Proyecto Final

Procesamiento del Lenguaje Natural
Máster en Inteligencia Artificial Aplicada
Curso 2025/2026

2 de diciembre de 2025

Índice

1. Introducción

Los grandes modelos del lenguaje–del inglés, Large Language Models (LLM)–han revolucionado por completo la inteligencia artificial, pero también la forma con la que afrontamos muchas tareas de la vida laboral y cotidiana. No obstante, su verdadero valor radica en su capacidad para ser aplicados a problemas concretos más allá de los usos convencionales, permitiendo soluciones avanzadas en áreas como la atención al cliente, la verificación de hechos y la asistencia en investigación.

Durante el curso hemos visto como la invención de la arquitectura de Transformer en 2017 revolucionó por completo el procesamiento del lenguaje natural–del inglés, Natural Language Processing (NLP)–con la introducción de mecanismos de atención para manejar dependencias a largo plazo, eliminando la necesidad de modelos secuenciales como las redes recurrentes. Además de la arquitectura del Transformer inicial (**Encoder-Decoder**), útil para tareas de transformación de secuencias como la traducción automática o el resumen de texto, existen otras dos variantes de esta arquitectura: **Encoder** (e.g., BERT), ideales para tareas que requieren análisis de texto como clasificación o respuesta a preguntas; y **Decoder** (e.g., modelos GPT), adecuada para tareas generación de texto.

Es a partir de la ampliación y adaptación de estos últimos modelos Decoder como se logró principalmente la transacción hacia los grandes modelos del lenguaje. Al aumentar de manera significativa el número de parámetros y los datos de entrenamiento, los modelos ganaron capacidad para capturar patrones lingüísticos más complejos, conocimientos generales del mundo y diferentes estilos lingüísticos.

No obstantes, los LLMs no responden siempre con información correcta, sino que pueden "alucinar", esto es, darnos una respuesta incorrecta o inventada. Aquí es dónde entran en contexto los sistemas **RAG (Retrieval Augmented Generation)**, que buscan reducir estas alucinaciones, así como mejorar la respuesta del sistema mediante la incorporación de fuentes adicionales de conocimiento. Son también especialmente relevantes cuando queremos que un LLM de respuestas concretas respecto a una fuente de documentación específica, no sobre los conocimiento generales que haya podido aprender durante su preentrenamiento.

Objetivo. El objetivo de este proyecto es que los estudiantes adquieran comprensión de flujos de trabajo iterativos que incorporan modelos del lenguaje y sistemas RAG para el desarrollo de tres aplicaciones prácticas:

A. Un chatbot que permita responder preguntas sobre una base de datos documental

B. Un sistema de verificación de hechos

C. Un sistema de asistencia en investigación académica

Instrucciones preliminares

- Cada aplicación debe cumplir un conjunto de **requisitos funcionales obligatorios**.
- Los estudiantes **deben usar una base de datos vectorial** para implementar el sistema RAG. Las consultas directas a fuentes como Wikipedia **no son válidas**.
- Todos los sistemas deben ser **evaluados según los criterios especificados** (Tabla 1).
- Los estudiantes pueden mejorar su sistema con **funcionalidades adicionales**, ya sea incorporando alguna de las ideas sugeridas o proponiendo las suyas propias. Aunque estas funcionalidades son opcionales, no implementar ninguna limitará la calificación máxima a 8.
- **Para el proyecto básico, solo se pueden usar modelos de lenguaje de código abierto.**
- En todos los casos, se puede desarrollar un frontend básico con herramientas como Streamlit[1] para mejorar la presentación, aunque esto **no** se considerará parte de la mejora en términos de NLP.

Grupos de trabajo. Los estudiantes trabajarán en grupos de cuatro o cinco integrantes (**3 grupos de 4 integrantes y 7 de 5**). Cada grupo deberá elegir una de las tres aplicaciones indicadas antes del 24 de noviembre, con un máximo de 4 grupos por aplicación. La elección de grupo se hará a través de Aula Global.

[1] https://github.com/streamlit/streamlit

Entrega. Los alumnos deberán preparar:

1. Un repositorio de GitHub con el código implementado y debidamente comentado.
2. Un documento PDF de **máximo dos páginas** que incluya:
   - El enlace al repositorio de GitHub.
   - Una descripción concisa de las decisiones de diseño, incluyendo (pero no limitado a) la selección del conjunto de datos, las tecnologías empleadas, la configuración de la base de datos y cualquier funcionalidad adicional implementada.
   - Detalles de configuración tanto del LLM como de la base de datos vectorial, así como técnicas de optimización (ajuste de parámetros, estrategias de prompting, etc.).
   - Una explicación de cómo el sistema cumple los requisitos funcionales, cómo se mitigan las alucinaciones y cualquier otro detalle que hagan el trabajo destacable o innovador.
   - Evaluación del sistema basada en los criterios de la Tabla 1. En la Sección 3.5 se sugieren métricas para evaluar estas dimensiones, aunque también pueden usarse métricas alternativas. Declaraciones vagas como *"el sistema funciona porque proporciona respuestas correctamente formateadas o parece responder adecuadamente"* son insuficientes: deben proporcionarse métricas específicas que validen el rendimiento.
   - Conclusiones y limitaciones del sistema. La omisión de cualquiera de estas secciones supondrá una penalización. Deben ser secciones específicas y directamente relacionadas con el trabajo realizado: afirmaciones genéricas como *"una línea futura es reducir las alucinaciones del sistema"* son insuficientes. En su lugar, el informe debe analizar por qué se producen las alucinaciones, en qué circunstancias y proporcionar sugerencias concretas de mejora.
3. Una presentación que tendrá lugar en enero.

Criterio	**Chatbot**	**Sistema de Verificación de Hechos**	**Asistencia a la Investigación**
Calidad del Sistema RAG	Recuperación de información relevante y generación de respuestas coherentes y útiles.	Recuperación de evidencia relevante y obtención de veredictos precisos y exactos sobre la veracidad de las afirmaciones.	Recuperación de artículos clave y comparación precisa con trabajos de investigación previos.
Cobertura Documental	Porcentaje de consultas respondidas usando la información de la base de datos.	Capacidad para encontrar evidencia suficiente para verificar las afirmaciones.	Capacidad para recuperar artículos relevantes relacionados con el tema de investigación.
Tiempo de Respuesta	Velocidad a la que se generan las respuestas.	Velocidad con la que se verifican las afirmaciones y se proporcionan veredictos.	Velocidad de recuperación de artículos y generación de comparaciones.

Cuadro 1: Criterios de evaluación para cada aplicación.

El PDF será el único documento que se subirá a Aula Global, y solo un miembro del grupo deberá entregarlo. La fecha límite será el 23 de Diciembre, a las 23,59 horas.

Evaluación. El proyecto se evaluará de acuerdo con los criterios siguientes:

- **Proyecto básico** (8 puntos)
  - Cumplimiento de requisitos funcionales (2)
  - Metodología (2)
  - Calidad del informe (1)
  - Calidad del código (1)
  - Presentación (2)
- **Extensión**: **Máximo 2 puntos** dependiendo del alcance del trabajo. La calificación se calculará como $2 \times \alpha$, donde α será asignado por el profesor de la asignatura en función del alcance del trabajo seleccionado.
  - Originalidad $(0, 4)$
  - Calidad del trabajo y metodología $(1, 6)$

2. Descripción de las aplicaciones prácticas

2.1. Chatbot para responder preguntas de una base de datos documental

Un chatbot sin RAG puede ofrecer respuestas imprecisas al no tener acceso a los documentos en cuestión, limitándose a información general preentrenada. En este ejercicio, nos centraremos en superar esta limitación al construir un chatbot con RAG.

Requisitos funcionales. El chatbot implementado debe cumplir los siguientes requisitos:

- Las respuestas deben incluir el documento o los documentos específicos que sustentan la respuesta.
- Si la base de datos no contiene un documento relevante, el chatbot no debe proporcionar una respuesta.
- Las respuestas deben estar en el mismo idioma que la pregunta.

A continuación se presenta un ejemplo de la interacción esperada con el chatbot:

Usuario:
¿Cuáles son los requisitos para renovar el DNI en España?

Chatbot:
De acuerdo con el documento del BOE con referencia BOE-A-2005-21163, la renovación debe llevarse a cabo personalmente por el titular del DNI. Puede consultar más información aquí.

Usuario:
¿Cómo se puede acceder a las becas de estudio?

Chatbot:
Lo siento, no tengo información sobre eso en los documentos disponibles del BOE. Por favor, revisa el BOE directamente o proporciona más detalles para una búsqueda más precisa.

Base de datos. Como base de datos documental, se puede utilizar cualquier colección de documentos (en texto plano o PDF) sobre la cual un usuario podría estar interesado en hacer preguntas. A continuación, se proporcionan algunos ejemplos, pero los estudiantes pueden elegir otros conjuntos de documentos que les resulten más interesantes o relevantes.

- Documentos del Boletín Oficial del Estado (BOE)[2]. Aquí[3] podéis acceder a una colección de 216,000 documentos en formato HTML o XML.
- Documentos del plan de estudios del Máster en Inteligencia Artificial Aplicada (o de otros másteres de la UC3M u otra universidad).
- Manuales de usuario y documentación de productos electrónicos en línea, como la base de datos de manuales de Sony[4]. Aquí podéis acceder a manuales de usuario y guías de productos, como cámaras o televisores, en formato PDF.

Funcionalidades adicionales.

- Mostrar sugerencia de temas similares. Para ello se puede entrenar un modelo de tópicos sobre la base de datos, y mostrar sugerencias de tópicos / preguntas una vez que el usuario haga una pregunta.
- Añadir la opción de generar resúmenes automáticos de los cuales se ha extraído información, para que el usuario pueda obtener una versión condensada del contenido recuperado. Para este se puede utilizar un modelo de Transformer fine-tuneado en el corpus de entrenamiento y/o un LLM.
- Permitir al chatbot manejar consultas en varios idiomas, traduciendo tanto las preguntas como las respuestas, si es necesario, mediante modelos preentrenados (o finetuneados) de traducción.

[2] https://www.boe.es/
[3] https://github.com/PlanTL-INTELCOMP/DBimport/tree/master/data_Law_BOE
[4] https://www.sony.com/electronics/support

2.2. Sistema de verificación de hechos

Esta aplicación se centra en desarrollar un sistema RAG que verifique automáticamente la veracidad de una declaración utilizando un corpus específico. El sistema recuperará información relevante del corpus y empleará un LLM para evaluar la veracidad de la declaración.

Requisitos funcionales. El sistema debe cumplir con los siguientes requisitos:

- Proporcionar una respuesta clara que explique la veracidad de la declaración.
- Acompañar las respuestas con las fuentes y fragmentos del corpus que justifiquen el veredicto.
- Si el sistema no puede verificar la veracidad de la declaración por falta de información, debe indicarlo sin inventar una respuesta.
- Las respuestas deben estar en el mismo idioma que la pregunta.

A continuación se presenta un ejemplo de la interacción esperada con el sistema:

Usuario:
El cambio climático no tiene impacto en la frecuencia de eventos climáticos extremos.

Sistema:
Falso. Según estudios científicos recientes, el cambio climático está aumentando la frecuencia de eventos climáticos extremos como huracanes, olas de calor e inundaciones.

- Aumento de las temperaturas: el cambio climático está provocando un aumento de las temperaturas globales, lo que provoca olas de calor más frecuentes e intensas [...] [fuente]
- Aumento de las temperaturas: el cambio climático está provocando un aumento de las temperaturas globales, lo que provoca olas de calor más frecuentes e intensas [...] [fuente]

Usuario:
La tecnología 5G tiene un impacto directo en la salud humana.

Sistema:
Lo siento, no tengo suficiente información en el corpus para verificar esta afirmación.

Base de datos. Como fuente de información sobre la que corroborar los hechos, se puede utilizar cualquier base de datos textual que contenga información verídica. A continuación, se proporcionan algunos ejemplos, pero los estudiantes pueden elegir otros conjuntos de documentos que les resulten más interesantes o relevantes.

- Un dump de Wikipedia, que se puede obtener fácilmente a través de la librería `wikipedia-api`[5].
- PubMed. Conjunto de artículos científicos sobre temas relacionados con medicina o ciencias de la salud, que está disponible a través de su página web[6].
- COVID-19. Corpus de artículos académicos sobre COVID-19 y la investigación relacionada con el coronavirus, que se puede descargar fácilmente a través del siguiente repositorio[7].

Funcionalidades adicionales.

- Inclusión de la confianza del sistema que indique cuán seguro está el modelo sobre la veracidad de la declaración, basado en la evidencia recuperada del corpus.
- Comparación de prestaciones respecto a un sistema de verificación de hechos tradicional.
- Añadir la opción de generar resúmenes automáticos de los cuales se ha extraído información, para que el usuario pueda obtener una versión condensada del contenido recuperado. Para este se puede utilizar un modelo de Transformer fine-tuneado en el corpus de entrenamiento y/o un LLM.
- Permitir al chatbot manejar consultas en varios idiomas, traduciendo tanto las preguntas como las respuestas, si es necesario, mediante modelos preentrenados (o finetuneados) de traducción.

[5] https://pypi.org/project/Wikipedia-API/
[6] https://ftp.ncbi.nlm.nih.gov/pubmed/baseline/
[7] https://github.com/allenai/cord19

2.3. Sistema de asistencia en investigación académica

Una parte crucial de la investigación es la comparación del trabajo actual con investigaciones previas (revisión del estado del arte). Esta aplicación busca asistir a investigadores permitiendo que introduzcan una línea de investigación, y el sistema les proporcionará artículos relevantes al tema. Además, el sistema será capaz de responder en qué se diferencia lo ya existente de lo que el investigador propone.

Requisitos funcionales.

- El sistema debe recuperar artículos relevantes en función de la línea de investigación introducida por el usuario.
- El sistema debe ser capaz de identificar y mostrar diferencias clave entre la línea de investigación del usuario y los trabajos existentes.
- Si el sistema no encuentra artículos relevantes, no debe inventarse artículos para dar una respuesta.
- Las respuestas deben estar en el mismo idioma que la pregunta. Como excepción, los nombres de los artículos podrán estar en el idioma original.

A continuación se presenta un ejemplo de la interacción esperada con el sistema:

Investigador:
Estoy investigando sobre implementaciones de modelos de tópicos neuronales federadas.

Sistema:
He encontrado varios artículos relevantes, incluyendo "Federated topic modeling" y "Federated non-negative matrix factorization for short texts topic modeling with mutual information". Estos trabajos se centran en la implementación de aproximaciones federadas de modelos de tópicos bayesianos, como LDA o NMF, pero ninguno ofrece una implementación basada en modelos de tópicos neuronales.

Investigador:
Estoy investigando sobre sistemas para la asistencia a investigadores.

Sistema:
Lo siento, pero no tengo más artículos específicos en el corpus actual sobre este tema. Esto no implica que no haya ninguno. Puedes intentar ampliar tu búsqueda en otras bases de datos.

Base de datos. Como fuente de información sobre la que buscar artículos académicos, nos tenemos que limitar a las tradicionales, como por ejemplo:

- Artículos pbulicados en el ACL. Por ejemplo, este dataset[8] contiene 80 000 artículos/pósteres datados hasta septiembre de 2022.
- PubMed. Conjunto de artículos científicos sobre temas relacionados con medicina o ciencias de la salud, que está disponible a través de su página web[9].
- arXiv. Kaggle[10] ofrece un dataset con más de 1.7 millones de resúmenes y metadatos de artículos de arXiv.

Funcionalidades adicionales.

- El sistema puede proporcionar métricas de similitud entre la idea propuesta y los artículos encontrados.
- Clasificación automática del estado del arte: El sistema puede organizar los artículos recuperados en diferentes categorías basadas en su relevancia, enfoque metodológico o contribuciones.
- Para cada artículo recuperado, el sistema puede generar resúmenes automáticos que destaquen los puntos clave. Para este se puede utilizar un modelo de Transformer fine-tuneado en el corpus de entrenamiento y/o un LLM.
- Permitir al chatbot manejar consultas en varios idiomas, traduciendo tanto las preguntas como las respuestas, si es necesario, mediante modelos preentrenados (o finetuneados) de traducción.

[8] ttps://paperswithcode.com/dataset/acl-anthology-corpus-with-full-text
[9] https://ftp.ncbi.nlm.nih.gov/pubmed/baseline/
[10] https://www.kaggle.com/datasets/Cornell-University/arxiv

3. Información complementaria

Esta sección introduce algunos conceptos relevantes para el desarrollo de la práctica.

3.1. Prompting

A pesar de contar con modelos Transformer preentrenados muy potentes, en muchas ocasiones es necesario realizar un fine-tuning para lograr buenos resultados en la resolución de tareas específicas. No obstante, el conocimiento general del lenguaje que poseen estos modelos preentrenados permite que el dataset necesario para el fine-tuning sea mucho más pequeño.

Sin embargo, no siempre es posible hacer fine-tuning, como cuando no tenemos acceso a los parámetros del modelo y solo podemos interactuar con él a través de una API. En estos casos, entra en juego el concepto de **"in-context learning" (ICL)**: la capacidad de los LLMs para resolver tareas utilizando el contexto proporcionado en la interacción, sin necesidad de reentrenar el modelo o actualizar sus pesos. El ICL aprovecha los ejemplos y las instrucciones dentro del mensaje para "aprender" a realizar una tarea.

Una técnica clave para aprovechar el ICL es el **prompting**, que consiste en diseñar cuidadosamente el texto (prompt, e.g.: *"Genera un texto de 3 párrafos sobre la Inteligencia Artificial"*) para guiar la respuesta del modelo hacia la tarea deseada. Dependiendo del caso de uso, a veces se prefiere prompting sobre fine-tuning, y viceversa[11]. Relacionado con esto está el concepto de **"prompt tuning"**, que busca optimizar los prompts para tareas específicas. Existen dos tipos: **"hard tuning"** y **"soft tuning"**. Mientras que en el hard tuning simplemente modificamos el prompt (e.g., variando los ejemplos), en el soft prompt tuning se concatena a los embeddings de entrada un tensor que es un parámetro a aprender. Existen varias técnicas de ICL, las cuales se resumen en el Cuadro 2.

Técnica	**Descripción**
Few-Shot Prompting	El LLM aprende a completar una tarea a partir de unos pocos ejemplos. La selección de ejemplos en el prompt afecta directamente los resultados; se deben considerar aspectos como la cantidad, orden, calidad de las etiquetas y formato de los ejemplos.
Zero-Shot Prompting	No se utilizan ejemplos en el prompt. Técnicas como Role Prompting, Style Prompting y System 2 Attention (S2A) son ejemplos de Zero-Shot.
Chain-of-Thought Generation	Impulsa al LLM a articular su razonamiento antes de dar la respuesta final. Incluye Few-Shot y Zero-Shot CoT. La primera usa una frase inductora ("Pensemos paso a paso"), mientras que la segunda utiliza ejemplos con cadenas de pensamiento. Mejora el rendimiento en tareas complejas.
Decomposition	Descompone un problema complejo en subpreguntas más simples. Similar al Chain-of-Thought, pero más centrado en la estructura del problema.
Ensembling	Utiliza diferentes prompts para resolver el mismo problema, y combina las respuestas en un resultado final (por ejemplo, mediante votación por mayoría). Reduce la varianza de los resultados pero aumenta el coste computacional.
Self-Criticism	El LLM critica sus propias respuestas (por ejemplo, preguntando si el resultado es correcto) o proporciona retroalimentación que se utiliza para mejorar la respuesta.

Cuadro 2: Resumen de técnicas de prompting para LLMs. Cada una de las técnicas indicadas incluye diversas subtécnicas. Para más información, puede consultar [10].

3.2. Parámetros clave en la generación de texto con LLMs

Al trabajar con LLMs como los de las familias GPT o LLaMA para la generación de texto, hay una serie de parámetros que nos permiten controlar las respuestas generadas. Entre ellos encontramos [2, 8]:

- **Temperatura.** Permite ajustar la aleatoriedad en las respuestas del modelo. Un valor de temperatura bajo (por ejemplo, 0.2) hará que el modelo genere texto de manera más determinista, eligiendo las palabras más probables. Por el contrario, un valor de temperatura alto (por ejemplo, 0.7) generará respuestas más variadas y creativas. El rango de valores típicos es entre 0 y 1.
- **Top-p.** También conocido como muestreo de núcleo, es un parámetro que define el tamaño del conjunto de tokens que se consideran a la hora de hacer el muestreo para la generación. Un valor alto de top-p significa que el modelo tendrá en cuenta una mayor cantidad de palabras posibles,

[11] Prompting vs Fine-tuning: https://www.prompthub.us/blog/fine-tuning-vs-prompt-engineering

incluyendo algunas menos probables, aumentando así la diversidad del texto generado sin introducir demasiada aleatoriedad. Este parámetro también tiene un rango entre 0 y 1.

- **Penalización por frecuencia.** Este parámetro controla la repetición de palabras dentro de la respuesta generada. A mayor penalización, menor será la probabilidad de que ciertos tokens aparezcan repetidamente en una misma respuesta. Suele tener un valor entre 0 y 2.
- **Penalización por presencia.** Similar a la penalización por frecuencia, este parámetro influye en la probabilidad de introducir nuevos temas o conceptos en una respuesta. Un valor más alto aumenta la probabilidad de que el modelo explore nuevos términos o tópicos que aún no se han mencionado, reduciendo la posibilidad de repetir tokens ya utilizados. Este parámetro también suele encontrarse entre 0 y 2.

3.3. Retrieval-Augmented Generation (RAG)

La mejor forma de entender el concepto de RAG es ponernos en el contexto de que estamos trabajando en una empresa tecnológica, la cual cuenta con documentación interna privada. La empresa quiere que sus trabajadores sean capaces de acceder a esta información de forma rápida, y para ello quisiera tener una plataforma que utilizara LLMs para dar respuestas a preguntas hechas por los trabajadores acerca de estos documentos. No obstante, un LLM tendría problemas para contestar a estas preguntas que son ad-hoc de los documentos de la empresa. Aquí es donde entra un sistema RAG [6].

Un sistema RAG, del inglés Retrieval Augmented Generation (Generación Aumentada por Recuperación), consiste en un sistema que permite utilizar información contextual extraída de una fuente de documentos a la hora de dar respuesta a una pregunta. El proceso es el siguiente: un usuario hace una pregunta a una IA a través de una interfaz conversacional y, antes de proporcionar una respuesta directa en base al conocimiento general que ha aprendido durante su entrenamiento, realiza un proceso de búsqueda en una base de conocimiento y utiliza dicha información para producir una respuesta más precisa y adaptada al contexto, sin necesudad de reetrenamiento. De esta manera, además, se reduce significativamente la posibilidad de que el modelo genere información incorrecta o inventada (i.e., que el modelo alucine).

3.3.1. Componentes de un sistema RAG

Para que el LLM pueda responder al usuario utilizando el contexto, un sistema RAG combina dos fases (ver Figura 2): la recuperación y la generación, asumiendo la previa indexación de los documentos en una base de conocimiento (ver Figura 1), típicamente una base de datos vectorial.

Recuperación. Esta fase es crucial para la calidad y relevancia de las respuestas proporcionadas por el LLM. Si no se recupera la información adecuada, la calidad de la respuesta será deficiente. Para obtener documentos relevantes, se pueden utilizar técnicas como la búsqueda por palabras clave, la búsqueda semántica basada en embeddings contextuales, o métodos más tradicionales de recuperación de información, como BM25.

Generación. En esta fase, el LLM genera una respuesta coherente y relevante utilizando los documentos recuperados durante la fase de recuperación (es decir, el contexto).

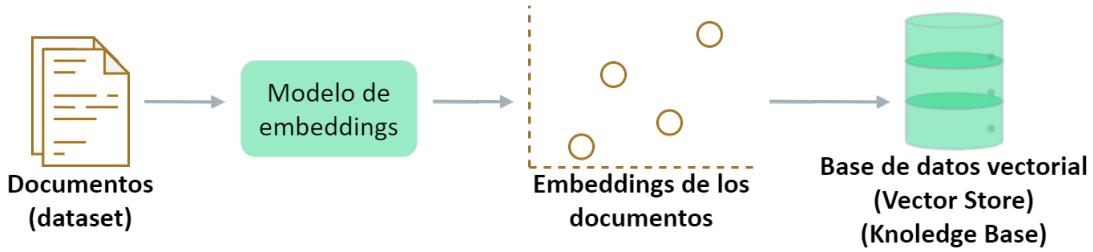

Figura 1: Indexado de los documentos en una base de conocimiento.

Existen varios métodos que permiten mejorar el rendimiento del sistema RAG básico descrito anteriormente; algunas de ellas se resumen en el Cuadro 3.

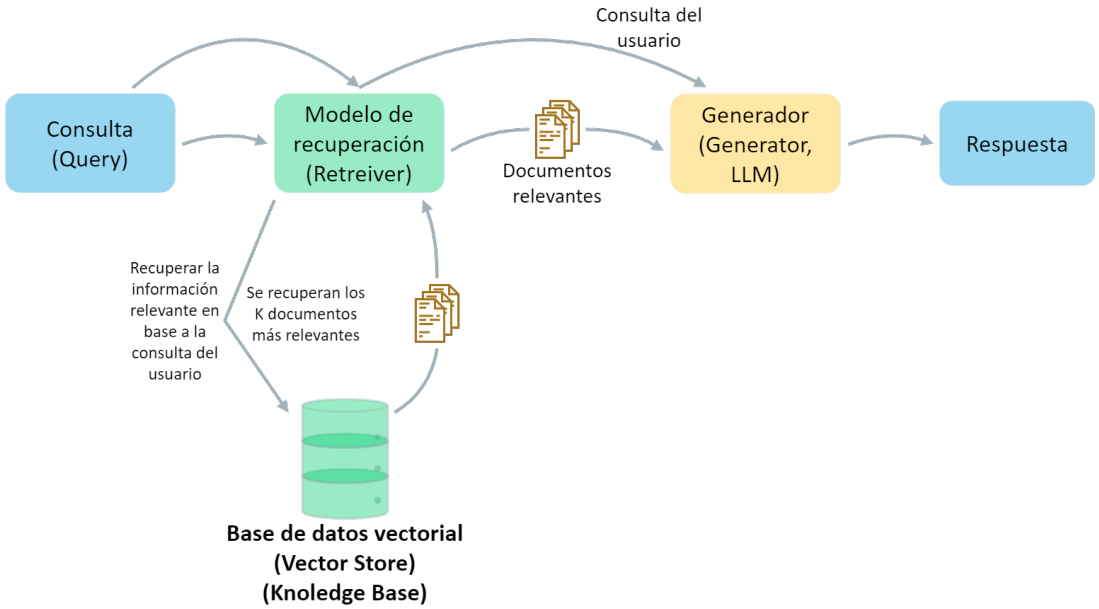

Figura 2: Esquema de un sistema de RAG. El proceso comienza con una consulta del usuario, que es enviada al modelo de recuperación (retriever), el cual busca en una base de datos vectorial (vector store) los documentos más relevantes utilizando técnicas de búsqueda semántica. Estos documentos relevantes son luego enviados a un LLM, que finalmente genera la respuesta en función de la consulta original y la información recuperada.

Técnica	**Descripción**
Fragmentación de texto	Dividir documentos grandes en fragmentos más pequeños para mejorar la recuperación de información, ya que los embeddings contextuales tienen un límite de contexto. Esto facilita recuperar información más relevante para una consulta dada.
Expansión de consultas	Generar varias subconsultas a partir de la consulta inicial, lo que aumenta las posibilidades de recuperar los documentos más importantes al asociar embeddings a múltiples consultas.
Reclasificación	Reordenar o filtrar los documentos recuperados basándose en su relevancia para la consulta del usuario. Un LLM puede ser utilizado para identificar la información más importante entre los documentos iniciales recuperados..

Cuadro 3: Técnicas para mejorar las prestaciones de un sistema RAG[12].

3.4. Bases de datos vectoriales

Para realizar la extracción de documentos de manera eficiente mediante búsqueda semántica, los embeddings deben almacenarse en una base de datos adecuada. Las bases de datos vectoriales son sistemas especializados diseñados para el almacenamiento y recuperación óptima de representaciones vectoriales de datos.

Algunos ejemplos de bases de datos de este tipo incluyen **FAISS** (Facebook AI Similarity Search)[12], **Chroma**[13], y **ColBERT**[14] (Contextualized Late Interaction over BERT)[4]. Nótese que ColBERT no es una base de datos vectorial en sí misma, sino una arquitectura que combina la búsqueda semántica eficiente con modelos de lenguaje como BERT. ColBERT se basa en una técnica conocida como "interacción tardía contextualizada", donde los documentos y las consultas son transformados en representaciones vectoriales antes de que se realicen las interacciones para la búsqueda[15].

3.5. Métricas de evaluación

Algunas métricas de evaluación que se utilizan en el contexto de sistemas RAG incluyen:

- **Recall@K:** Mide qué tan bien los documentos relevantes están clasificados entre los primeros K resultados [3].

[12] https://github.com/facebookresearch/faiss
[13] https://docs.trychroma.com/
[14] https://github.com/stanford-futuredata/ColBERT
[15] https://jina.ai/news/what-is-colbert-and-late-interaction-and-why-they-matter-in-search/

- **Mean Reciprocal Rank (MRR).** Evalúa el orden en que se presentan los documentos relevantes en la lista de resultados, medido por el ranking de la primera respuesta relevante [1].
- **FactScore.** Mide la precisión del texto generado en base a su comparación con documentos de referencia. Un FactScore alto indica que la respuesta generada coincide con los documentos originales, garantizando así que el sistema proporciona información confiable y precisa [7, 11].
- **BERTScore** Mide la similitud semántica entre dos fragmentos de texto utilizando embeddings contextuales tipo BERT, proporcionando valores para precision, recall, y F1. Un BERTScore más alto indica que las respuestas generadas son semánticamente similares a las respuestas de referencia, lo que refleja su calidad y adecuación contextual [13, 9, 5].

4. Tecnologías

A continuación, se listan una serie de librerías que permiten la interacción con LLMs y que pueden ser utilizadas durante el desarrollo de la práctica. Nótese que estas no son las únicas; los estudiantes pueden utilizar otras si así lo desean.

4.1. DSPy

DSPy[16] es una librería que abstrae la interacción con LLMs mediante clases de Python. Además, facilita la optimización de prompts y el ajuste de pesos de los LLMs de manera sistemática. Proporciona soporte para modelos de HuggingFace, OpenAI, así como modelos alojados localmente.

Existen tres conceptos básicos para el funcionamiento de DSPy: *signatures*, *modules* y *optimizers*:

- **Signature.** Define el comportamiento de la tarea que queremos que nuestro LLM desempeñe mediante la especificación de las entradas y salidas del programa. Equivale a definir un "prompt", pero en lugar de pasarle un texto directamente al LLM, DSPy se encarga de construir y optimizar dicha definición. Por ejemplo, la siguiente signature define una tarea de para la extracción de hechos de un texto:

```python
class GenerateFacts(dspy.Signature):
    """
    Extract self-contained and fully contextualized facts from the given passage.
    """
    passage = dspy.InputField(
        desc="The passage may contain one or several claims")
    facts = dspy.OutputField(
        desc="List of self-contained and fully contextualized claims in the form
        ↪  'subject + verb + object' without using pronouns or vague references",
        ↪  prefix="Facts:")
```

Figura 3: *DSPy Signature* para la extracción de hechos de un texto.

- **Module.** Implementa un flujo de trabajo completo, incluyendo la lógica para conectar varias operaciones (que pueden ser definidas por una o más Signatures).

```python
class FactsGeneratorModule(dspy.Module):
    def __init__(self):
        super().__init__()
        self.generate_facts = dspy.Predict(GenerateFacts)

    def process_facts(self, facts):
        process_facts = # Logic to process facts
        return process_facts

    def forward(self, passage):
        facts = self.generate_facts(passage=passage).facts
        processed_facts = self.process_facts(facts)
        return dspy.Prediction(facts=processed_facts)
```

Figura 4: *DSPy Module* para la extracción de hechos de un texto.

DSPy ofrece varios módulos que facilitan la creación de pipelines[17], siendo los más importantes:

[16] https://dspy-docs.vercel.app/
[17] https://dspy-docs.vercel.app/api/category/modules

- **ChainOfThought**: Induce al LLM a "pensar" (seguir un razonamiento encadenado) antes de llegar a la respuesta.
- **Retrieve**: Recupera información relevante de una base de datos en respuesta a una consulta.
- **Predict**: Genera una predicción en base a la entrada proporcionada.

- **Optimizer.** Los optimizadores en DSPy son algoritmos que permiten ajustar automáticamente los prompts y los pesos del LLM, maximizando la calidad de los resultados de acuerdo a métricas específicas. Algunos de los optimizadores más relevantes incluyen:

  - **BootstrapFewShot**: Genera ejemplos automáticamente para implementar un aprendizaje "few-shot" y mejora la calidad del programa a través de la optimización de estos ejemplos, esto es, encuentra los ejemplos que maximizan la métrica sobre la cual estamos llevando a cabo la optimización.
  - **BootstrapFewShotWithRandomSearch**: Expande el enfoque de BootstrapFewShot aplicando una búsqueda aleatoria sobre los ejemplos generados, seleccionando las mejores configuraciones de manera iterativa.
  - **MIPRO y COPRO**: Permiten optimizar las instrucciones, y en el caso de MIPROv2, también los ejemplos few-shot.
  - **BootstrapFinetune**: Permite destilar un programa basado en prompts eb actualizaciones de pesos. La salida es un program DSPy con los mismos pesos, pero dónde cada paso se lleva a cabo a través de un modelo fine-tuneado en lugar de a través de prompts al LLM.

  Por ejemplo, los bloques de código 6-7 permiten llevar a cabo la optimización de `FactsGeneratorModule`. A través de la clase `FactsDataset` transformamos nuestros datos de entrenamiento en la entrada de datos de DSPY (`dspy.Example`). La función `optimize_module` permite optimizar el módulo en función de la métrica definida en `combined_score`. Los parámetros `mbd=4`, `mld=16`, `ncp=2` y `mr=2` controlan la optimización de la siguiente manera:

  - `mbd` (*max_bootstrapped_demos*): Máximo número de ejemplos iniciales generados automáticamente (bootstrapped) para alimentar el modelo.
  - `mld` (*max_labeled_demos*): Máximo Número de ejemplos etiquetados usados para ajustar el modelo.
  - `ncp` (*num_candidate_programs*): Cantidad de programas candidatos que se probarán durante la optimización.
  - `mr` (*max_rounds*): Máximo número de rondas de optimización.

```python
def optimize_module(self, data_path, mbd=4, mld=16, ncp=2, mr=2, dev_size=0.25):
    dataset = FactsDataset(data_fpath=data_path, dev_size=dev_size)

    trainset = dataset._train
    devset = dataset._dev
    testset = dataset._test

    config = dict(max_bootstrapped_demos=mbd, max_labeled_demos=mld,
    ↪  num_candidate_programs=ncp, max_rounds=mr)
    teleprompter = BootstrapFewShotWithRandomSearch(metric=self.combined_score,
    ↪  **config)

    compiled_pred = teleprompter.compile(FactsGenerator(), trainset=trainset,
    ↪  valset=devset)
    return compiled_pred
```

Figura 5: Función para la optimización del módulo `FactsGeneratorModule`.

```python
class FactsDataset(Dataset):
    def __init__(
        self,
        data_fpath: str,
        dev_size: Optional[float] = 0.2,
        test_size: Optional[float] = 0.2,
        text_key: str = "passage",
        seed: Optional[int] = 11235,
        *args,
        **kwargs
    ) -> None:
        super().__init__(*args, **kwargs)

        self._train, self._dev, self._test = [], [], []

        train_data = pd.read_csv(pathlib.Path(data_fpath))

        train_data, temp_data = train_test_split(
            train_data, test_size=dev_size + test_size, random_state=seed)
        dev_data, test_data = train_test_split(
            temp_data, test_size=test_size / (dev_size + test_size),
            ↪ random_state=seed)

        self._train = [dspy.Example({**row}).with_inputs(text_key) for row in
        ↪ self._convert_to_json(train_data)]
        self._dev = [dspy.Example({**row}).with_inputs(text_key) for row in
        ↪ self._convert_to_json(dev_data)]
        self._test = [dspy.Example({**row}).with_inputs(text_key) for row in
        ↪ self._convert_to_json(test_data)]
```

Figura 6: Clase para la conversión en ejemplos compatibles con DSPy.

```python
def combined_score(example, pred, trace=None):
    def sbert_similarity_score(example, pred, trace=None):
        try:
            scores = []
            predicted_lst = pred["facts"]
            try:
                gt_lst = ast.literal_eval(example.facts)
            except Exception as e:
                print("Error in parsing ground truth facts: ", e)
                gt_lst = example.facts.split(".")

            min_facts = min(len(predicted_lst), len(gt_lst))

            # Generate embeddings for predicted and ground truth facts
            predicted_embeddings = tr_model.encode(predicted_lst[:min_facts])
            gt_embeddings = tr_model.encode(gt_lst[:min_facts])

            # Calculate cosine similarity for each pair of embeddings
            for pred_emb, gt_emb in zip(predicted_embeddings, gt_embeddings):
                similarity = 1 - cosine(pred_emb, gt_emb)
                scores.append(similarity)

            return np.mean(scores) # Return the average similarity score

        except Exception as e:
            return 0.0

    return sbert_similarity_score(example, pred, trace)
```

Figura 7: Métrica para la optimzación del módulo `FactsGeneratorModule`.

4.2. ollama

`ollama`[18] permite gestionar e implementar modelos de lenguaje grande (LLM) de manera local. Si bien `ollama` facilita el despliegue de LLMs localmente, es importante tener en cuenta que se requieren ciertos recursos mínimos para ejecutar estos modelos: al menos 8 GB de RAM para los modelos de 7B, 16 GB para los modelos de 13B y 32 GB para los modelos de 33B.

Una vez que tenemos un modelo desplegado, la interacción con los modelos de `ollama` puede realizarse tanto a través de la API como mediante la librería de Python[19]. En la interacción con `ollama`, se pueden utilizar tanto las funciones `ollama.chat` y `ollama.generate` en Python, como las rutas de la API `/api/chat` y `/api/generate`, que aceptan diferentes parámetros. La principal diferencia entre ellas es que `ollama.chat` permite mantener el contexto de las interacciones previas (ideal para conversaciones con historia), mientras que `ollama.generate` no lo hace. Los bloques de código 8 y 9 muestran cómo interactuar con ambas a través de código Python. Además, en la sección 5 se muestra cómo podéis hacer llamadas a los modelos `llama3.1` y `llama3.2` que tenemos desplegados en los servidores del departamento. En cualquier caso, podéis encontrar más detalles en la documentación[20].

Los siguientes enlaces os pueden resultar de interés para la construcción de sistemas RAG con `ollama`:

- Ollama Embedding models
- RAG implementation from scratch

```python
import requests
import json

def generate_response(model, system, prompt,
↪  url="http://localhost:11434/api/generate"):
    data = {
        "model": model,
        "system": system,
        "prompt": prompt,
        "stream": False,
    }

    headers = {"Content-Type": "application/json" }

    try:
        response = requests.post(url, headers=headers, data=json.dumps(data))
        if response.status_code == 200:
            return response.json()
        else:
            return f"Error: {response.status_code}, {response.text}"

    except Exception as e:
        return f"An error occurred: {str(e)}"

generate_response("llama3.2", "You are a helpful AI Assistant", "What are you?")
```

Figura 8: Ejemplo de función para llamar a la API de `ollama`.

```python
import ollama

response = ollama.chat(
    model=llm_model,
    messages=[
    {
        'role': 'user',
        'content': 'Describe me what topic modeling is as if I were a kid',
    },
])
print(response['message']['content'])
```

Figura 9: Ejemplo de función para llamar a la API de `ollama` a través de la librería de Python.

[18] https://ollama.com/library
[19] https://pypi.org/project/ollama-python/
[20] https://github.com/ollama/ollama/tree/main/docs

4.3. LlamaIndex

`LlamaIndex`[21] (anteriormente `GPT-Index`) facilita el desarrollo de aplicaciones con LLMs. Se destaca por su capacidad de cargar datos desde múltiples formatos (e.g., PDFs, bases de datos SQL), indexar en diversas bases de datos vectoriales, realizar consultas RAG y ofrecer módulos para evaluación y ajuste de rendimiento. Además, permite la conexión con modelos de lenguaje tanto gratuitos como comerciales. A continuación, se incluyen enlaces con instrucciones detalladas para instanciar y generar texto con modelos gratuitos, así como desarrollar un sistema RAG:

- Hugging Face LLMs
- HuggingFaceInferenceAPI
- Building RAG from Scratch (Open-source only!)

4.4. LangChain

`LangChain`[22] es una librería que, al igual que `LlamaIndex`, abstrae la interacción con LLMs mediante varios componentes. Ambas ofrecen funcionalidades similares, pero se diferencian en enfoque y optimización. `LlamaIndex` se centra en la eficiencia y simplicidad, particularmente en tareas de búsqueda y recuperación de información, mientras que `LangChain` es más adecuada para aplicaciones complejas que involucren múltiples flujos de trabajo con LLMs, como la generación encadenada de texto, la integración con APIs externas y el manejo avanzado de contexto[23].

A continuación, algunos enlaces útiles si prefierís utilizar `LangChain` para el desarrollo de la práctica:

- Implementing RAG with Langchain and Hugging Face
- LangChain Document QA with ollama
- RAG Pipeline with Ollama and ChromaDB
- Building an Open-Source RAG Application Using Ollama, TextEmbed, and LangChain

5. Entorno de despliegue en la UC3M

Los alumnos tendrán acceso a los LLMs desplegados en la UC3M. Podemos interactuar con estos modelos mediante un comando `curl`, tal y como se muestra en el bloque de código 10, o mediante las tecnologías mencionadas en la Sección 4. Nótese que para cualquiera de las teconologías anteriores, los alumnos deberán configurar el LLM host de forma que se haga una query a la URL del host `yiyuan.tsc.uc3m.es` con la API-KEY indicada en el ejemplo del `curl` (10).

```
curl -k -H "X-API-KEY:sk-af55e7023913527f0d96c038eec2ef2d"
↪  https://yiyuan.tsc.uc3m.es/api/generate -d '{

"model": "llama3.1:8b",

"prompt": "Why is the sky blue?",

"stream": false

}'
```

Figura 10: Ejemplo de llamada al modelo `llama3.1:8b` mediante curl.

Los LLMs actualmente desplegados son:

- `llama3.1:8b`
- `qwen3:8b`
- `gemma3:4b`

No obstante, los alumnos pueden solicitar el despliegue de otros modelos si es necesario para el desarrollo de su práctica.

[21] https://github.com/run-llama/llama_index
[22] https://github.com/langchain-ai/langchain
[23] https://www.gettingstarted.ai/langchain-vs-llamaindex-difference-and-which-one-to-choose/

Referencias

[1] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 international conference on web search and data mining*, pages 87–94, 2008.

[2] Miguel de la Vega. Explorando los hiperparámetros temperature y top p en los modelos de lenguaje de openai. https://medium.com/@1511425435311/explorando-los-hiperparámetros-temperature-y-top-p-en-los-modelos-de-lenguaje-de-openai-7ee74d8912c0, 2023.

[3] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[4] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020.

[5] Hwang Youn Kim, Ghazanfar Ali, and Jae-In Hwang. Enhancing doctor-patient communication in surgical explanations: Designing effective facial expressions and gestures for animated physician characters. *Computer Animation and Virtual Worlds*, 35(3):e2236, 2024.

[6] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *CoRR*, abs/2005.11401, 2020.

[7] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *EMNLP*, 2023.

[8] Anaya Multimedia. *Inteligencia artificial generativa con modelos de ChatGPT y OpenAI.* Anaya Multimedia, 2023. Títulos Especiales.

[9] Mercy Ranjit, Gopinath Ganapathy, Ranjit Manuel, and Tanuja Ganu. Retrieval augmented chest x-ray report generation using openai gpt models. In Kaivalya Deshpande, Madalina Fiterau, Shalmali Joshi, Zachary Lipton, Rajesh Ranganath, Iñigo Urteaga, and Serene Yeung, editors, *Proceedings of the 8th Machine Learning for Healthcare Conference*, volume 219 of *Proceedings of Machine Learning Research*, pages 650–666. PMLR, 11–12 Aug 2023.

[10] Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, et al. The prompt report: A systematic survey of prompting techniques. *arXiv preprint arXiv:2406.06608*, 2024.

[11] Sheikh Shafayat, Eunsu Kim, Juhyun Oh, and Alice Oh. Multi-fact: Assessing multilingual llms' multi-regional knowledge using factscore, 2024.

[12] Syntonize. Sistema rag: Qué es y conceptos claves. https://www.linkedin.com/pulse/sistema-rag-qué-es-y-conceptos-claves-syntonize-kxqge/, 2023.

[13] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with BERT. *CoRR*, abs/1904.09675, 2019.