Informe del Trabajo de Módulo LLM

1. Introducción

Este informe describe el diseño, desarrollo y evaluación de un asistente turístico basado en Large Language Models (LLM), desarrollado como proyecto final de la asignatura **Large Language Models** del Máster en Inteligencia Artificial, Cloud Computing y DevOps de PontIA.

---

2. Objetivo del proyecto

El objetivo general del proyecto es desarrollar un prototipo funcional de asistente turístico capaz de:

- Responder preguntas sobre una guía turística utilizando recuperación semántica.
- Mantener el contexto de la conversación a lo largo de múltiples turnos.
- Invocar funciones externas para enriquecer las respuestas, en particular una función de predicción meteorológica.

Todo el sistema se implementa y demuestra desde un único notebook, garantizando su reproducibilidad.

---

3. Diseño de la solución

3.1 Arquitectura general

La solución se estructura en las siguientes fases:

1. Carga y preprocesamiento de la guía turística.
2. Segmentación del contenido (*chunking*) y generación de embeddings.
3. Almacenamiento de los embeddings en un vector store.
4. Recuperación de contexto relevante ante cada consulta del usuario.
5. Generación de respuestas mediante un LLM, incorporando el contexto recuperado.
6. Gestión del historial conversacional.
7. Ejecución de llamadas a funciones externas cuando el modelo lo solicita.

3.2 Modelo LLM Utilizado

Se ha decidido utilizar OpenAI como proveedor de modelos de *Large Language Models (LLM)* debido a su madurez, estabilidad y facilidad de integración mediante API.

Durante la fase de experimentación se realizaron pruebas con los modelos GPT-5 mini y GPT-5 nano, ambos orientados a ofrecer un equilibrio entre calidad de respuesta y eficiencia en costes. Tras comparar los resultados obtenidos, se observó que el modelo GPT-5 nano proporciona respuestas de calidad similar a las del modelo *mini* para el caso de uso, cumpliendo adecuadamente los requisitos del asistente turístico.

Dado que el rendimiento era comparable, se optó por utilizar GPT-5 nano como modelo principal,

A continuación se detallan los costes aproximados del modelo GPT-5 nano por millón de tokens:

- Input: $0.05
- Cached Input: $0.01
- Output: $0.40

---

4. Implementación de RAG

4.1 Estrategia de *chunking*

La guía turística se divide en fragmentos de tamaño controlado para mejorar la calidad de la recuperación de información y la relevancia de las respuestas generadas.

Se ha definido un tamaño máximo de **650 tokens por fragmento**, ya que el documento no contiene secciones excesivamente extensas, pero sí requiere que cada *chunk* incluya recomendaciones completas. Además, se ha aplicado un **solapamiento de 150 tokens** entre fragmentos para evitar la pérdida de contexto entre secciones consecutivas.

Los parámetros utilizados son:

- **max_chunk_size:** 650 tokens
- **chunk_overlap_tokens:** 150 tokens

4.2 Generación de embeddings y vector store

Cada fragmento de la guía turística se transforma en un **embedding vectorial** y se almacena utilizando el **vector store proporcionado por OpenAI**. Este servicio gestiona de

forma automática el almacenamiento y la búsqueda semántica de los fragmentos, simplificando la implementación.

El uso del vector store de OpenAI permite además que el sistema **seleccione internamente el modelo de embeddings más adecuado**, sin necesidad de configurarlo manualmente.

4.3 Recuperación y citación de fuentes

Ante una consulta del usuario, el sistema recupera los fragmentos más relevantes y los incorpora al *prompt* del modelo. Las respuestas incluyen referencias explícitas a las fuentes utilizadas.

---

5. Diálogo multiturno

El asistente tiene la capacidad de mantener un historial de la conversación que permite generar respuestas coherentes a lo largo de múltiples turnos, reutilizando el contexto proporcionado previamente por el usuario. De este modo, el sistema es capaz de responder a preguntas de seguimiento sin necesidad de que el usuario repita información.

Para validar este comportamiento, se realizaron pruebas de conversación multiturno utilizando **trazas (*trace*)**, que permiten agrupar varios turnos dentro de un mismo flujo conversacional.

---

6. Llamadas a funciones externas

6.1 Definición de la función get_weather

Se define una función externa `get_weather()` que permite al LLM identificar correctamente cuándo debe ser invocada. La función devuelve información meteorológica simulada o real para una fecha concreta.

Para esta versión se ha descartado el uso de parámetro fecha (limitaciones de la API que fecha te puedes traer) o localización (se usa de forma estática la localización de Tenerife).

Para futuras versiones con más tiempo se puede mejorar esta parte usando parámetros y tipandolos con pydantic.

---

7. Mejoras futuras

Como trabajo futuro se proponen mejoras como:

- Integración de múltiples fuentes documentales.
- Uso de técnicas avanzadas de memoria conversacional.
- Despliegue del asistente en una interfaz web.
- Mejorar observabilidad del agente
- Mejorar la función de get_weather() incorporando parámetros como la fecha o zona de la isla para que mejore la precisión del tiempo.

---

11. Conclusiones

El resultado es un **prototipo funcional y reproducible**, implementado en un único notebook. Este proyecto demuestra que, utilizando las herramientas que ofrece OpenAI, se puede crear de forma rápida una **prueba de concepto** de un asistente conversacional bastante competente.

Durante el desarrollo se ha comprobado que, con relativamente poco código y una configuración sencilla, es posible construir un asistente que combine recuperación de información, diálogo multiturno y uso de herramientas externas.

Me hubiera gustado disponer de más tiempo para profundizar en los distintas opciones de extras disponibles y poder pensar con más calma qué otros extras o mejoras se podrían añadir al sistema,