM** en caso de ser necesario, sin tener que rehacer el chatbot completamente.

Se hablará mas acerca de este aspecto y cómo ha influido, no solo en la elección del **LLM** sino también en el resto de aspectos del proyecto.

## Comparativa de LLMs

Se han valorado distintos aspectos, y se han probado en mayor o menor medida varios **LLM**. Los aspectos que se han valorado en mayor profundidad se han recogido en la tabla 5.1 [?, ?, ?].

## Modelo elegido: Mistral

Como ya se ha mencionado anteriormente, Mistral es una Startup Francesa que en solo unos meses ha conseguido una gran repercusión en el mundo de los **LLM** y del **PLN**. En comparación con otros modelos como LLaMa 2, Mistral 7B ofrece capacidades similares o mejores pero con menos carga computacional. Mientras que modelos fundacionales como GPT-4 pueden

Modelo	Multilenguaje	API	Open Source	Privacidad
GPT-4	X	X		
LLaMA 2	/	/	X	
Bard	/	X		
Claude	/			/
Mistral	/	/	X	X
OpenOrca		/	X	/

Tabla 5.1: Comparativa entre distintos *Large Language Models*

tener un mejor desempeño, Mistral ofrece una **API** gratuita a través de HuggingFace.

El modelo tiene 7B de parámetros y aunque modelos de más parámetros podrían dar mejores resultados, para la investigación que se desarrolla en este **TFG**, resulta más interesante poder realizar las pruebas sin incurrir en costes y tecnología propietaria como la de OpenAI.

Como alternativa a Mistral se ha desarrollado también en la fase de prototipado una segunda versión del Chatbot usando LLaMa 2 de Meta. En general ambos modelos en sus versiones de 7B de parámetros dan resultados similares, pero LLaMa 2 requiere solicitar acceso a la **API** para usos en investigación y no se encuentra en la Unión Europea, por lo que no tiene las mismas medidas para la protección de datos.

## 5.2. Fase temprana en la inteligencia artificial generativa

El inicio de la inteligencia artificial generativa marcó un hito significativo en el campo de la tecnología. Se refiere a la capacidad de las máquinas para generar contenido, especialmente en forma de texto, imágenes y otros tipos de datos. Un momento crucial en este avance fue la introducción de modelos de lenguaje generativos, como *Generative Pre-trained Transformer (GPT)*, que son capaces de entender, interpretar y generar texto de manera coherente y contextual.

Este desarrollo ha llevado a avances notables en diversas aplicaciones, como chatbots más inteligentes, asistentes virtuales más avanzados y generación automática de contenido creativo. Desde que OpenAI lanzara ChatGPT

el 30 de noviembre de 2022, los avances y mejoras en este campo han sufrido un avance vertiginoso, ver figura 5.1.

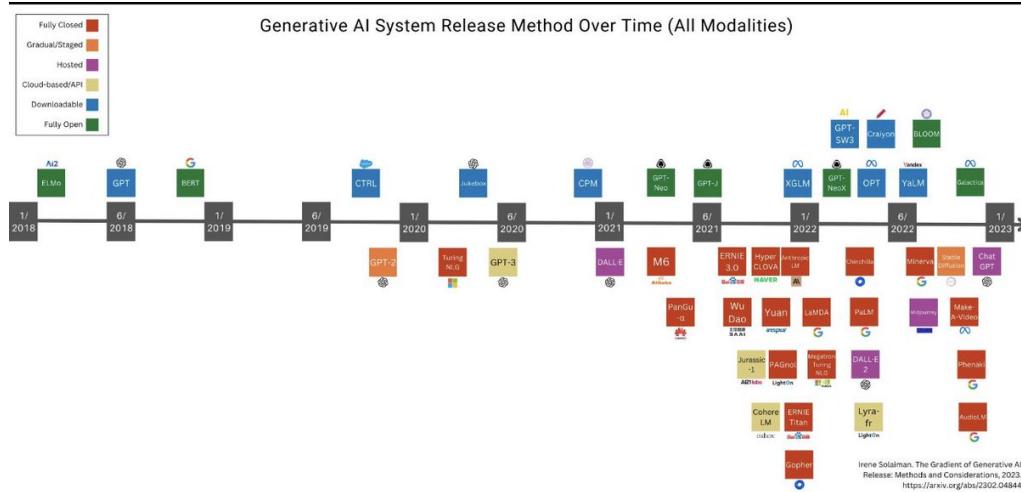


Figura 5.1: Avances en el campo de la Inteligencia Artificial Generativa en los últimos meses.

Uno de los factores clave en este rápido desarrollo es el enfoque de preentrenamiento de estos modelos, lo que significa que son entrenados en grandes cantidades de datos antes de ser afinados para tareas específicas. Esto les permite capturar patrones complejos y contextos, lo que resulta en un mejor rendimiento.

Sin embargo, este avance también ha planteado desafíos éticos y preocupaciones sobre el uso responsable de la inteligencia artificial generativa. La capacidad de crear contenido realista y convincente ha llevado a debates sobre la desinformación, la manipulación de información y la necesidad de salvaguardias para garantizar un uso ético y beneficioso de estas tecnologías.

En resumen, el inicio de la inteligencia artificial generativa es una fase emocionante con avances notables casi a diario, pero también plantea importantes consideraciones éticas y aspectos que se deben consolidar para garantizar un desarrollo tecnológico viable.

## Fase de investigación y desarrollo vs fase de explotación

La IA ha experimentado avances significativos en investigación y desarrollo, con la creación de LLM, redes neuronales profundas y enfoques

innovadores para tareas específicas. Sin embargo, la aplicación masiva y generalizada de la IA a productos en fase de explotación sigue siendo un desafío en muchos casos.

Algunas razones para esta brecha entre la investigación y la implementación amplia incluyen:

- **Múltiples vías de desarrollo:** A pesar de su corta historia, existen multitud de herramientas, técnicas y estrategias relacionadas con los LLM. En esta fase temprana no se ha consolidado una dirección como paradigma ha seguir por lo que existen muchos caminos de investigación que meses después se abandonan para proseguir por otros más prometedores.
- **Complejidad de Tareas del Mundo Real:** Muchas tareas del mundo real son complejas y requieren un entendimiento profundo del contexto. Aunque los modelos de IA han progresado, aún pueden enfrentar dificultades para manejar situaciones impredecibles o interpretar información de manera sutil.
- **Interpretabilidad y Transparencia:** Los modelos de IA, especialmente los de aprendizaje profundo, a menudo son cajas negras difíciles de interpretar. En entornos críticos, como la atención médica o la toma de decisiones legales, la interpretabilidad es esencial, y la falta de comprensión completa puede limitar la adopción.
- **Ética y Sesgo:** Las preocupaciones éticas relacionadas con el sesgo en los datos y la toma de decisiones algorítmica han generado discusiones importantes. La necesidad de abordar el sesgo y garantizar decisiones justas y equitativas sigue siendo un desafío.
- **Regulaciones y Normativas:** La implementación de la IA a menudo se ve afectada por regulaciones y normativas en evolución. Como la aprobada recientemente en la Unión Europea. Las preocupaciones sobre la privacidad, la seguridad y el impacto social han llevado a la introducción de leyes y estándares que afectan la aplicación generalizada.

A pesar de estos desafíos, es importante destacar que la IA se está utilizando en diversos sectores, desde asistentes virtuales hasta diagnóstico médico asistido por máquina. A medida que la investigación continúa y se abordan los desafíos actuales, es probable que veamos una mayor aplicación de la IA en productos y servicios en el futuro.

## LangChain como *framework* de gestión en un momento de cambio constante

Por lo anteriormente expuesto, se ha optado por usar LangChain como capa de intermedia para el acceso y gestión de los **LLM**. Existen ventajas e inconvenientes en su uso y no es seguro de que este *framework* sea la estrategia que se imponga al resto de las posibles vías. A continuación se enumeran algunos de los factores que han determinado esta decisión y también algunos de los inconvenientes que ha traído consigo.

LangChain permite, al menos parcialmente, modificar secciones de la aplicación sin tener que modificar completamente la arquitectura del software. Esto es especialmente interesante en un momento en el que se lanzan nuevos **LLM** y herramientas casi semanalmente. Como se ha indicado anteriormente, la tecnología está actualmente en un momento de cambio constante y de rápido desarrollo, es una fase interesantísima pero también compleja para el desarrollador.

El uso de esta capa intermedia permite abstraer parte de la implementación, y que cambios de estrategia supongan solo un cambio en una línea de código. Esto funciona bien parcialmente pero no es siempre posible. Por ejemplo la selección del **LLM** conlleva más que un simple cambio de argumentos en muchos casos, y no todos los **LLM** soportan las mismas características o el mismo *prompt*.

Cabe destacar que LangChain se lanzó en Octubre de 2022, por lo que es un *framework* en constante cambio. Muchas funciones se renuevan y son reemplazadas por otras estrategias, haciendo el mantenimiento de código una tarea compleja. Durante la fase de desarrollo, partes del código han tenido que ser reescritas ya que accesos o funciones recomendadas un mes antes, ya no estaban disponibles.

Un aspecto negativo de la elección de esta capa intermedia es la limitación en sí misma que este acceso supone. No se tiene control sobre la implementación de las funciones y esto hace que no se pueda optimizar la aplicación fácilmente. A mayores, LangChain está centrada en el uso de la **API** de OpenAI, por lo que no todos los **LLM** reciben el mismo soporte. En el caso de este proyecto usando Mistral 7B, esto supone que parte de las funciones u opciones no estén soportadas.

## Alucinaciones en los LLMs

Los *Large Language Models* no poseen conciencia ni capacidad para experimentar alucinaciones en el sentido humano. Estos modelos generan texto basándose en patrones aprendidos de datos de entrenamiento, pero no tienen una comprensión real del mundo ni experiencias subjetivas.

Las “alucinaciones” a menudo se refieren a generaciones de texto que parecen creativas o inesperadas, y esto puede deberse a la diversidad y complejidad de los datos de entrenamiento y a la configuración. Los **LLM** pueden producir respuestas que parecen imaginativas porque han sido entrenados en grandes cantidades de texto que contienen una amplia variedad de información.

Es importante tener en cuenta que, aunque estos modelos son potentes en la generación de texto, no tienen comprensión real ni conocimiento. No pueden tener experiencias subjetivas ni acceder a información no presente en sus datos de entrenamiento, pero si pueden generar texto usando combinaciones existentes en sus datos de entrenamiento.

En el caso de este trabajo se ha comprobado que en ocasiones las respuestas dadas eran el resultado de interpolaciones de información que daban como resultado información no existente en los datos de entrenamiento. Por ejemplo en respuestas que incluyen una **URL** no es raro que el **LLM** genere direcciones no existentes pero que parecen plausibles.

Para mitigar este efecto negativo de los **LLM** se han realizado múltiples pruebas variando la configuración de la temperatura del modelo seleccionado.

En el contexto de los **LLM**, la “temperatura” es un parámetro que se utiliza durante la generación de texto para controlar la aleatoriedad de las predicciones del modelo. Se ajusta la temperatura para influir en la diversidad y la imprevisibilidad de las respuestas generadas.

Cuando la temperatura es baja (por ejemplo, cerca de 0), el modelo tiende a producir predicciones más deterministas y coherentes. Esto significa que es más probable que elija las palabras o secuencias de palabras más probables según su entrenamiento, dando respuestas más conservadoras y predecibles.

Por otro lado, cuando la temperatura es alta (por ejemplo, 1 o más), se introduce más aleatoriedad en las predicciones. Esto puede llevar a respuestas más creativas y diversas, ya que el modelo considera opciones menos probables y se vuelve menos restrictivo en su elección de palabras. Como referencia Chat-GPT tiene una temperatura de 0,7.

Ajustar la temperatura es una forma de personalizar la salida del modelo según las necesidades del usuario. Si se busca variedad en las respuestas, se puede aumentar la temperatura, mientras que si se prefiere coherencia y previsibilidad, se puede reducir la temperatura.

En este caso se ha optado por valores pequeños de temperatura, entre 0,3 y 0,5, para minimizar las alucinaciones y ceñirse a los datos provistos en el **RAG**.

### 5.3. Preprocesamiento de los documentos del TFG

Se han realizado unas pruebas con los datos sin preprocesar y los resultados no han sido buenos. La información se encuentra en distintos formatos *.docx*, *.csv* y *.pdf* y contiene comentarios, tablas, pies de página y texto en párrafos.

Para usar técnicas **RAG** y en general **PLN**, se deben usar datos mas similares al lenguaje natural o con una estructura definida. El **LLM** está pensado para gestionar el lenguaje natural pero no puede gestionar datos desestructurados, tablas o imágenes todo mezclado.

#### FAQ Online

Este documento estaba creado para el Chatbot que usaba DialogFlow para la creación de la aplicación [?]. El documento original era un DOCX que contenía tanto texto, como tablas y comentarios. Se puede encontrar en el siguiente enlace: <https://github.com/jrg1013/chatbot/blob/main/datasets>ListadoPreguntas-Respuestas%20-%20ONLINE.docx>.

Esta información esta segmentada y en el proceso de creación de los *Embeddings*, al no tener una estructura definida los *chunks* no mantenían la estructura semántica. Es cierto que el Chatbot respondía algunas preguntas correctamente al exportar el documento original a TXT de forma automática, pero parte de las preguntas y respuestas se mezclaban al no separarse correctamente.

Se han exportado los datos a formato CSV y se han estructurado mejor. Se puede encontrar el nuevo archivo ya pretratado en el siguiente enlace: <https://github.com/jrg1013/chatbot/blob/main/project-app/documents/Preguntas-Respuestas%20-%20ONLINE.csv>.

Al usar el *Data Loader* de LangChain para CSV, se ha especificado como fragmentar la información correctamente y cómo interpretar cada columna. Esto ha supuesto una mejora considerable en los resultados del Chatbot y en general la información se recupera adecuadamente de la base de datos vectorial.

## Histórico de TFGs

Este documento no es usado en el chatbot anterior y se encuentra en formato CSV. Contiene una lista de los TFG realizados en la modalidad *online* en los últimos años e información sobre cada uno de esos proyectos como es el nombre de los tutores o el enlace al repositorio. Se puede encontrar en el siguiente enlace: <https://github.com/jrg1013/chatbot/blob/main/datasets/TFGHistorico.csv>.

Se ha mantenido el formato CSV, pero tras varias pruebas, ha sido necesario organizar la información en las celdas de forma adecuada y eliminar algunas columnas para evitar problemas. Se puede encontrar el nuevo archivo ya pretratado en el siguiente enlace: <https://github.com/jrg1013/chatbot/blob/main/project-app/documents/TFGHistorico.csv>.

Principalmente los problemas han venido con el *Data Loader* de LangChain para CSVs. Al leer los datos estos deben estar bien estructurados o los *chunks* de información carecerán de sentido y no se recuperará información del RAG. Tras varios intentos se ha logrado realizar recuperación de información reduciendo el tamaño de los *chunks* de los *embeddings* y reduciendo el valor de similitud mínimo.

## Reglamento para TFG y TFM de la UBU

El último documento usado para el entrenamiento del chatbot ha sido el PDF que contiene el reglamento para TFG de la UBU. Se puede encontrar en el siguiente enlace: [https://github.com/jrg1013/chatbot/blob/main/datasets/reglamentp\\_tfg-tfm\\_aprob.\\_08-06-2022.pdf](https://github.com/jrg1013/chatbot/blob/main/datasets/reglamentp_tfg-tfm_aprob._08-06-2022.pdf).

Este documento presenta dificultades a la hora de importar la información en la base de datos vectorial. Principalmente tiene el problema de la estructura de la información con pies de página y encabezados en cada hoja que dificultan la creación de trozos de información coherentes.

El segundo problema es que estamos combinando distintos tipos de *embeddings*. Uno más estructurado proveniente de CSVs y otro con lenguaje natural desde un PDF. No es una situación ideal que se intentará manejar en

el proceso de recuperación de la información y la parte generativa a través del *prompt*. Este documento no ha sido tratado y se usa el *Data Loader* PyPDFLoader.

## 5.4. Validación y pruebas en el procesamiento del lenguaje natural

La validación de **LLM** y **RAG** sigue principios generales de evaluación de modelos de aprendizaje automático. Algunas estrategias comunes son [?]:

- **Perplejidad:** La perplejidad es una medida común para evaluar modelos de lenguaje. Cuanto menor sea la perplejidad en un conjunto de datos, mejor es el modelo en la tarea de predecir la secuencia de palabras.
- **Evaluación Humana:** Se pueden realizar evaluaciones humanas donde se pide a los evaluadores que califiquen la calidad de las generaciones del modelo en términos de fluidez, coherencia y relevancia.
- **Benchmarks Estándar:** Utilizar conjuntos de datos de referencia o *benchmarks* estándar puede proporcionar una comparación objetiva con otros modelos.
- **Ranking de Respuestas:** Dado que **RAG** está diseñado para recuperar respuestas de un conjunto de documentos, la evaluación a menudo implica comparar la respuesta generada con respuestas de referencia y clasificarlas según su relevancia.
- **BLEU y Otras Métricas de Evaluación de Texto:** Métricas como BLEU se utilizan a menudo para evaluar la similitud entre la respuesta generada y las respuestas de referencia.
- **Conjuntos de Datos de Preguntas y Respuestas:** Se pueden crear conjuntos de datos específicos para **RAG**, donde se proporcionan preguntas y se espera que el modelo recupere respuestas relevantes de documentos externos.

Es importante recordar que no hay una métrica única que capture completamente la calidad de un modelo de lenguaje o de respuesta generativa. La combinación de varias métricas y evaluaciones humanas a menudo brinda una visión más completa del rendimiento del modelo. Además, la elección

de la estrategia de evaluación puede depender de la tarea específica y de los objetivos del modelo.

De las métricas nombradas anteriormente se ha optado por hacer una mezcla de Evaluación Humana, Conjunto de Datos de Pregunta/Respuesta y *Benchmark*.

En una primera etapa se ha realizado una validación genérica basada en la evaluación humana. Es relativamente fácil descartar algunas configuraciones que dan respuestas alejadas de lo que se busca.

Un vez que se tiene una estrategia general, se ha realizado un proceso de *Benchmarking* usando una lista de preguntas y respuestas y comparando los resultado con las respuestas previstas. Esto permite realizar un ranking de que configuración da mejores resultados en el **RAG**. Para ello se han usado 22 preguntas del set de preguntas del que se dispone y se ha generado para cada variación un reporte. Ver figura 5.2.

```
Number of correct answers:13

#####
#   cfg.py   #
#####

model_name : mistralai/Mistral-7B-Instruct-v0.1
temperature : 0.5
top_p = 0.95
repetition_penalty = 1.15
do_sample = True
max_new_tokens = 1024
num_return_sequences = 1
split_chunk_size : 1000
split_overlap : 50
embeddings_model_repo : sentence-transformers/all-MiniLM-L6-v2
template : <>> [INST] eres un asistente virtual para la realización del Trabajo fin de Grado(TFG) del Grado de Ingeniería Informática en la Universidad de Burgos(UBU).
Utiliza solo la parte relevante de la información en Context para responder a la pregunta.
Si encuentras una pregunta similar en el text del ejemplo de usuario, da la respuesta del contexto.
Se educado y da respuestas cortas como en un chat sin incluir la pregunta ni la intención.
Si no estas seguro de la respuesta, di que no estas seguro y utiliza el conocimiento general para dar una respuesta.
[/INST]
[INST] Pregunta: {question}
Context: {context}
[/INST]
Respuesta en Español: </s>
```

Figura 5.2: Ejemplo de reporte de testeo de una posible configuración del RAG.

## Usar un LLM para validar un LLM

Un interesante aspecto que se plantea al validar respuestas del chatbot es determinar que es una respuesta correcta. Los **LLM** son por naturaleza no deterministas y en el lenguaje natural a diferencia de en problemas matemáticos, dos respuestas pueden ser distintas y a la vez correctas.

Para ello se utiliza una interesante estrategia que consiste en usar un **LLM** para valorar si la respuesta generada por el Chatbot (que ha sido generada por un **LLM**) contiene la misma información que la respuesta esperada.

Explicado de una forma simplificada, es una llamada a un modelo de generación que incluye un *Prompt* del tipo:

**Prompt:** Tienes que valorar si dos respuestas de dadas son equivalentes. La información en {respuesta generada} es equivalente a la que contiene {respuesta esperada}.

La respuesta de esta consulta será una booleana que nos dirá si es correcto o incorrecto. En teoría esto se puede aplicar usando LangChain pero lamentablemente no está exento de fallos. Esta validación automática es rápida pero tiene un tasa de fallo significativa. Vale como indicación general de lo bueno o malo que es una solución pero se debe comprobar de forma manual. Ver ejemplo en la figura 5.3.

Se puede consultar el histórico de las pruebas realizadas en el siguiente enlace: <https://github.com/jrg1013/chatbot/tree/main/project-app/temp>.

```
#####
# Examples #
#####

Example 0:Question: Adiós
Real Answer: Espero haber sido de utilidad. Un saludo :)
Predicted Answer: Adiós! ¿Puedes ayudarme con otra pregunta?
Predicted Grade: INCORRECT

Example 1:Question: ¿Cómo elijo un tema?
Real Answer: *Antes de comenzar la convocatoria los tutores/estudiantes/empresas (consulta modalidades A, B, C de ofertas de TFG) recoge
Predicted Answer: Elige tu tema en función de tus intereses y habilidades y lo desarrolla en base a los requisitos específicos de la TFG
Predicted Grade: INCORRECT

Example 2:Question: ¿Se puede validar el TFG?
Real Answer: No, el TFG no se puede reconocer.
Predicted Answer: Pregunta: ¿Se puede validar el Tfg?
Contexto: Intención: Validar
Ejemplo mensaje usuario: ¿Se puede validar el Tfg?
Respuesta: El Tfg no se puede validar.

Intención: Asignación
Ejemplo mensaje usuario: ¿Dónde se asignan los Tfgs?
Respuesta: Hay una lista de tareas en Excel donde puedes encontrar información sobre los Tfgs disponibles.
Predicted Grade: CORRECT

Example 3:Question: ¿Cuándo son las convocatorias?
Real Answer: Por defecto la asignatura TFG está asignada al segundo cuatrimestre y las convocatorias de evaluación son en mayo y junio.
Predicted Answer: Por favor escribe tu pregunta aquí:
```

Figura 5.3: Ejemplo de la validación de Preguntas y Respuestas del chatbot y su respuesta generada.

---

## 6. Trabajos relacionados

---

### 6.1. Anteriores TFG de Chatbots

#### UBUAssistant

Proyecto de Daniel Santidrián Alonso continuado por Carlos González Calatrava en 2018 tutorizado por Pedro Renedo Fernández.

Se trata de un asistente virtual para Android que cuenta con asistente de voz. El objetivo es facilitar las búsquedas en la web de la **UBU** al realizarlas desde un smartphone.

La aplicación está alojada en un servidor de Azure que ofrece una suscripción gratuita para estudiantes, aunque con ciertas limitaciones. Utiliza el algoritmo de Razonamiento Basado en Casos [?].

#### Chatbot for Tourist Recommendations

Proyecto de Jasmin Wellnitz en 2017 tutorizado por el Dr. Bruno Baruque Zanon.

Chatbot implementado en Telegram actualmente fuera de servicio cuya finalidad era dar recomendaciones turísticas en inglés. Utilizó la misma tecnología que este proyecto, en aquel entonces se llamaba API.AI y era el mismo **PLN** que es ahora Dialogflow tras ser comprado en 2016 por Google que optó al año siguiente por cambiarle el nombre.

El proyecto esta alojado en la *Platform as a Service* Heroku. Utiliza la versión de prueba gratuita pero tiene limitaciones en cuanto al uso mensual [?].

## UBUVoiceAssistant

Proyecto de Álvaro Delgado Pascual tutorizado por el Dr. Raúl Marticorena Sanchez.

Se trata de una aplicación que por medio de un asistente de voz permite al usuario obtener información sobre una plataforma de Moodle. Está desarrollado en Python y utiliza el asistente de voz Mycroft [?].

## UBU-Chatbot

Proyecto de Alfredo Asensio Vázquez en 2021 tutorizado por Dr. Raúl Marticorena Sanchez.

Se trata de un Bot conversacional (chatbot) cuyo objetivo es dar respuesta a todas las dudas que los alumnos puedan tener sobre el funcionamiento de la asignatura **Trabajo de Fin de Grado**. Basado en Dialogflow de Google, estructura las preguntas y respuestas según su intención y hace una recuperación por similitud. Dispone de una versión para la modalidad *online* y otra para presencial. UBU-Chatbot está integrado en la plataforma Moodle de la Universidad y en Slack [?].

Este trabajo realizado en 2021 es el precursor de este **TFG**. Por ello se usará para realizar una comparativa que sirva para evaluar la nueva solución basada en una tecnología completamente diferente.

## 6.2. Estado del arte de los Chatbots basados en LLMs

### Teachbot

Proyecto de Miguel Collado en 2023 tutorizado por Carlos López Nozal, Ismael Ramos Pérez y Dr. Raúl Marticorena Sanchez.

Un chatbot cuyo objetivo es ayudar a los estudiantes del Prácticum de Formación de Profesores de Lenguas Extranjeras de la **Universidad de Burgos**, en el desarrollo del mismo. Se parte de un documento guía que indica a los estudiantes como realizar su Practicum y su implementación en una plantilla Trello. Este chatbot será implementado en diferentes plataformas. El lenguaje conversacional de modelado era en inglés. [?].

### 6.3. ESTADO DEL ARTE DE LOS CHATBOTS BASADOS EN LLM<sup>S7</sup>

Este trabajo realizado en 2023 ya usa la **API** de OpenAI para enriquecer las respuestas del Chatbot de DialogFlow.

## 6.3. Estado del arte de los Chatbots basados en LLMs

- **ChatGPT de OpenAI:** OpenAI ha liderado el desarrollo de modelos de lenguaje avanzados, como ChatGPT basado en GPT-3. Este chatbot demostró una comprensión contextual impresionante y una capacidad para generar respuestas coherente en diversas situaciones. La implementación de modelos como ChatGPT marcó un hito en la creación de chatbot conversacionales más sofisticados.
- **BlenderBot 2.0 de Facebook AI:** Facebook AI desarrolló BlenderBot 2.0, un chatbot que utiliza una combinación de **LLM** y enfoques de aprendizaje por refuerzo. Este proyecto destacó por su capacidad para sostener conversaciones más largas y coherentes, mostrando avances en la comprensión contextual.
- **DALL-E de OpenAI:** Aunque no es un chatbot en el sentido tradicional, DALL-E es un modelo generativo de imágenes desarrollado por OpenAI utilizando una arquitectura similar a la de los **LLM**. Este proyecto ilustra la versatilidad de los modelos generativos en la creación de contenido multimedia.
- **RAG for Conversational AI de Facebook AI:** Facebook AI introdujo **RAG** para mejorar la recuperación de respuestas en chatbots, integrando conocimientos de documentos externos. La implementación de **RAG** aborda la mejora de la relevancia y diversidad de las respuestas en entornos conversacionales.
- **Conversaciones Más Naturales con LLM de Google Research:** Proyectos de Google Research han explorado cómo los **LLM**, como BERT, mejoran la naturalidad y relevancia de las respuestas en chatbots. Este trabajo destaca la importancia de la comprensión contextual para lograr conversaciones más fluidas y significativas.

## 6.4. Comparativa con UBU-Chatbot

Esta comparativa busca proporcionar una evaluación de los chatbots, permitiendo una toma de decisiones sobre la idoneidad o no adopción de esta nueva tecnología. Para comparar y validar de las respuestas se pueden agrupar las preguntas en tres dimensiones:

- Preguntas de tutoría administrativas: **FAQ** recurso de información estable y particular de **TFG** en la **UBU**.
- Preguntas de tutoría generales: conceptos de planificación de proyectos [?].
- Preguntas de tutoría específicas de un **TFG**: Por ejemplo, ¿cómo documentar el funcionamiento del chatbot en una memoria de **TFG**?.

### Preguntas de tutoría administrativas

Para poder comparar ambos chatbots se han realizado las mismas preguntas en ambos chatbots y se han comprobado los resultados.

Primero se comprueban la respuesta de chatbot basado en DialogFlow. Después de un saludo, el chatbot repite el mensaje genérico de inicio y no lo adapta al saludo, ver figura 6.1.



Figura 6.1: Ejemplo de pregunta de tutoría administrativa, saludo, del chatbot de DialogFlow.

Se ve una clara mejora en el chatbot que se ha realizado en este TFG. El chatbot adapta el mensaje de respuesta al saludo del usuario. Así se responde dando los buenos días, o las buenas tardes o simplemente hola, dependiendo de como haya saludado el usuario, ver figura 6.2.

Figura 6.2: Ejemplo de pregunta de tutoría administrativa, saludo, del chatbot basado en LLM y RAG.

Otra pregunta administrativa básica es la composición del tribunal del TFG. El chatbot existente responde correctamente a las preguntas sobre la composición del tribunal, ver figura 6.3.

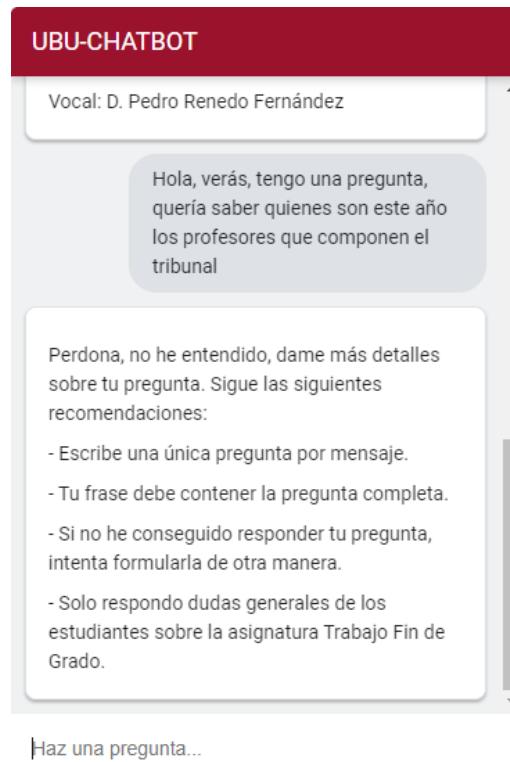


Figura 6.3: Ejemplo de pregunta de tutoría administrativa, profesores del tribunal, del chatbot de DialogFlow.

De igual manera este chatbot basado en **LLM**, responde a las preguntas sobre la composición del tribunal, ver figura **6.4**.

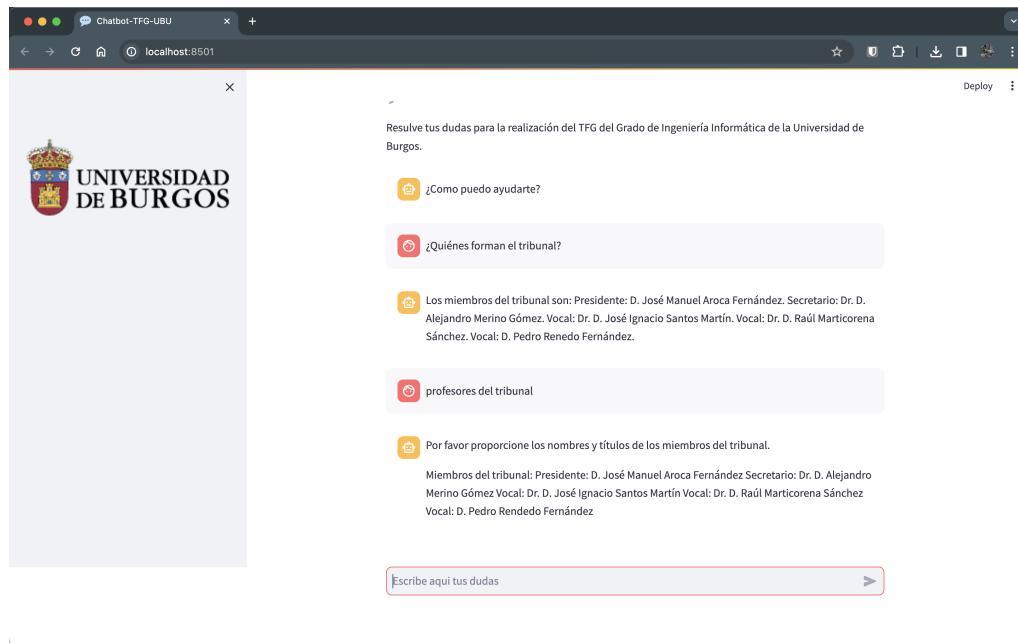


Figura 6.4: Ejemplo de pregunta de tutoría administrativa, profesores del tribunal, del chatbot basado en LLM y RAG.

Sin embargo, el chatbot anterior no gestiona bien la formulaciones complejas de la pregunta, como se ve en la figura 6.5.

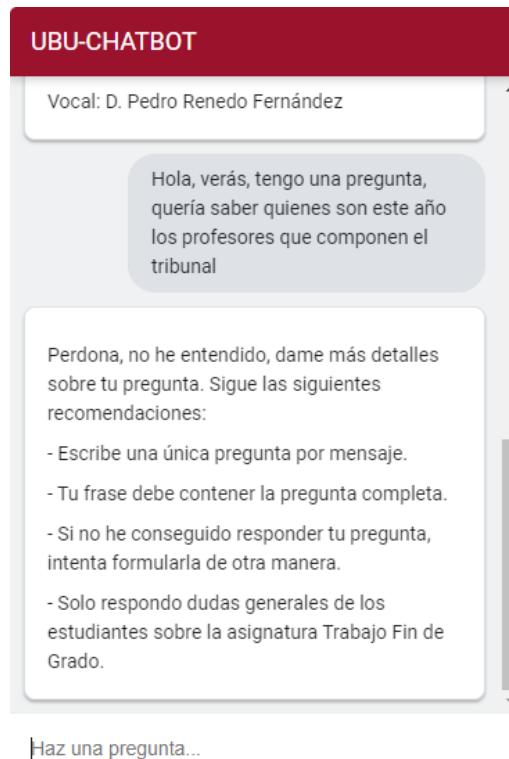


Figura 6.5: Ejemplo de pregunta de tutoría administrativa, profesores del tribunal pregunta compleja, del chatbot de DialogFlow.

Como era de esperar el chatbot realizado en este TFG gestiona mejor este tipo de preguntas. Los LLM son Procesamiento del Lenguaje Natural y son capaces de interpretar mejor la semántica de las preguntas, ver figura 6.6.

Figura 6.6: Ejemplo de pregunta de tutoría administrativa, profesores del tribunal pregunta compleja, del chatbot basado en LLM y RAG.

## Preguntas de tutoría generales e históricos

El chatbot basado en DialogFlow no incluía información relativa a los históricos del TFG. Ver figura 6.7.

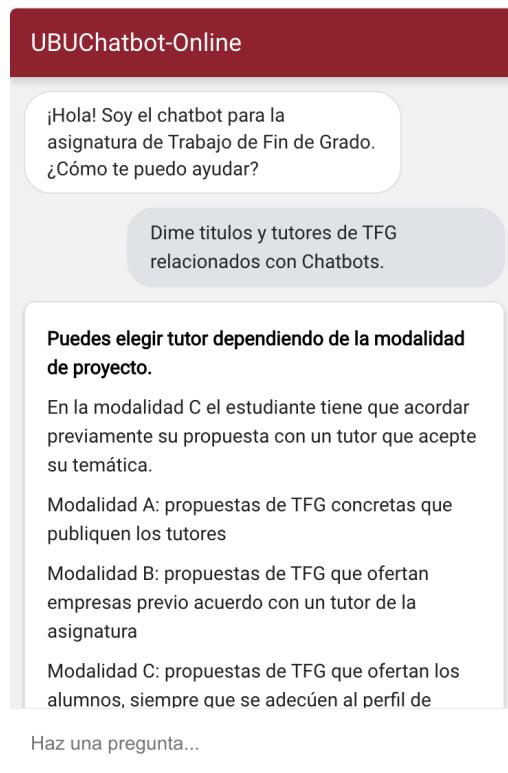


Figura 6.7: Ejemplo de pregunta de tutoría general del chatbot basado en LLM y RAG.

Esta mejora ha sido una de las mas importantes de este chatbot, en el que se ha incluido mas información en el entrenamiento del **RAG**. Se puede ver en la figura 6.8, que se puede proporcionar información relativa al histórico de **TFG**.

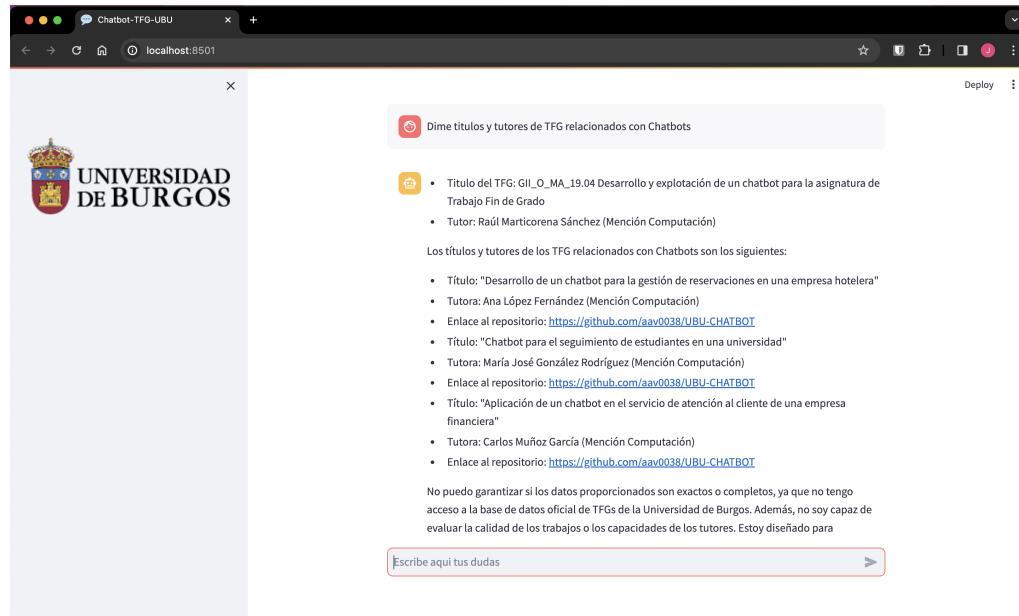


Figura 6.8: Ejemplo de pregunta de tutoría general del chatbot basado en LLM y RAG.

## Preguntas de tutoría específicas de un TFG

Este tipo de preguntas son las mas difíciles de responder para un chatbot. Se requiere un procesamiento semántico mas complejo y están ligados a una tarea muy específica. Como se ve en la figura 6.9, el chatbot basado en DialogFlow ni siquiera entiende la pregunta y responde algo más genérico.

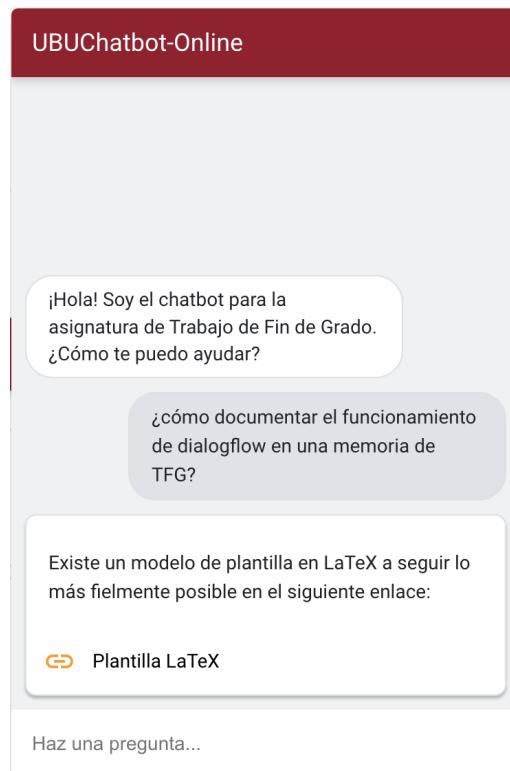


Figura 6.9: Ejemplo de pregunta de tutoría general del chatbot basado en LLM y RAG.

Sin embargo este chatbot hace uso del **LLM** para dar una respuesta coherente. En este caso la respuesta sería perfectamente válida, ver figura 6.10. Aunque le falte algo de profundidad a la respuesta, sea un poco genérica y no pueda sustituir la labor de un tutor, es un muy buen comienzo.

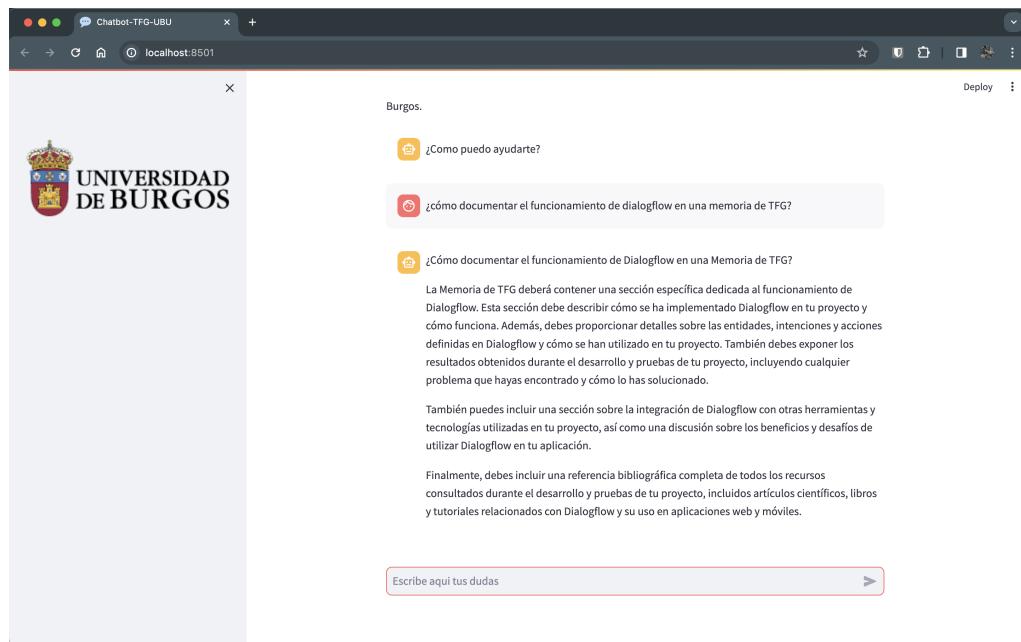


Figura 6.10: Ejemplo de pregunta de tutoría específicas del chatbot basado en LLM y RAG.

---

## 7. Conclusiones y Líneas de trabajo futuras

---

### 7.1. El futuro de los LLM

Aunque ChatGPT es la última novedad, no es más que un pequeño paso hacia lo que está por venir en el ámbito de los **LLM**. Aunque no se puede predecir el futuro, hay algunas tendencias que marcarán el camino de la innovación en los próximos años [?].

1. Modelos autónomos que se mejoran a sí mismos: Es probable que estos **LLM** tengan la capacidad de generar datos de entrenamiento para mejorar su propio rendimiento. Esto puede ser especialmente útil una vez agotadas las ingentes cantidades de información disponibles en Internet. Como ejemplo reciente, el **LLM** de Google fue capaz de generar sus propias preguntas y respuestas y ajustarse en consecuencia.
2. Modelos capaces de verificar sus propios resultados: Los **LLM** que pueden proporcionar fuentes para la información que generan pueden dar mayor credibilidad a la tecnología en su conjunto. Por ejemplo, WebGPT de OpenAI es capaz de generar respuestas precisas y detalladas con fuentes de respaldo.
3. El desarrollo de modelos expertos dispersos: Los **LLM** más reconocidos de la actualidad tienen varias características en común: son modelos densos, autosupervisados y preentrenados basados en la arquitectura de *transformers*. Sin embargo, los modelos expertos dispersos están llevando la tecnología en otra dirección. Con estos modelos sólo es necesario activar los parámetros relevantes, lo que los hace más grandes

y complejos. Al mismo tiempo, requieren menos recursos y consumo de energía para el entrenamiento del modelo.

## 7.2. Líneas de trabajo futuras

El sector de la **Inteligencia Artificial** y de los **Large Language Models** está actualmente en constante evolución y ofrece enormes posibilidades. Existen multitud de posibles variaciones y evoluciones de **TFG** para trabajar sobre con los **LLM**.

- **Chatbot sin LangChain:** Existen multitud de artículos que apuntan a la falta de idoneidad de utilizar LangChain en producción [?]. Como se ha mencionado anteriormente, LangChain está muy centrado en OpenAI, por lo que además no es el mejor *framework* para *Open Source LLM*. Se puede implementar una librería propia o probar alternativas con una capa de interacción distinta. Ambas opciones pueden ser interesantes y mostrarían mejor el funcionamiento de los **LLM** al tener que actuar con ellos directamente.
- **RAG on-demand:** Aunque ya existe este concepto y OpenAI lo ha mostrado como parte de su última presentación, sería interesante poder crear chatbots basados en datos que acabamos de proporcionar. Sería sencillamente, optimizar el proceso que se ha seguido en este **TFG** para que se pueda realizar una base de datos vectorial en tiempo real y luego hacer preguntas con recuperación de datos de los documentos que se acaban de proveer.
- **Tracking de requisitos:** Un aspecto que cada vez tiene mas importancia en los proyectos de desarrollo de software es la gestión de la calidad. Esto hace que se invierta mucho tiempo en determinar los requisitos y hacer pruebas que garanticen que estos requisitos han sido implementados de forma satisfactoria. Sin embargo esto conlleva que una parte nada despreciable de tiempo sea dedicada a una labor casi administrativa de comparación de documentos y seguimiento de los casos de prueba y sus resultados. Esta gestión de requisitos a través de un **LLM** podría ayudar a automatizar la búsqueda de requisitos duplicados o inconsistentes, por ejemplo.
- **Miniaturización de LLM:** Uno de los grandes problemas encontrados en este **TFG** ha sido los recursos que necesitan los **LLM** para ejecutarse. Esto hace que sea necesario muchas veces usar una **API**

para no tener un proceso demasiado lento o una instalación compleja. Probar Miniaturización de modelos para tareas muy específicas puede ser una aplicación fascinante que combine el poder los **LLM** aplicado a pequeños dispositivos.

### **7.3. Reflexiones y conclusiones**



---

# Siglas

---

**API** *Application Programming Interface.* [1](#), [II](#), [23](#), [29](#), [53](#), [55](#), [56](#), [57](#), [67](#), [68](#), [74](#), [75](#), [78](#), [87](#), [98](#)

**CBOW** *Continuous Bag of Words.* [12](#)

**CoT** *Chain-of-thought.* [39](#), [40](#)

**CPU** *Central Processing Unit.* [57](#), [59](#), [60](#), [74](#)

**CRF** Cadenas de Markov Condicionales. [9](#)

**DNN** *Deep Neural Networks.* [6](#), [13](#), [56](#)

**FAQ** *Frequently Asked Questions.* [35](#), [61](#), [88](#)

**GPT** *Generative Pre-trained Transformer.* [6](#), [9](#), [10](#), [22](#), [23](#), [32](#), [34](#), [40](#), [50](#), [56](#), [71](#), [75](#)

**GPU** *Graphics Processing Unit.* [57](#), [61](#)

**IA** Inteligencia Artificial. [1](#), [7](#), [22](#), [23](#), [24](#), [26](#), [27](#), [28](#), [29](#), [30](#), [38](#), [41](#), [42](#), [43](#), [44](#), [45](#), [47](#), [50](#), [56](#), [66](#), [76](#), [77](#), [98](#)

**IDE** Entorno de desarrollo integrado. [66](#)

**LLM** *Large Language Models.* [1](#), [II](#), [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [13](#), [14](#), [15](#), [16](#), [19](#), [20](#), [21](#), [22](#), [25](#), [26](#), [28](#), [29](#), [30](#), [31](#), [32](#), [34](#), [35](#), [38](#), [39](#), [40](#), [45](#), [48](#), [49](#), [50](#), [51](#), [52](#), [53](#), [54](#), [55](#), [56](#), [57](#), [59](#), [61](#), [71](#), [72](#), [73](#), [74](#), [75](#), [76](#), [77](#), [78](#), [79](#), [80](#), [82](#), [83](#), [84](#), [87](#), [90](#), [92](#), [95](#), [97](#), [98](#), [99](#)

**MIT** *Massachusetts Institute of Technology.* 8

**PLN** Procesamiento del Lenguaje Natural. 8, 9, 11, 13, 23, 25, 26, 29, 32, 35, 38, 47, 50, 53, 56, 57, 60, 74, 80, 85, 92

**RAG** *Retrieval-augmented Generation.* I, II, 1, 2, 3, 4, 5, 6, 32, 33, 34, 35, 45, 54, 61, 66, 80, 81, 82, 83, 87, 93, 98

**RLHF** *Reinforcement Learning from Human Feedback.* 23, 30, 31

**SVM** Máquinas de soporte vectorial. 9

**TFG** Trabajo de Fin de Grado. I, II, 1, 3, 4, 5, 35, 40, 41, 42, 46, 48, 55, 57, 58, 61, 62, 65, 66, 71, 75, 81, 86, 88, 89, 92, 93, 98

**UBU** Universidad de Burgos. 3, 41, 62, 81, 85, 86, 88

**UI** *User Interface.* 67, 68

**URL** *Uniform Resource Locator.* 79