Guías museísticos personalizados mediante avatares con Inteligencia Artificial Generativa

Trabajo de Fin de Grado
Curso 2024–2025

Autor
Rafael Moreno Portilla

Director
Rubén Fuentes-Fernández

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Calificación: 10

Guías museísticos personalizados mediante avatares con Inteligencia Artificial Generativa
Personalized Museum Guides through Avatars with Generative Artificial Intelligence

Trabajo de Fin de Grado en Ingeniería Informática

Autor
Rafael Moreno Portilla

Director
Rubén Fuentes-Fernández

Convocatoria: *Septiembre 2025*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Calificación: 10

09 de septiembre de 2025

Dedicatoria

A mis padres, Rafael y Soledad, por haberme dado todo en esta vida.

Agradecimientos

A mi tutor, Rubén, por tener tanta paciencia y haber dedicado tiempo durante verano para ayudarme a seguir con el trabajo. Y, por supuesto, a mis compañeros del Doble Grado, por estar siempre ahí apoyando con ese corazón tan grande que tienen.

Resumen

Guías museísticos personalizados mediante avatares con Inteligencia Artificial Generativa

Este Trabajo de Fin de Grado consiste en el desarrollo de un *chatbot* estilizado de manera que se comporte como un personaje histórico del siglo XVI en Sudamérica: don Francisco de Arobe. Para ello se utiliza un sistema de Inteligencia Artificial en el que interactúan múltiples modelos, siendo el más importante de ellos un Modelo Grande de Lenguaje. Se ha llevado a cabo en respuesta a la iniciativa del Museo de América. La aplicación se desarrollará para adecuarse al contexto en el que se va a usar: una sala de un museo. Con este objetivo, se prestará especial atención en evitar comportamientos que puedan provocar disonancia con el público.

Palabras clave

chatbot, Inteligencia Artificial, Modelo Grande de Lenguaje, modelos, personaje

Abstract

Personalized Museum Guides through Avatars with Generative Artificial Intelligence

This Bachelor's Thesis consists of the development of a stylized *chatbot* designed to behave as a historical 16th-century character in South America: don Francisco de Arobe. For this purpose, an Artificial Intelligence system is used in which multiple models interact, the most important of them being the Large Language Model. It has been carried out in response to an initiative from the Museum of America. The application will be developed to suit the context in which it will be used: a museum hall. With this objective, special attention will be paid to avoiding behaviors that could create dissonance with the public.

Keywords

chatbot, Artificial Intelligence, Large Language Model, models, character

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

A lo largo de la Historia, el ser humano siempre se ha esforzado en progresar como civilización. Durante los últimos años, este progreso se ha centrado en el desarrollo de las nuevas tecnologías (dispositivos móviles, Internet, etc). Uno de los frutos más recientes es la aparición de la Inteligencia Artificial (IA).

Ésta es una rama de la informática que tiene como objetivo desarrollar sistemas para realizar tareas que, hasta hace poco, requerían de la inteligencia humana. Dichas tareas incluyen el razonamiento, el aprendizaje o la toma de decisiones.

A diferencia de los programas tradicionales, que siguen de forma rígida un conjunto de instrucciones, los sistemas de IA pueden adaptarse a nuevas situaciones. Ésto se debe a que extraen patrones de datos para luego poder enfrentarse a entradas completamente nuevas, lo cual genera sistemas más flexibles.

En este contexto, los museos y centros culturales afrontan un desafío creciente: mantener el interés de un público que está cada vez más habituado a experiencias inmediatas, interactivas y digitales. Estos espacios desempeñan un papel fundamental en la preservación y difusión de la historia y el patrimonio cultural. Sin embargo, su metodología algo tradicional de la transmisión de conocimiento suele resultar menos atractivas para las generaciones más jóvenes, acostumbradas al uso continuo de Internet, redes sociales y tecnologías interactivas.

Para dar respuesta a esta problemática, muchos museos han comenzado a integrar herramientas tecnológicas que les permitan ofrecer experiencias más dinámicas e inmersivas. Estas iniciativas buscan tanto complementar la exposición física como mejorar la accesibilidad y la interacción del visitante con el contenido cultural. Entre distintas iniciativas, podemos encontrar desde algunas tan sencillas como el diseño de un mapa del edificio hasta la más reciente realidad virtual.

La IA se presenta como una oportunidad para transformar la forma en que los usuarios se relacionan con la información dentro de un museo. Mediante técnicas de procesamiento de lenguaje, reconocimiento de imágenes o generación de contenidos

personalizados, la IA puede adaptarse a las necesidades de cada visitante y proporcionar experiencias más cercanas, interactivas y atractivas.

Siguiendo la línea marcada por los museos, este trabajo persigue el desarrollo acabado de una aplicación que utilice IA para aumentar el interés del público hacia estos centros a través de una actividad más cercana.

1.1. Propuesta del Museo América

A continuación detallaremos nuestro proyecto, que surge de una propuesta del Museo de América. Revisaremos la propuesta inicial y su contexto.

Para facilitar las explicaciones sobre las obras del museo y con vistas a proporcionar un servicio que capte a gente de edades más tempranas, el Museo de América de Madrid propuso desarrollar un *chatbot* que representase a un personaje histórico. Los visitantes podrían conversar con él, lo cual serviría como atracción turística para el público y como guía automático capaz de resolver dudas sobre el propio personaje.

La propuesta se hizo en relación al cuadro "Los mulatos de Esmeraldas". En concreto, el proyecto consistiría en un *chatbot* que representase a don Francisco de Arobe, el personaje principal del cuadro. Antes que nada, conviene que introduzcamos a don Francisco de Arobe y su contexto, y así poder entender mejor el proyecto y sus dificultades.

Durante la Época de la Conquista y la Colonización (siglos XVI - XVIII), muchos españoles viajaron a las indias de América para continuar el avance de la Monarquía Católica iniciado bajo el reinado de Isabel y Fernando. Uno de los conquistadores más renombrados fue Francisco Pizarro. Éste desembarcó en 1532 en la costa norte de Ecuador para dirigirse al sur, hacia el corazón de Perú. Con la caída del imperio inca, esta zona fronteriza empezó a despoblarse debido a enfermedades y traslados forzosos. Además, muchos barcos llenos de desertores y esclavos fugados naufragaban en esta costa. Los náufragos que sobrevivían se refugiaban en la selva y atemorizaban a las poblaciones cercanas. Por todos estos motivos, esta zona, bautizada con el nombre de Esmeraldas, se convirtió en un lugar casi maldito.

En uno de estos barcos naufragados viajaba Andrés Mangache, un africano esclavizado. Por suerte, habiendo conseguido escapar junto con su amada, se rodeó de seguidores y plantó cara a los indígenas autóctonos. Aunaron fuerzas con algunos naturales, como los de la "tierra de Arobe", del cual sus descendientes seguramente tomaron el apellido "Arobe".

Los Mangache engendraron a dos hijos, entre los que se encuentra nuestro don Francisco de Arobe. Éste último acabó ocupando el puesto de cacique tras la muerte de su padre y de su hermano mayor, Juan.

Si bien su cacicazgo no fue totalmente favorable a las autoridades de la zona, los Arobe se ganaron la fama de ser más colaboradores y cercanos que otros pueblos y caciques, llegando incluso a convertirse al cristianismo.

Como consecuencia de esta relación y de la ayuda que proporcionaron a muchos otros náufragos, se logró arreglar un acuerdo entre los Arobe y el juez real Juan Barrio de Sepúlveda. En 1599, don Francisco viajó a Quito con dos de sus hijos para prometer su lealtad a la Corona, tras lo cual se le otorgó el título de gobernador del territorio esmeraldeño.

Como consecuencia de este evento, se pintó el cuadro "Los mulatos de Esmeraldas" y se envió a Felipe IV, permaneciendo en la colección real hasta finales del siglo XIX. Posteriormente se trasladó al Museo de América, donde se encuentra hoy en día. En el cuadro (ver Fig. 1.1) se representa a don Francisco, en el centro, con sus dos hijos, uno a cada lado.

Figura 1.1: Los mulatos de Esmeraldas (1599), Andrés Sánchez Gallque

1.2. Planteamiento general y requisitos del sistema

Con este proyecto en mente, conviene que especifiquemos un primer planteamiento de nuestra aplicación y sus requisitos.

El *chatbot* se implementará como un sistema de generación aumentada de recuperación (*Retrieval Augmented Generation*, RAG). Esto permitirá dotarle de capacidades de comprensión y generación de lenguaje natural, al tiempo que se limitan sus respuestas en buena medida a una base documental restringida. El núcleo de

este sistema RAG es un modelo grande de lenguaje (*Large Language Model*, LLM) con un LLM, que es el que soporta las capacidades de procesamiento relacionadas con el lenguaje natural. Éste se enlazará con otros modelos de IA para lograr la personalización del *chatbot* y que así se exprese como don Francisco de Arobe. Dado que don Francisco vivió durante comienzos de la Edad Moderna, es importante que dicho *chatbot* no sólo intente hablar como una persona de aquella época, sino que también únicamente posea información relativa a su historia.

La primera tarea es difícil, ya que de aquella época no se poseen más que escritos cultos como textos legislativos o literarios, por lo que no se conoce exactamente cuál sería la forma de hablar de don Francisco. Sin embargo, se tratará de dar un aire de solemnidad y sencillez a sus palabras para ambientarlo en el contexto de un esclavo africano que se convierte en gobernador.

Para la segunda tarea, la información sobre la época la aportaremos con otros modelos de IA, como ya hemos mencionado. La restricción del conocimiento del *chatbot* se realizará a través de la elección de un LLM que sufra pocas alucinaciones y que desempeñe bien la funcionalidad de responder a una pregunta únicamente con la información aportada.

Es especialmente importante que el *chatbot* no sufra alucinaciones, ya que éstas podrían llevar a confusión a los usuarios o incluso a la exposición de hechos falsos o el uso de palabras inadecuadas. De todo esto nos cercioraremos también mediante la elección del LLM.

Para lanzar la aplicación es imprescindible que dispongamos al menos de un dispositivo de salida (posiblemente algún tipo de pantalla), y otro de entrada (como un teclado). La interfaz se desarrollará sobre estos dispositivos y deberá ser sencilla y responder a las necesidades de una gran mayoría de los usuarios.

Este proyecto fue comenzado en un TFG anterior[1]. Se partirá de dicho sistema para abordar los aspectos abiertos y concluir una aplicación lista para su despliegue en el Museo de América. Del proyecto original se ha mantenido el concepto del proyecto y la GUI, pero se ha reestructurado y reimplementado el resto de la aplicación para adaptarla a la nueva arquitectura del sistema, los avances en la infraestructura y para resolver los problemas de funcionamiento, en especial el rendimiento.

1.3. Objetivo principal y subobjetivos

El objetivo principal consiste en diseñar un *chatbot* desarrollado para adoptar el rol de un personaje histórico, en nuestro caso, don Francisco de Arobe. Es decir, pretendemos dar contexto e instrucciones concretas a un *chatbot* para que este actúe como un personaje, de manera que se simule una conversación real con él.

[1] https://github.com/mgm48/tfg

Esta aplicación se instalará en una sala del museo, al lado de un cuadro del personaje que quiere representar el *chatbot*. Por ello, estará expuesta a la interacción con cualquier visitante que acceda al museo y pase por la sala. Este hecho pone de relieve la importancia de que se eviten alucinaciones y, especialmente, respuestas no adecuadas que puedan incomodar a los visitantes o resultar inapropiados en este contexto. Por ésta misma razón, también es relevante el tono en el que se den las respuestas. Este tono debe ser formal y cortés.

Además, dado que trata de simular a un personaje histórico, debemos cerciorarnos de que las respuestas son acordes con el personaje. Esto implica no sólo que posea conocimiento detallado del personaje y su contexto, sino que sus respuestas se limiten a este conocimiento. Es decir, que no sea capaz de responder sobre información que el personaje no conoce. Para ello, dispondremos de documentación sobre el personaje (limitada, dada sus características), con la que se alimentarán las respuestas del *chatbot*.

Los puntos objetivo clave del proyecto serán:

- Integrar un modelo de Inteligencia Artificial Generativa (IAG) en la aplicación como base del *chatbot*. Se utilizará un LLM. Éste estará disponible localmente para facilitar su ajuste y optimizar su rendimiento.
- Implementar una recuperación de contexto para ayudar al LLM local a generar respuestas según el conocimiento del personaje histórico. Para ello se utilizará una base de datos que almacenará la información del personaje y un sistema de recuperación de contexto. De esta forma se lograrán respuestas acordes con el personaje del LLM. Esto se denomina RAG.
- Diseñar una Interfaz Gráfica de Usuario (*Graphic User Interface*, GUI) que permita tanto interactuar con el *chatbot* como mostrar el historial del *chat* con la conversación del momento.
- Desarrollar las funcionalidades necesarias para lograr un *chatbot* con el que se pueda interactuar también verbalmente y que pueda mostrar los resultados mediante voz (*speech-to-speech chatbot*). Para ello utilizaremos un modelo de IA que se encargue de la síntesis de voz y otro para la producción verbal de las respuestas.
- Proveer al usuario técnico una manera de añadir o quitar documentos de la base de conocimiento. Esta funcionalidad quedará oculta al usuario general.

1.4. Plan de trabajo

Las fases que seguiremos para lograr los objetivos anteriores son las siguientes:

1. **Estado de la cuestión.** Enero - febrero de 2025.
   Antes que nada estudiaremos con detalle el estado de la aplicación actual. Comprobaremos sus puntos débiles para solucionarlos y conseguir una aplicación completa y lista para usar.

   Analizados estos problemas procederemos al estudio de distintos modelos de IA para todas las funcionalidades anteriores, buscando la elección más acertada de cada uno de ellos.

2. **Diseño de la arquitectura y elección de la infraestructura.** Febrero de 2025.
   Definiremos el flujo de funcionamiento de la aplicación para integrar correctamente todas las partes.

   Posteriormente, escogeremos la plataforma de *software* y *hardware* que mejor se adapte al uso y los modelos anteriores.

3. **Desarrollo de la aplicación.** Febrero - mayo de 2025.
   Se implementarán las funcionalidades y se decidirá la mejor configuración de éstas para lograr una aplicación acabada y completa.

4. **Prueba y corrección.** Mayo - agosto de 2025
   Se estudiará la eficiencia y precisión de la aplicación y se realizarán los cambios necesarios para mejorar estos aspectos. Esta fase es fundamental para comprobar que la aplicación tiene un buen rendimiento suponiendo su uso en unas instalaciones adecuadas.

5. **Desarrollo de la guía de instalación y de uso.** Septiembre de 2025.
   Para facilitar su uso, se especificará de manera concreta cómo se debe instalar la aplicación y se proveerá una guía de uso.

El resto del documento se organiza como sigue:

- Capítulo 2. Abordaremos el estado de la cuestión describiendo la situación actual en los museos con las nuevas tecnologías. También introduciremos los distintos modelos que nos van a ser necesarios y algunas otras herramientas de infraestructura.

- Capítulo 3. Describiremos detalladamente la arquitectura de la aplicación.

- Capítulo 4. Realizaremos un estudio sobre los modelos actuales que mejor sirvan a nuestro propósito y propondremos uno para cada tipo de modelo.

- Capítulo 5. Detallaremos cómo hemos hecho para optimizar el uso de modelos con tal de mejorar el rendimiento de nuestra aplicación.

- Capítulo 6. Proporcionaremos una guía para usar la aplicación.

- Capítulo 7. Resumiremos todo lo realizado y especificaremos algunos puntos de trabajo que se pueden realizar en un futuro para continuar este proyecto.

Capítulo 2

Estado de la Cuestión

2.1. Relevancia del uso de tecnologías en los museos

Los museos han sido desde sus inicios intermediarios entre la sociedad actual y la historia del pasado. A pesar del cambio que han sufrido a lo largo de los siglos, su misión esencial permanece: conservar, investigar y compartir el patrimonio histórico. Hoy en día se enfrentan principalmente a dos nuevos retos.

El primero de ellos es la aparición de las nuevas tecnologías que mencionamos en la introducción. El desarrollo de la tecnología lleva vigente en la humanidad desde que esta existe. Por tanto, más que una aparición, se trata de un progreso abrupto de algunas tecnologías en cuestión de pocos años. Este avance vertiginoso plantea un desafío a las instituciones culturales para seguir resultando atractivas al público.

Al mismo tiempo, ese progreso ofrece nuevas oportunidades. Estas van desde la mejora de instalaciones para el cuidado de las obras y objetos de los museos hasta la invención de nuevas experiencias. En esta línea, muchos centros ya han decidido implementar nuevos servicios. Por ejemplo, **The British Museum**, en Londres, colabora con Google Arts & Culture y Microsoft para digitalizar su colección y ofrecer visitas en realidad virtual (*Virtual Reality*, VR) (British Museum, 2020).

Además, también se han creado nuevos centros independientes con estas mismas miras de experiencias novedosas. En Madrid mismo, **MAD Madrid Artes Digitales** es el único centro permanente dedicado a experiencias inmersivas en la capital española. Desde su fundación en 2022, ha producido múltiples eventos como los dedicados a Tutankamón y Pompeya. Actualmente, está disponible la exposición en VR del Titanic (Taggart, 2025).

En 2019, la pandemia del COVID-19 forzó el cierre de la mayoría de museos en todo el mundo, muchos de los cuales se clausuraron definitivamente. Este hecho también ha supuesto un incentivo para desarrollar experiencias telemáticas usando las Tecnologías de la Información y la Comunicación (TIC) (López López et al., 2025). El **Museo Nacional del Prado** en Madrid propuso iniciativas (Longhi-Heredia

et al., 2021) como #ElPradoContigo para mantener el contacto con su audiencia durante el confinamiento. Ésta consistía en una serie de vídeos cortos narrados por historiadores y otros trabajadores del museo que explicaban algunas de las obras maestras del museo.

El segundo reto es el cambio que se ha producido en el público objetivo que asiste a los museos. Por una parte, la edad media de los visitantes al museo ha aumentado significativamente (Europa Press, 2024). La pérdida paulatina del interés juvenil por la historia preocupa a los centros culturales, que pretenden educar también a las nuevas generaciones. Además, también es notable que con la mejora de los medios de transporte de larga distancia, cada vez son más frecuentes las visitas culturales a otros países, lo que ha supuesto un aumento del público turista (Zárate Martín y García Ferrero, 2017). Dicho público es común que no posean información extensa sobre el contenido de los museos y la historia del país.

2.2. Inteligencia Artificial

Entre los avances más recientes de la tecnología se encuentra la **IA**. Se trata de un sector de la informática que nace de la observación de tareas que requieren inteligencia humana, como **aprender, razonar, resolver problemas o comprender lenguaje**.

Una parte de la IA trata de realizar estas tareas a través de algoritmos y modelos matemáticos para analizar grandes cantidades de datos, detectar patrones y tomar decisiones en base a lo anterior. El rango de su aplicación abarca desde asistentes virtuales hasta diagnósticos médicos o recomendaciones personalizadas.

Dentro esa aproximación a la IA, una de las ramas que más impacto y atención han tenido estos últimos años es la **IAG**. Dicha rama utiliza modelos avanzados como las famosas redes neuronales profundas. Se centra en crear contenido como texto, imágenes o música a partir de datos existentes de manera coherente. Entre los múltiples usos de la IAG, se encuentra su aplicación a la creación de *chatbots*. Los *chatbots* son aplicaciones que pretenden imitar a un participante en una conversación humana, generalmente en un contexto acotado.

Por norma general, la efectividad de un modelo de IA depende de los algoritmos utilizados y los datos con los que se entrena el modelo. Éstos datos suponen un riesgo potencial, ya que pueden estar sesgados y provocar discriminaciones. Además, los propios algoritmos también pueden derivar en problemas como la opacidad en los procesos de toma de decisiones (*black box algorithms*). De la misma manera, los sistemas que hacen uso de IA, a medida que avanza su autonomía, podrían tener comportamientos impredecibles o potencialmente dañinos. En los siguientes capítulos trataremos los problemas potenciales más relevantes para nuestra aplicación.

A continuación vamos a introducir una serie de conceptos sobre IA que nos serán necesarios más adelante.

El aprendizaje automático o ***machine learning*** es una rama de la IA que permite que los ordenadores aprendan a realizar tareas sin haber sido programadas explícitamente para ello. Utilizan conjuntos de datos muy grandes para entrenar modelos que intentan extraer patrones para luego hacer predicciones o tomar decisiones basadas en información nueva. Dentro de él, las **redes neuronales artificiales** (ver Fig. 2.1) constituyen una familia de algoritmos inspirados en el funcionamiento del cerebro humano, en los que la información fluye a través de unidades llamadas neuronas conectadas en capas (Rumelhart et al., 1986).

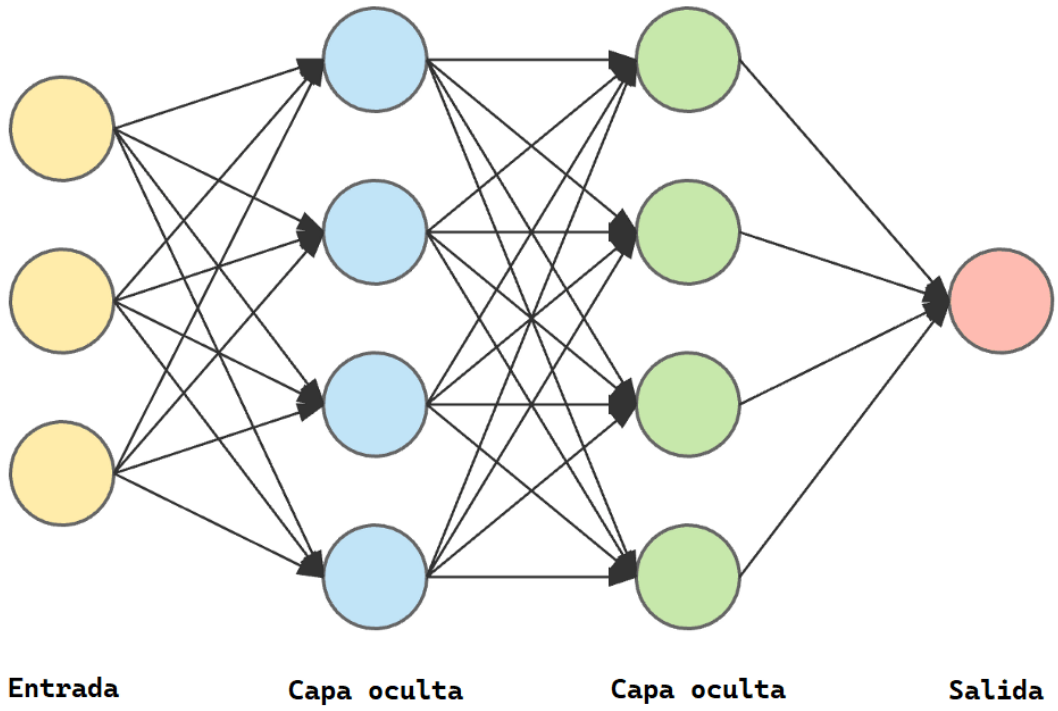

Figura 2.1: Imagen de una red neuronal con dos capas ocultas (OpenWebinars, 2023).

Diferenciamos tres tipos de capas en una red neuronal:

- Capas de entrada. Son las primeras capas y simplemente reciben los datos.
- Capas ocultas. Capas intermedias que procesan los datos y extraen patrones.
- Capas de salida. Son las capas finales y generan una predicción, por ejemplo, una categoría o un número. En el caso de los LLMs predicen un texto de respuesta frente a la entrada proporcionada.

Cuando estas redes cuentan con un gran número de capas intermedias, múltiples tipos de neuronas y otras modificaciones arquitectónicas, se denominan **redes**

neuronales profundas. El estudio y aplicación de este tipo de modelos se llama **aprendizaje profundo** (*deep learning*) que ha demostrado una gran capacidad de extraer patrones complejos de grandes cantidades de datos.

Las redes neuronales profundas tienen decenas o cientos de estas capas ocultas. Durante el entrenamiento, los datos de entrada van atravesando estas capas ocultas hasta llegar a las de salida, donde se genera la predicción. Posteriormente, el algoritmo compara su salida con la salida correcta y calcula el error con respecto a esta. Finalmente usa la técnica de *backpropagation* para ajustar las conexiones entre las distintas capas de la red (Rumelhart et al., 1986). Estos ajustes consisten en variar los pesos de las conexiones, es decir, la importancia de un atributo para relacionar la entrada con la salida esperada. Se denomina *backpropagation* porque los ajustes parten de la capa de salida y van retrocediendo capas hasta llegar a la de entrada.

2.2.1. LLM

El LLM es el modelo de IA principal, ya que se encargará de generar respuestas a partir de una pregunta dada. Los LLM generan lenguaje natural en base a la entrada que reciben. Se dice que son grandes (*large*) por la cantidad de parámetros que poseen (algunos alcanzan los miles de millones), lo cual los hace modelos muy pesados.

Los LLM nacen como evolución de los modelos de lenguaje clásicos, que predecían palabras de manera secuencial (la predicción se basaba únicamente en las palabras que la precedían). Con la llegada de las redes neuronales profundas y la profundización de otros campos del *deep learning* aparecen también los LLM. Los LLM funcionan prediciendo la siguiente palabra de un texto dado un contexto, pero gracias a su entrenamiento masivo en grandes corpus, aprenden patrones lingüísticos complejos y relaciones semánticas. Utilizan mecanismos de atención que consideran toda la entrada para ponderar la importancia de cada palabra en el contexto que se dan. Su salida es un texto generado de manera probabilística, lo que permite creatividad y variabilidad en las respuestas.

Problemas y limitaciones:

- Alto coste computacional. Su entrenamiento y ejecución requieren de recusos masivos.
- Alucinaciones. Debido a que las respuestas se generan de manera probabilística, la información puede ser incorrecta o inventada.
- Sesgos. Por la misma razón, reproducen sesgos presentes en los datos de entrenamiento.

2.2.2. *Embedders*

Queremos almacenar la información en la base de datos para luego recuperar las piezas de información más relevantes a las *queries* que nos hagan. Para ello, necesitamos una forma de medir la relación entre la *query* y la información de nuestra base de datos. Aquí es donde introduciremos nuestros *embedders*.

Un *embedding* es un vector que representa la carga semántica de un trozo de texto. Así, cuando almacenemos pedazos de texto en nuestra base de datos, le añadiremos el *embedding* que lleva asociado para luego poder hacer búsqueda por similitud o cercanía en un espacio vectorial multidimensional. Muchas veces la dimensión del espacio supone una limitación importante para el espacio y procesamiento de los *embeddings*.

Los *embedders* no son más que los modelos de IA que generan un *embedding* a partir de un *chunk* o trozo de texto. La elección del *embedder* es importante, ya que dependen mucho del dominio para el que han sido diseñados. Es decir, un *embedder* entrenado en un dominio general puede no ser óptimo para textos muy especializados, como textos jurídicos o legislativos.

2.2.3. *Rerankers*

Una vez obtenidos los pedazos de información más relevantes, conviene aportarlos a nuestro LLM en orden de relevancia. Para eso utilizaremos un modelo *reranker*. De nuevo, elegiremos un modelo ligero, ya que esta tarea no influye tanto en la precisión de nuestra aplicación.

Básicamente, la manera en la que funciona un *reranker* es que toma una serie de *chunks* y una consulta y genera el *embedding* de cada uno de ellos[1]. Después, asocia a cada *chunk* un valor numérico que representa la relevancia de ese pedazo de información con respecto a la consulta. Finalmente, ordena la lista de *chunks* según ese valor y la devuelve.

2.2.4. Modelos de voz

Los modelos de voz son una herramienta opcional que ha sido añadida para dar más atractivo a la aplicación. Estos modelos permitirán al usuario manejar la aplicación mediante voz y sonido, y no simplemente a través de texto.

Dispondremos de dos modelos. El primero para la generación de texto a partir de un audio que se grabe con la voz del usuario. Esta familia de modelos se conocen

[1]Este *embedding* es distinto del que genera el *embedder* ya que el propósito también es distinto. Cada uno ha sido entrenado con un objetivo diferente, luego los *embedding* que generarán pueden ser distintos en tamaño y valores

como modelos *Speech To Text* (STT). El segundo para la síntesis de audio desde el texto generado como respuesta del LLM. Esta segunda familia de modelos se conocimiento como *Text To Speech* (TTS).

Todo ello en su conjunto, tanto la grabación de voz como la producción de sonido, lograrán que el *chatbot* tenga un componente mucho más humano, alejando la sensación de estar interactuando con una máquina.

2.2.5. Optimización

Los LLM que vamos a utilizar han sido escogidos priorizando que las respuestas que proporcionen sean precisas y no sufran alucinaciones. Sin embargo, estos modelos serán demasiado pesados en general, por lo que tendremos que reducir su tamaño en detrimento de algo de precisión (aunque no demasiada). Para ellos, vamos a utilizar la **cuantización**.

Además, mencionaremos una técnica conocida como el ***fine-tuning*** que ayudará a que los modelos sean más precisos en las tareas concretas para los que los necesitamos. Algunos de los modelos que seleccionaremos en el futuro habrán sido entrenados con dicha técnica.

2.2.5.1. Cuantización

La cuantización es una técnica de compresión usada en modelos de aprendizaje profundo (*deep learning*) que consiste en reducir la precisión numérica de los pesos del modelo.

Cuando decimos que reducimos la precisión numérica de los pesos, nos referimos a que si un valor se expresaba, por ejemplo, con representación float32 (cada valor ocupa 32 bits) ahora pasa a uno de precisión menor, como int8 (que ocupa 8 bits). La representación de números *float* es la de números decimales, mientras que *int* es la de números enteros. Esto no significa que un valor en representación int8 no pueda expresar números decimales. Para esto se aplican técnicas de reescalado, de manera que luego se puede construir un valor decimal que es una aproximación del valor original. Con un ejemplo se entiende mejor. Supongamos que tenemos la variable con precisión float32 siguiente

$$x = 3,14159265.$$

Si le aplicamos un escalado de 10 a 127 y luego redondeamos, obtenemos el número siguiente

$$x2 = x \cdot \frac{127}{10} \approx 39,9.$$

Luego redondeamos ese número a 40 para poder representarlo en precisión int8. Cuando usemos ese valor, podemos aplicar el escalado inverso y obtenemos un valor aproximado del decimal original

$$x3 = 40 \cdot \frac{10}{127} = 3,15.$$

En el nombre del modelo viene indicado el nivel de cuantización. Por ejemplo, llama-7b-Q6_K es el modelo llama-7b pero cuantizado de manera que el número de bits de los pesos se reduce a 6 (de ahí el Q6).

Si la cuantización es buena, la pérdida de precisión en la predicción del modelo es mínima. Existen varios métodos de cuantización, entre los que destacan la *Gradient Post-Training Quantization* (GPTQ) y la *Activation-aware Weight Quantization* (AWQ).

La técnica de GPTQ es posterior al entrenamiento y simplemente reduce la precisión de estos pesos fijándose en su valor actual para que ocupen menos espacio y puedan usarse más rápido en la inferencia más adelante, cuando se use el modelo.

Dicha técnica recorre una por una las capas ocultas de la red y cuantiza en cada paso todos los pesos de una capa. Después de cuantizarlos, evalúa cómo afecta esto a las salidas del modelo y ajusta los pesos de la capa mediante una técnica basada en optimización local que se sirve del gradiente de la función de error. Luego repite el proceso con la siguiente capa oculta hasta llegar procesar todas. Estas observaciones y ajustes se realizan con una pequeña muestra de datos de entrada distintos de los datos que se han utilizado para el entrenamiento con los que se infieren las respuestas. El hecho de que se realice después del entrenamiento tiene como ventaja que no requiere reentrenamiento, un proceso pesado y laborioso.

La precisión de las respuestas de un modelo cuantizado por GPTQ es mayor que la de un modelo que de primeras es más pequeño. Es decir, aunque un modelo cuantizado (cuyo modelo original es pesado) y un modelo pequeño tengan el mismo tamaño (medido en el espacio de memoria que ocupan), el modelo cuantizado suele obtener mejores resultados.

A diferencia de la GPTQ, la AWQ no sólo mira los pesos del modelo una vez entrenado, sino que también considera cómo se activan esos pesos durante la inferencia del entrenamiento, por lo que puede llegar a ser más precisa. A pesar de todo, la técnica GPTQ ofrece resultados suficientemente buenos y, además, tiene un soporte mucho más amplio, por lo que hemos decidido tomar modelos cuantizados mediante esta técnica.

2.2.5.2. Formatos y características

A veces, estos modelos cuantizados los usaremos en formato *GPT-Generated Unified Format* (GGUF). Este formato de archivo es usado para almacenar los mo-

delos de manera modular y extensible. Es decir, que está diseñado de manera que distintas secciones del modelo (como los pesos, la configuración y otros metadatos) se almacenan de manera separada dentro del mismo archivo. Esto permite que se modifique una parte del modelo sin romperlo con la idea de que el modelo pueda crecer fácilmente en el futuro sin necesidad de rehacerlo por completo.

Se trata de un formato que agrupa todo el modelo en un único archivo, lo que lo hace más fácil de compartir y cargar. Los modelos que implementan este formato terminan con la extensión ".gguf". En los nombres de los modelos también viene indicado el número de parámetros que utilizan: 7b (7 billones), 8b (8 billones) y 9b (9 billones).

2.2.5.3. *Fine-tuning*

Algunos de los modelos han sido sometidos a *fine-tuning*. Esta es una técnica usada en *machine learning* que consiste en adaptar un modelo previamente entrenado a una tarea específica o a un nuevo conjunto de datos. Esta técnica también es posterior al entrenamiento, es decir, que una vez inferidas las salidas con los datos de entrenamiento se utiliza otra muestra distinta, generalmente más pequeña, para inferir las nuevas salidas y ajustar los pesos de las capas neuronales.

Concretamente, a nuestros modelos se le ha aplicado el *instruction tunning*, un tipo de *fine-tuning* en el que el modelo se ajusta usando pares de instrucciones y respuestas deseadas, de modo que aprenda a seguir instrucciones humanas de manera más precisa y coherente. En este proceso se utilizan ejemplos concretos que enseñan al modelo a responder preguntas, ser conciso, escribir en primera persona, o realizar otras acciones indicadas por el usuario. Esta técnica no altera la arquitectura del modelo, sino que ajusta sus pesos para que sus salidas se alineen mejor con las expectativas humanas.

Aquellos modelos en cuyo nombre aparece la palabra *instruct* son versiones del modelo con el mismo nombre a los que se les ha aplicado *instruction tuning*[2].

En nuestro caso es especialmente importante ya que utilizaremos varios *prompts* para indicarle al modelo lo que debe hacer exactamente: responder a una pregunta, ser breve, responder en primera persona, etc. Esta característica es crucial.

2.3. Infraestructura

Para desarrollar la aplicación, hemos optado por las siguientes piezas de infraestructura.

[2]En el caso del modelo gemma-2-9b-it-Q6_K.gguf, las letras "it" significan *instruction tuned*.

El lenguaje de programación en el que se ha desarrollado al completo la aplicación es **Python** (Python Software Foundation, 2025). La razón es que la mayoría del código, *Application Program Interfaces* (APIs) y aplicaciones sobre IA están implementadas en este lenguaje, por lo que se han desarrollado muchas bibliotecas que nos resultarán útiles. Además, se trata de un lenguaje muy sencillo, por lo que aumenta la legibilidad y la facilidad de uso.

El almacenamiento del contexto histórico del personaje se realizará a través de una base de datos. Se ha escogido **ChromaDB** (Chroma, 2025) ya que se trata de una base de datos vectorial, lo cual ayudará a guardar dicha información de manera precisa y a la recuperación de las piezas claves de contexto en la cadena RAG.

Para el control de versiones hemos elegido la herramienta más extendida, **GitHub** (GitHub, 2025). Esta permite al desarrollador tanto mantener un registro de los cambios que se han hecho sobre la aplicación como compartir la aplicación entre múltiples dispositivos de desarrollo. Este último factor ha sido clave ya que, como explicaremos más adelante, ha sido necesario probar distintas características de varios dispositivos.

Hemos decidido desarrollar esta aplicación sobre el IDE (*Integrated Development Environment*) **VSCode** (Microsoft, 2025). De nuevo, su facilidad de uso otorga múltiples ventajas y la compatibilidad con repositorios de GitHub, así como la posibilidad de uso remoto de computadoras y servidores a través de conexiones SSH (*Secured Shell*). El uso remoto ha sido imprescindible para ejecutar la aplicación en dispositivos distintos, como hemos mencionado.

Por último, establecemos como requisito que el ordenador (*Personal Computer*, PC) sobre el que se ejecute la aplicación tenga una **GPU**, necesariamente de **NVIDIA** (NVIDIA, 2025). Necesitamos este soporte *hardware* para poder paralelizar tareas de nuestra aplicación y así lograr un tiempo de respuesta viable para nuestros usuarios.

2.3.0.1. Bibliotecas de uso

Dentro del lenguaje Python, cabe destacar el uso de algunas bibliotecas que han sido fundamentales para el desarrollo de la aplicación:

- **Streamlit** (Streamlit, 2024). Hemos optado por mantener casi intacta la GUI respecto de la aplicación original, la cual estaba desarrollada sobre esta biblioteca.
- **Langchain** (LangChain, 2024). Es la biblioteca más extendida para la creación de *chatbot*s y permite la cooperación de varios modelos de IA en cadena para generar respuestas. Además, posee funcionalidades para dividir un documento de texto (*splitting*) en bloques más pequeños. Esto es útil para guardar la información de la base de datos de nuestro RAG.

- **Llama** (llama-cpp-python, 2025). Además de evaluar algunos LLMs de llama, utilizaremos la biblioteca llama-cpp-python como *wrapper* de los propios LLMs (sean o no modelos de Llama). Es decir, crearemos instancias de estos modelos a través de un envoltorio usando esta biblioteca. Esta pieza es fundamental, pues estos *wrappers* permiten que los modelos sean ejecutados sobre la Unidad Gráfica de Procesamiento (*Graphic Processing Unit*, GPU), como veremos más adelante.
- **Transformers** (Hugging Face, 2025e). La utilizaremos también como *wrapper* para instanciar LLMs.

Todas estas bibliotecas y otras útiles las almacenaremos en un entorno virtual de **Anaconda** (Anaconda Inc., 2020), separando las bibliotecas instaladas en el sistema de las de nuestra aplicación. De este modo, la aplicación se vuelve fácilmente instalable y transportable a otros dispositivos. Además, se evitan conflictos de dependencias por bibliotecas instaladas en el sistema.

2.3.0.2. Hugging Face

En el desarrollo de la aplicación ha sido fundamental el uso de Hugging Face (Hugging Face, 2025a). Se trata de una página web que sirve como repositorio de modelos de IA. Además, posee múltiples comparadores con filtros según las características de los modelos que facilita la evaluación de éstos (por ejemplo, los comparadores de LLMs[3] o de *embedders*[4]).

[3]https://huggingface.co/open-llm-leaderboard
[4]https://huggingface.co/mteb

Capítulo 3

Diseño y arquitectura

Nuestra aplicación tiene como núcleo el *chatbot* con sistema RAG para alimentar las respuestas. Para lograr el funcionamiento de este chatbot, hemos dispuesto en coordinación varios de los modelos de IA que ya nombramos previamente.

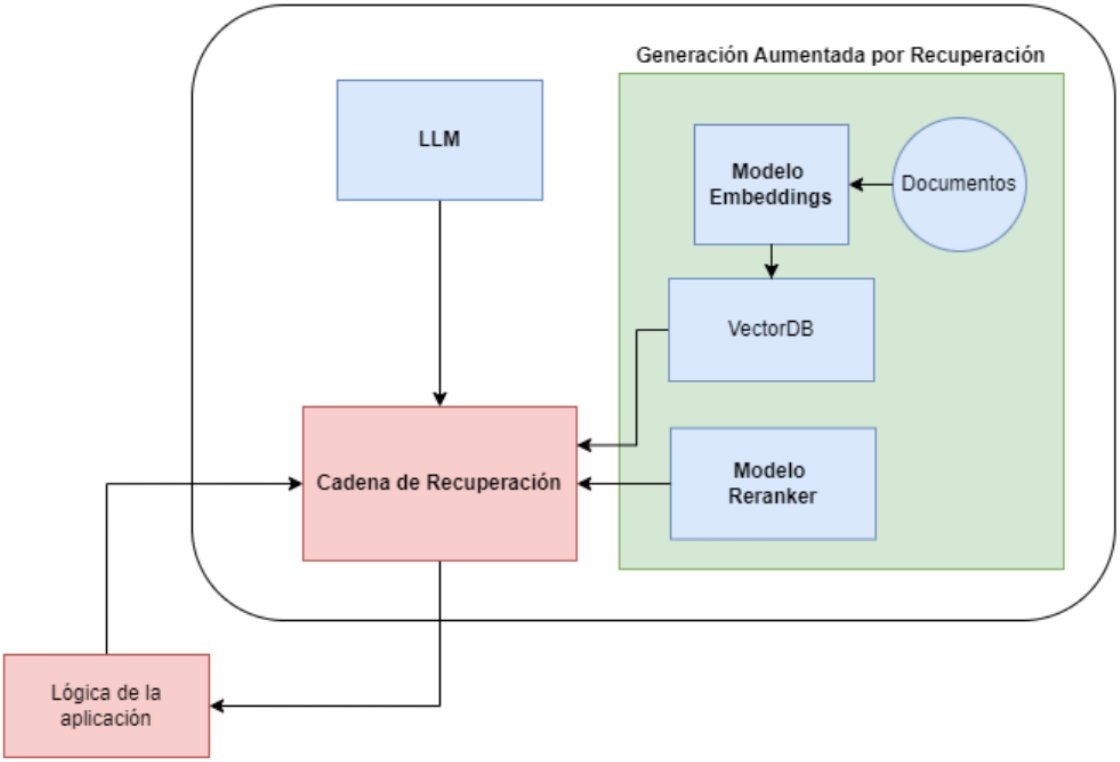

Figura 3.1: Arquitectura del *chatbot* con sistema RAG.

La coordinación de estos modelos la desempeñamos sirviéndonos de una **cadena de recuperación** (*retrieval chain*). En un principio, optamos por usar las cadenas diseñadas en Langchain. Esto facilita su uso evitando al desarrollador tener que abordar aspectos básicos tales como la conexión de unos sistemas con otros. Sin

embargo, dado que estas facilidades también conllevan un acceso limitado a la cadena para poder modificarla, decidimos que era mejor realizar una implementación completa usando *pipelines* de Python.

Los *pipes* o tuberías son una manera de unir distintas piezas de una aplicación conectando la salida de una con la entrada de la siguiente. Es un concepto parecido a las tuberías de Unix/Linux. Denotamos cadena o *pipeline* a la secuencia de varias tuberías seguidas.

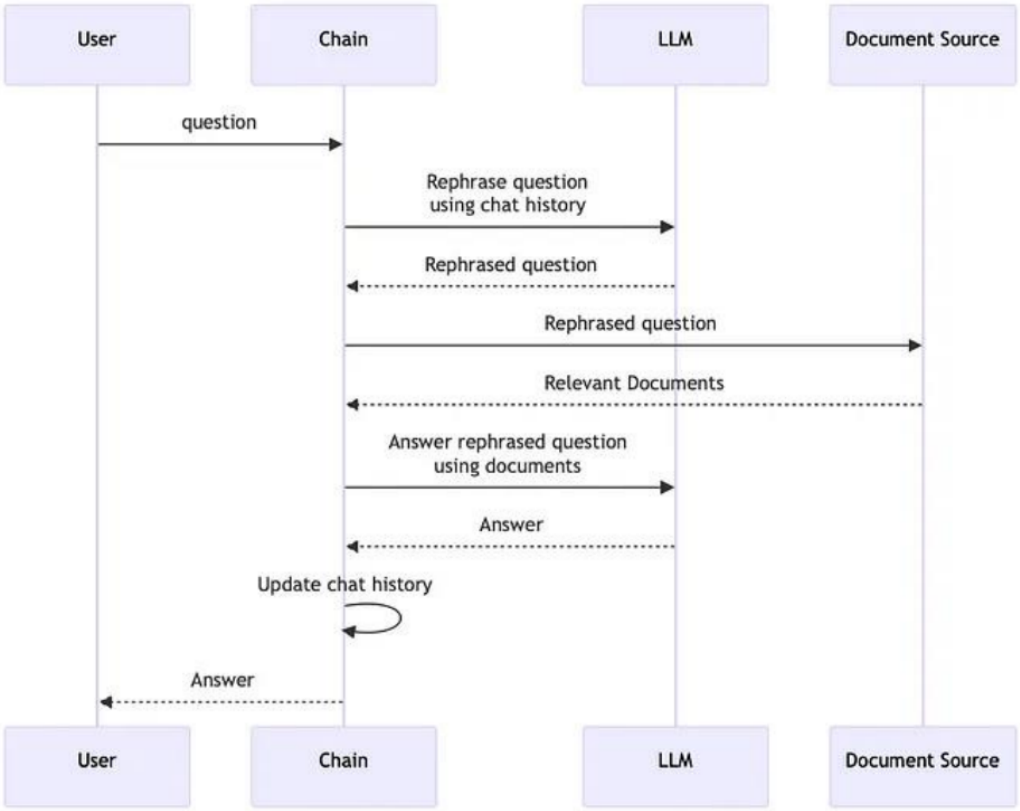

Figura 3.2: Cadena de recuperación (Díaz, 2023).

En nuestro contexto, uniremos los distintos modelos con *pipes* para que interactúen unos con otros. Mediante esta metodología, nosotros diseñanemos la manera en que interactúan los modelos entre sí dentro de la cadena. Desde entonces, sólo nos concernirá la inicialización de la cadena y su invocación. Es decir, una vez diseñada la aplicación, simplemente incorporamos los elementos necesarios a la cadena y la usamos proporcionándole la entrada inicial. El diseño de nuestra cadena de recuperación es el siguiente:

1. Entrada inicial ("*question*" en la Fig. 3.2). Recibe la pregunta que realiza el usuario.

2. Reformulación (primer "*rephrased question*" en la Fig. 3.2). Si no es la primera pregunta y, por tanto, hay un historial del *chat*, se realiza una primera llamada al LLM con un *prompt* que le indica que reformule la pregunta dado el historial del chat anterior para hacerla independiente del contexto. Veámoslo con un ejemplo:

   USUARIO: ¿Dónde vives?
   CHATBOT: Yo, Francisco de Arobe, vivo en Esmeraldas, en la bahía de San Mateo.
   USUARIO: Descríbeme el paisaje de allí.

   Para responder a esta segunda petición, solicitamos al LLM que reconstruya la pregunta con tal de que ésta no sea dependiente de las preguntas anteriores. Una reformulación válida podría ser "Descríbeme el paisaje de Esmeraldas, en la bahía de San Mateo". Si se trata de la primera pregunta nos ahorramos este paso.

3. Recuperación de información ("*relevant documents*" en la Fig. 3.2). Con la pregunta reformulada se realiza una búsqueda por similitud en la base de datos para obtener las piezas de información más relevantes a la pregunta. Para ello se transforma la pregunta en un *embedding*, se aplica una función de similitud con el resto de *embeddings* de la base de datos y, una vez devueltos los *chunks* más relacionados, estos son reorganizados y filtrados por el *reranker*.

   Es fundamental realizar este paso con anterioridad a la recuperación de información, ya que si hubiésemos buscado las piezas más relevantes a la pregunta sin reformular, la falta de contexto sobre el lugar del que se está solicitando información habría dado lugar a una búsqueda poco precisa y, consecuentemente, a una respuesta no acertada, que se percibiría como una alucinación del *chatbot*.

4. Respuesta del *chatbot* (primer "*answer*" en la Fig. 3.2). En este momento, se produce la segunda llamada al LLM, con la pregunta y la información relevante. El nuevo *prompt* le indica al LLM que debe responder a la pregunta sirviéndose de la información aportada.

5. Salida final ("*update chat history*" y segundo "*answer*" en la Fig. 3.2). Se devuelve la última respuesta del LLM al usuario y se actualiza el historial del *chat*.

Dejamos a continuación los dos *prompts* que se utlizan para enviar consultas al LLM. El *prompt* que se utiliza para la reformulación de la pregunta es el siguiente:

> Dado el chat anterior, reformula la siguiente pregunta para hacerla independiente del contexto. Responde en español. Pregunta:

> {Espacio para la pregunta}
> Escribe a continuación la pregunta reformulada:

Posteriormente, con la pregunta reformulada y el contexto aportado por el sistema RAG, se realiza una segunda petición al LLM para que responda a la pregunta, dándole unas instrucciones claras sobre cómo debe responder. El *prompt* que utilizamos es el siguiente:

> Tu nombre es don Francisco de Arobe, un personaje histórico del siglo XVI. Responde en una sola frase clara y directa. No des explicaciones largas. Sé breve. Si no sabes algo, di que no lo sabes. Pregunta:
> {Espacio para la pregunta}
> Ayúdate del siguiente contexto para responder:
> {Espacio para el contexto}
> Escribe a continuación tu respuesta en primera persona y en español:

3.1. Interfaz gráfica y manual de usuario

Vamos a describir el diseño de la GUI. Primeramente, es necesario indicar que la interfaz gráfica tiene dos modos: uno para usuario y otro para desarrollador. El segundo modo se diferencia del primero en que muestra algo más de información sobre la aplicación (los modelos utilizados) y permite interactuar con la base de datos. En concreto, permite borrarla y añadir archivos a ésta. Estas acciones no corresponden al usuario normal de la aplicación, así que las ocultamos en el primer modo. Dicho modo se determinará en la variable de entorno "DEV_MODE", dentro del archivo "config.yaml". El manual, junto con las imágenes que mostraremos, se corresponden al primer modo de uso de la aplicación.

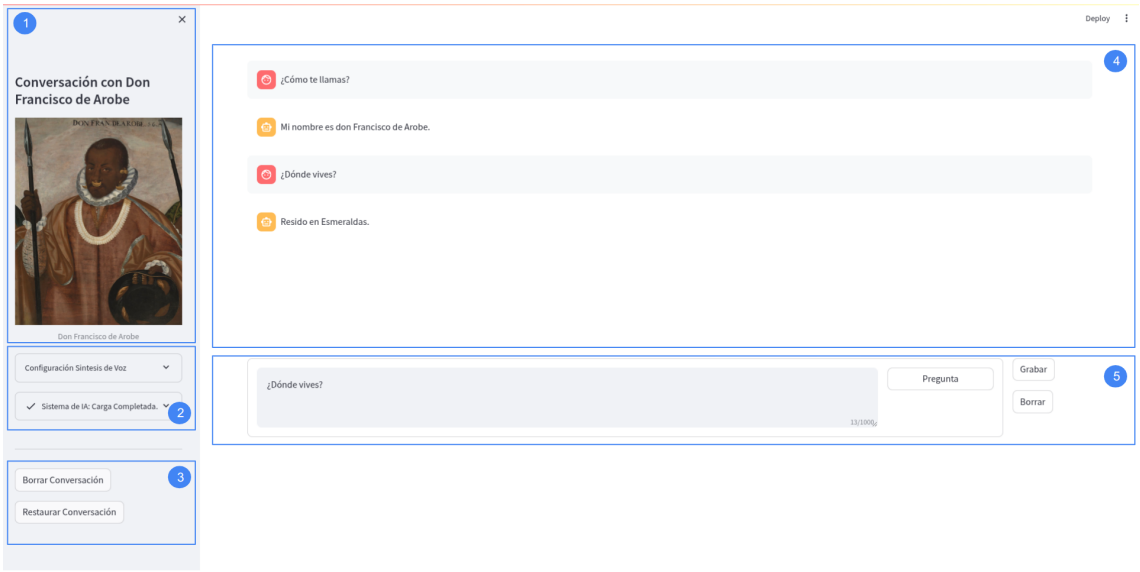

Figura 3.3: Imagen de la GUI.

La Figura 3.3 es una captura de pantalla de la GUI en el modo usuario. Los recuadros azules y la enumeración de éstos han sido añadidos en pos de ayudar a la explicación. En la sección (**1**) mostramos simplemente una imagen de don Francisco de Arobe extraída del cuadro "Los mulatos de Esmeraldas" y su nombre. Está dispuesto así para que el usuario conozca al personaje y se simule más una conversación real.

Por supuesto, es necesario que el usuario pueda observar las respuestas del *chatbot*. Además, para que sea más comprensible, hemos dispuesto también las preguntas que realizó el usuario. Toda la conversación se muestra a modo de historial del *chat* en (**4**). Los mensajes están situados de manera cronológica, de forma que el mensaje más reciente es el que se sitúa en la parte inferior del historial del chat. Las entradas con el icono en rojo corresponden con preguntas del usuario y las otras, las del icono amarillo, con las respuestas del *chatbot*.

A continuación tenemos dos botones que manejan el historial del chat (**3**). Esta sección tiene como objetivo que el usuario pueda empezar nuevas conversaciones o retomar conversaciones anteriores.

En la parte inferior, esto son las secciones (**2**), (**3**) y (**5**) de la GUI se encuentran las distintas herramientas para interactuar con la aplicación. Aquí las mencionaremos pero en el manual de usuario describiremos más detalladamente cómo usarlas.

En la (**2**) hay dos botones desplegables para interactuar con los modos de salida de respuesta del *chatbot* y para mostrar algo de información sobre el sistema de IA.

Las sección (3) y (5) sirven para interactuar con el *chatbot* y el historial del *chat*. Lo más importante es quizás el cuadro de texto de la sección (5) donde el usuario puede escribir sus consultas al *chatbot*.

3.2. Conclusiones de la arquitectura

La arquitectura propuesta se ha diseñado siguiendo un enfoque modular y escalable, de manera que los distintos modelos de IA se interconectan mediante cadenas o *pipelines*. Esta estructura permite definir de forma clara el flujo de información entre componentes, garantizando que cada modelo cumpla un rol específico dentro del sistema.

Todos los modelos se instancian una única vez al comienzo de la ejecución de la aplicación. Los modelos de voz se almacenan en una clase encargada del manejo de audio y se utilizan desde dicha clase. El resto, se añaden como piezas de la cadena.

El uso de tuberías facilita la sustitución de modelos sin necesidad de rediseñar toda la aplicación, lo que refuerza la modularidad y asegura la capacidad de evolución futura del sistema. Esta característica resulta especialmente relevante en un contexto en el que las herramientas y modelos de inteligencia artificial evolucionan con rapidez.

Además, la arquitectura cumple con los requisitos funcionales planteados: asegurar la coherencia en las respuestas, mantener un historial de conversación y permitir la integración de mecanismos de recuperación de información, así como la síntesis y grabación de voz. Todo ello garantiza que el sistema no solo sea técnicamente robusto, sino también flexible y mantenible a largo plazo.

Capítulo 4

Selección de modelos de IA

En este capítulo detallaremos los modelos concretos que hemos escogido para el sistema de IA.

4.0.1. Modelos de LLM

Para seleccionar en primer lugar los modelos que vamos a comparar hemos utilizado una clasificación de LLMs según su ejecución en español (La Leaderboard de Hugging Face[1]).

En dicha tabla, hemos seleccionado en los filtros las tareas más relevantes para nuestro LLM. Como se trata de un *chatbot* en español basado en un personaje histórico, las tareas son las siguientes:

- **Spanish**. Mide el nivel de comprensión del español que tiene.
- **COPA_es.** Evalúa la capacidad de razonamiento e inferencia. En concreto, se centra en la causalidad: ¿qué causa o resulta de una situación?
- **EsCoLA.** Puntúa la comprensión lectora a partir de textos escolares, como los que poblarán nuestra base de datos.
- **OffendES.** Detecta lenguaje ofensivo. Es muy útil para nuestra aplicación, ya que se desplegará en lugares públicos y accederán a ella personas de toda índole.
- **OpenBookQA_es.** Evaluación de tipo "libro abierto", donde el modelo razona con conocimiento externo.
- **RagQuAS.** Tarea que valora las respuestas de un LLM en un sistema RAG.

[1] https://huggingface.co/spaces/la-leaderboard/la-leaderboard

- **XNLI_es.** Mide la capacidad del LLM de inferencia textual en español. Es decir, dada una premisa y una hipótesis, debe decidir si la hipótesis se infiere, se contradice o es neutral con respecto a la premisa.
- **XQuAD_es.** Dado un párrafo y una pregunta, evalúa la capacidad del modelo de encontrar la respuesta exacta dentro del texto.

Nombre LLM	Average	Spanish	COPA_es	E
google/gemma-2-9b-it	45,44	47,25	78,8	3
google/gemma-2-2b-it	45,37	40,5	68,8	2
meta-llama/Meta-Llama-3.1-8B-Instruct	43,57	43,05	72	2
utter-project/EuroLLM-9B-Instruct	42,99	40,33	72,4	2
bertin-project/Gromenauer-7B	40,29	39,97	76	1
mistralai/Mistral-7B-Instruct-v0.3	38,57	40,8	63,2	1
Qwen/Qwen2.5-7B-Instruct	38,10	41,46	66,4	2
01-ai/Yi-1.5-9B-Chat	33,87	33,89	41,6	1
utter-project/EuroLLM-1.7B-Instruct	31,90	27,54	44,8	2
meta-llama/Llama-3.2-1B-Instruct	29,88	25,77	32	0

Figura 4.1: Captura de pantalla de tabla con datos extraídos de La Leaderboard. La tabla se encuentra en la hoja "Análisis LLM" del documento adjunto en el apéndice C.

Finalmente, en base a los resultados obtenidos en la Figura 4.1[2] , hemos seleccionado para examinar los modelos **gemma-2-9b-it** y **Meta-Llama-3.1-8b-Instruct**. Sin embargo, dado que el conjunto de datos sobre el que se han evaluado en Hugging Face estos LLMs es relativamente pequeño, hemos decidido escoger algunos otros modelos de lenguaje para examinar.

Después de investigar, de entre los modelos de lenguaje más conocidos que funcionan bien en español se encuentran los modelos Mistral. Estos son una familia de modelos de lenguaje *open-source*[3] desarrollados por la empresa francesa Mistral AI. De esta familia hemos escogido concretamente los modelos **CapybaraHermes-2.5-Mistral-7B** y **Mistral-7B-Instruct-v0.2**.

Todos los modelos escogidos son demasiado grandes para ejecutar en nuestro PC. Nos hemos servido de la cuantización para reducirlos a un tamaño viable.

[2]Se ha recortado a propósito con el fin de que se viesen correctamente los nombres de los modelos. Se puede ver correctamente en el documento que se menciona en la descripción de la imagen.

[3]De código abierto, es decir, que se pueden utilizar y modificar.

Recordamos que todos estos modelos los vamos a utilizar cuantizados porque nuestro PC no nos permite ejecutar el modelo original, pero la aplicación está preparada para instanciar modelos no cuantizados. Simplemente queremos comparar los distintos LLMs.

Para compararlos entre ellos, hemos realizado cuatro conjuntos de preguntas, cada uno con un propósito diferente:

- Preguntas estándares. Preguntas independientes sobre don Francisco de Arobe. Intentan medir el nivel de precisión del LLM en el global del sistema de IA.
- Preguntas con hilo. Se trata de una serie de preguntas con un hilo conductor o con coherencia entre ellas. Evalúa la capacidad del LLM de usar el historial del *chat*, es decir, las preguntas y respuestas anteriores, para responder a cada nueva pregunta.
- Preguntas extrañas. Serie de preguntas sobre las que don Francisco de Arobe no tiene conocimiento. Evalúa la capacidad del LLM de descartar respuestas que no cobrarían sentido en boca de nuestro personaje. En los *prompts* hemos indicado al LLM que no trate de inventarse respuestas a preguntas que no puede hallar en la información que le proporciona el sistema RAG. En estos casos, el LLM sencillamente debe responder que no sabe la respuesta.
- Preguntas inapropiadas. Se trata de preguntas que representan una conversación inapropiada o fuera de tono. Queremos comprobar la capacidad del *chatbot* de evitar conflictos y responder educadamente, sin usar expresiones vulgares o comentarios indebidos. Es importante porque va a estar expuesto en un lugar público.

Hemos analizado la cantidad de alucinaciones que tienen estos modelos y la precisión de las respuestas usando el contexto aportado por el sistema RAG. Para no extender en exceso este documento, los resultados obtenidos están recogidos en el documento adjunto en el apéndice C en las hojas "Preguntas estándares", "Preguntas con hilo", "Preguntas raras" y "Preguntas inapropiadas".

En el capítulo siguiente especificaremos mejor el flujo de la aplicación y la manera en la que se generan las respuestas. Sin embargo, nos es preciso adelantar que para cada pregunta que el usuario formula al *chatbot*, se realizan dos peticiones al LLM. La primera petición es la de reformular la pregunta del usuario a partir del historial del *chat*. Posteriormente, con la pregunta reformulada, se realiza una nueva consulta al LLM para que responda dicha pregunta con los pedazos de información que se le aportan.

Aclaramos estas cuestiones para poder explicar mejor qué instrucciones se le dan al LLM y así examinar más adecuadamente cómo se ajusta a estas y también para resaltar que el error muchas veces no se va a deber a la segunda petición, la que propiamente generará la respuesta al usuario, sino a la reformulación incorrecta de la

pregunta. Observando el proceso de ejecución del programa, hemos observado cómo en algunos modelos la alucinación se produce en esta primera petición, resultando así en un desvarío de la respuesta final.

Realizaremos a continuación un análisis de dichos resultados para evaluar cada modelo.

En primer lugar, fijémonos en los modelos Capybara Hermes de la familia Mistral. Algo que resalta a simple vista es que las respuestas son generalmente extensas, lo cual se opone radicalmente a la instrucción que se le proporciona de ser breve. Además, muchas veces el LLM no se reduce a responder como si fuese el personaje histórico, sino que se sale de su papel. Podemos observar esto claramente en las filas 4, 5, 6 y 10 de la tabla de preguntas estándares entre otras.

Esta información nos indica claramente que el LLM no procesa bien las instrucciones, lo cual tiene sentido ya que no es un modelo *instruct* de los que indicamos previamente.

En otras preguntas (como la fila 4 de las preguntas raras) sufre alucinaciones más serias. Estas alucinaciones en su mayoría se trata de errores que se propagan por la mala reformulación de la pregunta del usuario en la petición inicial que se realiza al *chatbot*. Las respuestas, sin embargo, con coherentes hasta que llegamos a la sección de preguntas inapropiadas. En algunas de ellas no parece entender la pregunta, pues responde de manera indistinta sin detectar la ofensa.

En todos los demás modelos se ha aplicado *fine-tuning* para convertirlos en modelos *instruct*. La diferencia es muy notable en varios aspectos como la longitud de las respuestas, la adherencia al papel que debe interpretar (simular que responde como si fuese don Francisco, respondiendo en primera persona), etc. La adecuación de las respuestas a las instrucciones dadas se nota especialmente en los últimos dos modelos, los de Google y Meta.

Los modelos *instruct* de la familia Mistral, aunque en menor medida que los Capybara Hermes, algunas veces se extienden demasiado o se salen del papel (como en las filas 8 y 9 de las preguntas estándares). No sufre alucinaciones serias en ninguna de las preguntas estándares o con hilo, siempre responde de manera coherente ý con información relacionada a la pregunta. Sin embargo, sí utiliza expresiones confusas (“muerte traicionosa” en la fila 6 de las preguntas con hilo o “ Preferisco zapatillas” fila 10 de las preguntas raras) o responde de manera vaga e inconclusa a lo que se le pide (fila 7 de las preguntas con hilo).

Donde sí se observan errores importantes es en las últimas dos tandas de preguntas. Los más evidentes son claramente provocados por la reformulación de la pregunta (filas 5, 7 y 8 de las preguntas raras). Cabe resaltar también que, en general, entiende las preguntas inapropiadas y las responde educadamente, así que descartamos que pueda responder vulgarmente o crear una situación inoportuna.

El modelo gemma-2-9b-it-Q6_K.gguf es sin duda el que mejor se desenvuelve de todos ellos. Capta perfectamente las instrucciones, responde siempre de manera coherente sin inventarse información y siempre mantiene su papel como don Francisco de Arobe. Bajo las instrucciones que se le da, las respuestas pueden parecer algo cortas, pero no es problema, ya que esto se puede configurar con algunos parámetros del modelo, como mostraremos en el capítulo 5. En ocasiones, ocurre que aunque no desvaría demasiado, no termina de responder a lo que se le pregunta (filas 2 y 4 de las preguntas con hilo). Esto no es preocupante porque sucede poco y porque siempre es preferible que no termine de responder a la pregunta a que se invente información o responda falsamente. En caso de que el usuario no quede satisfecho con la respuesta tiene la opción de volver a preguntar.

En la mayoría de las preguntas raras, la respuesta del modelo es "No sé", que es exactamente lo que pedíamos que respondiera cuando no supiese la respuesta a una pregunta. En aquellas que responde, al menos lo hace con lógica, por lo que no supone un problema.

En la sección de preguntas inapropiadas, muchas veces capta la ofensa y responde rechazando explícitamente el conflicto y la falta de educación (filas 5, 7 y 8). En el resto, de nuevo responde que no sabe.

El último modelo, el de Meta, también tiene muy buenos resultados. Sin embargo, en la última sección de preguntas, la de preguntas inapropiadas, sufre varias alucinaciones y produce respuestas con poco sentido o que no continúan el hilo de la pregunta.

Por todas estas razones, el modelo que hemos decidido usar en nuestra aplicación es el **gemma-2-9b-it-Q6_K.gguf** de Google. Conviene recordar que todos los modelos probados son modelos cuantizados, lo que explica parcialmente algunos errores. Teniendo a disposición un *hardware* mejor, por ejemplo, un ordenador con 24 GB de VRAM, los resultados mejorarían muy notablemente.

4.1. Modelos de generación de *embeddings*

Para seleccionar el *embedder* adecuado nos hemos fijado en el Massive Text Embedding Benchmark Leaderboard (MTEB Leaderboard[4]) de Hugging Face.

Hemos especificado en los filtros las tareas que debe realizar nuestro *embedder*. Como se trata de un sistema RAG, las tareas son las siguientes:

- **Retrieval**. Mide la capacidad del modelo para recuperar documentos relevan-

[4]https://huggingface.co/spaces/mteb/leaderboard

tes a una *query*. Es la tarea más relacionada con RAG junto con la siguiente.

- **STS.** La Semantic Text Similarity mide cuánto de similares son dos textos.
- **InstructionRetrieval.** Mide la recuperación basada en instrucciones o preguntas naturales, que será nuestro caso, pues es muy cercano a cómo los usuarios interaccionan con un *chatbot*.
- **Compositionality.** Evalúa si el modelo puede razonar sobre conceptos compuestos. Por ejemplo, una *query* para la que esta tarea es relevante podría ser "¿En qué ciudades que no son colonias españolas has estado alguna vez?".
- **ZeroShotClassification.** Evalúa si el modelo puede clasificar textos en clases que nunca ha visto durante su entrenamiento.
- **Speed.** Mide la velocidad del modelo para generar *embeddings*.

El resto de parámetros a filtrar los hemos ajustado para seleccionar el modelo más acertado. Estos son los 10 mejores resultados de la tabla.

Rank (Bord…	Model	Zero-shot	Memory Us	Number of Para…	Embedding Dime…	Max Tokens
1	gemini-embedding-001	100%	Unknown	Unknown	3072	2048
2	GritLM-7B	100%	13813	7B	4096	4096
3	jasper_en_vision_language_v1	100%	3802	1B	8960	131072
4	gte-Qwen2-7B-instruct	⚠ NA	29040	7B	3584	32768
5	GritLM-8x7B	100%	89079	57B	4096	4096
6	multilingual-e5-large	100%	2136	560M	1024	514
7	stella_en_1.5B_v5	100%	5887	1B	8960	131072
7	text-multilingual-embedding-002	100%	Unknown	Unknown	768	2048
9	multilingual-e5-large-instruct	100%	1068	560M	1024	514
10	multilingual-e5-base	100%	1061	278M	768	514

Figura 4.2: Captura de pantalla con los 10 mejores resultados del MTEB Leaderboard.

En nuestro caso concreto, no es necesario un *embedder* muy sofisticado, ya que no poseemos demasiada información sobre don Francisco de Arobe. Eso nos permite seleccionar un modelo que no sea demasiado grande. Por ello, hemos seleccionado definitivamente el modelo **multilingual-e5-large-instruct**.

4.2. Modelo *reranker*

Hemos optado por el modelo **bge-reranker-base**[5]. Se trata de un modelo ligero y multilingüe que soporta consultas en español con muy buena precisión.

[5]https://huggingface.co/BAAI/bge-reranker-base

4.3. Modelos de voz

El modelo **Faster Whisper** (SYSTRAN, 2025) transforma audio a texto. La aplicación está diseñada para que, al grabar un audio, se calibre el nivel de ruido que detecta el micrófono durante unos instantes muy breves (unas décimas de segundo). Posteriormente, se escucha al usuario hasta que se detecta un periodo con un nivel de ruido similar al del entorno que se fijó en la calibración. En ese momento se detiene la grabación y se utiliza Faster Whisper para transformar dicho audio a texto.

En un primer momento, en lugar de este modelo de IA se utilizó el paquete *Speech recognition* de Python (Bhat, 2020) con el motor de reconocimiento de Google. Si bien detectaba correctamente las palabras pronunciadas, no era capaz de interpretar la entonación. Por ejemplo, si tratábamos de realizar una pregunta al *chatbot*, escribía las palabras de la pregunta pero no las interrogaciones. Esta manera de grabar audio era muy pobre y por eso nos decantamos finalmente por el modelo que hemos mencionado, el cual sí percibe correctamente la entonación del usuario.

Por otra parte, la aplicación también es capaz de transformar las respuestas del *chatbot* a audio. Hemos escogido el modelo **XTTS V2** (Coqui AI, 2025) porque, entre los modelos gratuitos, es de los que mejores resultados daba (Srivastav et al., 2024).

Además, dicho modelo posee la funcionalidad de estilizar la producción de voz con un audio. En nuestro caso, hemos utilizado una grabación de voz de Pepe Mediavilla, uno de los actores de voz de doblaje en español más famosos. En definitva, la aplicación ahora responderá mediante un audio, reproduciendo el texto generado por el LLM y estilizada con la voz del actor de doblaje.

En la decisión de elección de ambos ellos hemos priorizado que sean rápidos y ligeros.

Capítulo 5

Optimización y calibración

En este capítulo detallaremos las elecciones y opciones de configuración que hemos determinado con tal de mejorar el rendimiento de la aplicación en términos de tiempo de ejecución y uso de memoria.

Para ello, principalmente nos basaremos en el uso de la tarjeta gráfica del PC que hemos utilizado para este trabajo, la cual dispone de 8 GB de VRAM.

5.1. Uso de la GPU para aumentar la eficiencia de tiempo

El problema principal de nuestra aplicación es el tiempo de respuesta. Inicialmente, la aplicación se ha implementado para su ejecución íntegra sobre la Unidad Central de Procesamiento (*Central Processing Unit*, CPU). Así, se ha logrado un desarrollo rápido de las funcionalidades. Una vez lograda la correcta integración de todas las partes, se realizaron pruebas para comprobar el rendimiento de la aplicación. En dichas pruebas, se observó que el tiempo de respuesta del *chatbot* rondaba en torno a los dos minutos. Se vuelve inviable mantener al usuario esperando tanto tiempo a la respuesta a una consulta tan sencilla como "¿Quién eres?".

El motivo inicial que se pensó fue la limitación de hardware. La Memoria de Acceso Aleatorio (*Random Access Memory*, RAM) es la memoria que la CPU usa para acceder a datos rápidamente. Por otro lado, las GPUs utilizan la Memoria de Vídeo de Acceso Aleatorio (*Video Random Access Memory*, VRAM) para el acceso a datos, que está mejor optimizada. El uso de una RAM de 16GB parecía justificar la lentitud de la aplicación. Sin embargo, se comprobó que la solución mejor no está en adquirir una CPU con mayor capacidad, sino en quitar peso de ejecución a la CPU y cargarlo sobre una GPU. Para eso, hemos establecido como requisito que el PC en el que se ejecute la aplicación tenga tarjeta gráfica. Las razones por las que hemos optado por esta solución son las siguientes.

Primeramente, el diseño de las GPUs, a diferencia de las CPUs, permite realizar muchos cálculos en paralelo. Esto se debe a que poseen decenas de miles de núcleos, mientras que las CPUs sólo tienen unos pocos (normalmente entre 4 y 32). Esta característica resulta muy útil para sistemas como el nuestro porque los modelos de IA realizan tareas con muchísimos cálculos que se pueden paralelizar como, por ejemplo:

- Entrenamiento de redes neuronales
- Procesamiento de grandes lotes de datos o de modelos con muchos parámetros. Éste es nuestro caso, ya que los LLMs que hemos probado poseen miles de millones de parámetros.
- Multiplicación de matrices.

Las GPUs originalmente se utilizaban solo para dibujar imágenes y renderizar gráficos, de ahí que la memoria que utilizan se denomine VRAM. La plataforma *Compute Unified Device Architecture* (CUDA) creada por NVIDIA permite a las tarjetas gráficas realizar los ćalculos que acabamos de indicar.

Además, la proporción de coste de energía con la efectividad (medida en relación a la velocidad y la precisión) en tareas es mucho menor en GPUs. Es decir, que gasta mucha menos energía utilizando modelos más precisos de manera más rápida.

No se debe desestimar este factor ya que, en vista a que este sistema se utilice a diario en un centro cultural que se visita con frecuencia, los costes de energía se pueden volver significativos.

El PC utilizado en este trabajo cuenta con la tarjeta gráfica NVIDIA GeForce RTX 3060 Ti, que tiene un espacio de 8GB. Se trata de una tarjeta gráfica de gama media, pero dará buenos resultados en una aplicación como la nuestra.

Si bien es cierto que para las evaluaciones hemos obtenido otro ordenador cuya tarjeta gráfica posee mayor capacidad con el fin de poder instanciar modelos más grandes, ello únicamente nos ha resultado útil a la hora de evaluar la precisión del sistema. La precisión del sistema la evalúamos mediante la tasa de acierto de una pregunta que se le ha realizado (que responda la información que se le pide, que no sufra alucinaciones, etc.).

En esta sección, procederemos con la tarjeta gráfica anterior, ya que simplemente queremos examinar la mejora de rendimiento, es decir, del tiempo de respuesta, a través del uso de una GPU.

La aplicación se ha implementado de forma que se pueda modificar rápidamente para volver a su uso íntegro sobre CPU. Sin embargo, como hemos justificado antes, se desestima dicha posiblidad para su funcionamiento adecuado.

Dicho todo esto, procedemos a explicar en qué tareas hemos introducido el uso de una GPU.

5.1.0.1. Instancia del LLM

El mayor obstáculo en cuanto a la falta de rendimiento proviene del uso del LLM. La aplicación se detiene enormemente cuando el LLM está en ejecución sobre la CPU generando una repuesta. Ésto cobra mucho sentido, ya que el LLM está implementado con muchos más parámetros que los otros modelos. Consecuentemente, es el modelo de mayor tamaño con gran diferencia respecto a los otros. Esto se da incluso cuando usamos modelos cuantizados del mismo. A nosotros nos resultarán muy útiles porque nuestra GPU no admite modelos demasiado grandes.

En la siguiente tabla mostramos la comparación de tamaño entre el modelo de LLM que usaremos y el *reranker* para comparar el impacto del tamaño de cada uno en el uso de la memoria.

Característica	**Mistral-7B**	**BGE-Reranker**
Número de parámetros	7,000 millones (7B)	350 millones (0.35B)
Tamaño en FP16 (aprox.)	14 GB	700 MB
Tamaño en formatos cuantizados	4 - 7 GB	300 - 700 MB
Uso principal	Modelo de lenguaje general	Reranking de resultados

Tabla 5.1: Comparación entre modelos Mistral-7B y BGE-Reranker

Sin embargo, cuando analizamos el uso de la GPU a la hora de generar respuestas ésta aumenta muy levemente (5-10 % de su capacidad total), indicador claro de que no estamos usando bien los recursos de los que disponemos.

Inicialmente establecimos de manera fija que dicha instancia se cree usando la biblioteca *ctransformers*. Esta biblioteca permite usar modelos de LLM de manera local, pero es muy restringida en cuanto al control de los parámetros de la instancia del modelo y no tiene soporte de GPU para la aceleración en la inferencia de datos.

Percibido esto, hemos optado por usar las dos bibliotecas que enumeramos en las especificaciones de la arquitectura: ***transformers*** y ***llama-cpp-python***. La primera de ellas es la más universal. Se utiliza para instanciar y correr cualquier de modelo HuggingFace localmente. También tiene soporte para los modelos cuantizados por GPTQ.

En la sección sobre la infraestructura del proyecto del capítulo 2, mencionamos que tomaríamos LLMs en formato GGUF. Actualmente, la biblioteca *transformers* no soporta este formato. Por esa razón, hemos incluido también *llama-cpp-python*, que sí tiene soporte para archivos en este formato.

Por otro lado, aunque es menos amigable para el desarrollador porque requiere un conocimiento algo más profundo, permite una configuración más exhaustiva de los parámetros, proporcionando un mayor control sobre el uso de la GPU.

Debido a que los archivos binarios de la biblioteca *llama-cpp-python* utilizan una versión específica de CUDA para algunas de sus dependencias, distinta de la versión que utiliza Conda, necesitamos descargar manualmente este paquete (clonando el repositorio de llama-cpp-python) e instalarlo (asegurando de que indicamos que lo instale con soporte para CUDA).

Por otro lado, a la hora de ejecutar los binarios cuando cargamos el LLM, el entorno Conda también utiliza su propia versión de ciertas bibliotecas. En concreto, de glibcxx, el ABI[1] de C++ que provee GCC (el compilador de GNU). Esta versión de Conda no incluye algunos símbolos que necesitan los binarios precompilados.

Por así decirlo, estamos tratando de utilizar una máquina a la que le falta una pieza o que la tiene muy vieja. Por eso, antes de ejecutar el programa, hará falta obligar a python que ignore la versión de Conda y utilice la nuestra.

Ahora ya tenemos el paquete *llama-cpp* para Python y estamos en condiciones de crear instancias para modelos GGUF. Como mencionamos, esta biblioteca permite la modificación de múltiples parámetros para ajustar el uso de nuestra GPU. Estos son los más importantes:

- **n_gpu_layers**. Indica cuántas capas de la red neuronal se cargan sobre GPU en lugar de la CPU. Las capas que permanecen en la CPU aumentan drásticamente el tiempo de ejecución, por lo que conviene aumentar el valor de este parámetro al máximo. Sin embargo, a veces resultará imposible debido a nuestra limitación de recursos.

- **n_threads**. Número de hilos de CPU usados para procesear capas que no están sobre la GPU. Aprovechar el paralelismo de la CPU lógicamente también ayuda a reducir el tiempo de ejecución. Es inconveniente usar todos los hilos de la CPU, ya que ésta se puede saturar y afectar a la ejecución del resto de la aplicación.

  Conviene ajustar este parámetro al número de núcleos de la CPU. Ésta tiene una serie de núcleos físicos que vienen determinados en las especificacioens del *hardware*. A veces un mismo núcleo físico puede actuar como dos núcleos lógicos. Es decir, de cara al funcionamiento, el PC percibe que tiene más núcleos de los que realmente posee y puede asignar más hilos. Esta técnica se conoce como *Hyper-Threading*.

[1] *Application Binary Interface*

Si bien de esta manera se puede aprovechar mejor la CPU cuando hay muchas tareas pequeñas, en tareas muy pesadas como ciertas operaciones de inferencia de los LLMs, usar todos los hilos lógicos puede aumentar la comptencia por recursos internos del PC. Ello puede provocar una ganancia mínima o incluso negativa del rendimiento de la CPU.

Nosotros lo ajustaremos siempre al número de núcleos físicos del PC, ya que la mayoría de capas las cargaremos sobre GPU

- **n_ctx**. Cantidad máxima de *tokens* que el modelo puede considerar como contexto para generar su respuesta. Un *token* es la unidad mínima de texto que un LLM entiende y procesa. El contexto lo forman el *prompt* y los mensajes anteriores del *chat*.

  El modelo reserva inicialmente un espacio de VRAM para este contexto, por lo que aumentar demasiado el valor de este parámetro puede desbordar dicha memoria. Es conveniente también tener una longitud de contexto holgada, ya que si se supera, algunos *tokens* pueden descartarse o resumirse, provocando alucinaciones.

Tras la selección del LLM que realizamos en el capítulo 4, la configuración actual del proyecto está preparada para para usar el *wrapper* de *transformers* y el modelo gemma-2-9b-it-Q6_K.gguf. Sin embargo, el proyecto está listo para poder usar otros modelos en múltiples formatos y ambos *wrappers*. Para probar otras configuraciones, simplemente hay que modificar el archivo "config.yaml" en la carpeta principal del proyecto, en concreto, los atributos "WRAPPER", "LLM_PATH" y "LLM_NAME".

A continuación mostraremos otros parámetros de configuración de los LLMs que, si bien no influyen en el uso de la memoria, son relevantes a la hora de diseñar el modo en que los modelos generarán respuestas.

- **max_new_tokens**. Es el número máximo de *tokens* que el modelo puede generar en cada respuesta. Hay que establecer un valor relativamente alto ya que si se excede este límite, la respuesta se trunca en ese mismo instante, lo que puede dar lugar a frases sin terminar. Dado que el modelo que hemos escogido funciona bien con las instrucciones, es extraño que devuelva respuestas largas, así que resulta casi imposible que se llegue al límite.

- **temperature**. Controla el grado de aleatoriedad y creatividad. Nosotros hemos tomado un valor que se considera intermedio porque no queremos ni que se exceda inventándose información ni que sea excesivamente determinista o escueto.

- **top_p**. Controla el rango de opciones que el modelo considera para generar cada token. En primer lugar el modelo genera múltiples opciones y las ordena de la más probable a la menos probable. Luego calcula la probabilidad acumulada empezando por la más probable. Se truncan las respuestas cuya probabilidad acumulada sea mayor que el valor de este parámetro, el cual oscila entre 0 y 1.

- **repetition_penalty**. Es un factor que modifica la probabilidad que el modelo escoja nuevos *tokens* en función de los *tokens* ya generados. El valor neutral es 1, con el cual no se penaliza ni se favorecen a los tokens ya generados. Nosotros lo hemos establecido ligeramente superior para que evite la redundancia y sea más conciso.

5.1.0.2. Aceleración por GPU para los demás modelos de IA

El resto de modelos tienen parámetros o métodos que permiten su carga en GPU. Dado el pequeño tamaño de éstos, no supone un alivio grande del tiempo de ejecución, pero es conveniente.

Actualmente el proyecto está configurado para que todos los modelos se carguen sobre la tarjeta gráfica. Sin embargo, nuestro PC actual, como ya hemos mencionado, tiene sólo 8 GB de VRAM, por lo que no puede almacenar todos los modelos en GPU al mismo tiempo.

Por esta razón, nosotros a la hora de evaluar sólo hemos cargado sobre la tarjeta gráfica el modelo que pretendíamos analizar, mientras que el resto se mantenían en la CPU. La configuración del proyecto actual está prevista para que el LLM se cargue sobre GPU y el resto sobre CPU. De esta manera conseguimos aligerar la aplicación lo máximo posible. El otro modelo que también resulta demasiado lento sobre CPU es el de XTTS v2. Sin embargo, ahora mismo no podemos ejecutar los dos sobre GPU.

Las variables que determinan sobre qué memoria se cargan el LLM, el modelo XTTS v2 y los otros modelos son “LLM_DEVICE”, “TTS_DEVICE” y “DEVICE”, respectivamente. Lo óptimo sería priorizar que los modelos se carguen sobre GPU en el orden que los acabamos de mostrar, es decir, priorizamos que LLM se cargue en la tarjeta gráfica antes que el TTS y éste antes que el resto de modelos.

Capítulo 6

Manual de usuario

En este capítulo vamos a proporcionar un manual de usuario que describa cómo se debe usar la aplicación. De todos modos, se ha diseñado de manera muy intuitiva para que el usuario medio (un visitante del museo) pueda utilizarla sin este manual.

Para ejecutar la aplicación es necesario abrir la terminal desde una consola en el directorio del proyecto (o abrirla y luego cambiar el directorio al del proyecto). Posteriormente hay que ejecutar los siguientes comandos:

```
conda activate tfg
```

```
streamlit run app.py
```

Es necesario tener instalada correctamente la aplicación (ver apéndice A).

Una vez ejecutada la aplicación nos encontraremos ante la GUI de la Figura 3.3. En lo siguiente nos referiremos continuamente a esta figura para desarrollar el manual.

En la sección (**2**) tenemos dos botones desplegables. El primero de ellos sirve para cambiar el modo de reproducción de respuesta del *chatbot*. Éste puede ser sólo por texto o por texto y voz. Por defecto viene marcado el modo de sólo texto. Si se selecciona el segundo modo, al terminar de elaborar la respuesta se mostrará por pantalla en el historial del *chat* (en la zona (**4**)) y se reproducirá también por audio. El segundo desplegable muestra algo de información de la carga del sistema de IA.

En la sección (**3**) se encuentran dos botones para manejar el historial del *chat*. El primero de ellos sirve para borrarlo y así reiniciar la conversación con el *chatbot*. De esta manera, si un nuevo usuario accede de primeras a la aplicación, puede resetearlo para empezar una nueva conversación. También tiene como finalidad el terminar conversaciones que pudieran estar algo desviadas de lo que el usuario pide, y así poder empezar de nuevo. El segundo botón restaura el último historial del chat. En caso de que el usuario quiera rectificar un borrado de conversación, puede hacerlo

mediante este botón.

Por último, tenemos la sección principal de la GUI (5). Aquí encontramos el cuadro de texto en el que podemos escribir para enviar preguntas al *chatbot*. El envío se produce al pulsar el botón de "Pregunta". El botón que tiene escrito "Grabar" sirve para grabar audios de voz y poder así comunicarse con el *chatbot* vocalmente si así se desea. La aplicación se bloquea mientras se está grabando el audio para evitar conflictos. El audio grabado no se envía directamente al *chatbot*, sino que se refleja sobre el cuadro de texto en el que podenos escribir. Esto permite que el usuario pueda editar la grabación en caso de que no quedase conforme con ella. El último botón de esta sección, que tiene escrito "Borrar", sirve para eliminar del cuadro de texto el contenido actual.

Capítulo 7

Conclusiones y trabajo futuro

Este proyecto ha logrado desarrollar un *chatbot* que simula al personaje histórico don Francisco de Arobe para ser expuesto en una sala del Museo de América, junto al cuadro "Los mulatos de Esmeraldas".

Para ello se han integrado varios modelos de IA. Estos modelos son un LLM, un *embedder*, un *reranker*, un modelo TTS y un modelo STT.

Los tres primeros modelos componen la parte del sistema de IA que implementa el *chatbot* con RAG. Se ha realizado así para aportar al LLM información suficiente con el objetivo de que pueda responder simulando a don Francisco de Arobe. La información sobre este personaje se encuentra en una base de datos vectorial que se genera creando *embeddings* de la información que poseemos dividida en *chunks*.

En la selección del LLM se ha priorizado que no se salga del papel que le corresponde (el de don Francisco de Arobe), que responda con información relevante a la consulta, que no sufra alucinaciones y que responda en un tono formal y educado. Estas dos últimas razones se deben a que la aplicación será expuesta en una sala pública de un museo, por lo que cualquier persona que acceda al museo podría usarla. Por consiguiente hay que evitar que se creen situaciones incómodas.

Todos los modelos han sido elegidos teniendo en consideración la limitación de *hardware* que poseemos: un ordenador con 8 GB de VRAM.

El diseño de la GUI y las funcionalidades de voz añadidas gracias a los modelos STT y TTS se han dispuesto para que la aplicación resulte sencille de usar sin necesidad de explicación y para acercar lo máximo posible la experiencia a una conversación real.

La aplicación ha sido sometida a una serie de pruebas para comprobar su funcionamiento. Los resultados han sido muy buenos: con el modelo que aconsejamos, no hay alucinaciones, el tono se mantiene formal y consigue responder con información relevante o, al menos, no responder.

El rendimiento de la aplicación es bueno. Hemos evaluado cada modelo por separado, ya que nuestro PC no era capaz de cargar todos los modelos al mismo tiempo sobre su GPU. El tiempo de respuesta está en torno a los siete segundos, un tiempo más que aceptable sabiendo que la aplicación tiene que transformar el audio a texto, generar una respuesta y luego convertir la respuesta a audio de nuevo. Si no se usan las funciones de audio el tiempo de respuesta disminuye a unos cinco segundos.

7.1. Trabajo futuro

Por último, vamos a detallar algunos puntos que pueden mejorar el rendimiento de la aplicación. Para esta sección, conviene tener en cuenta que la aplicación se ha construido sobre el personaje de don Francisco de Arobe y funciona muy bien para esta finalidad. Sin embargo, es muy fácilmente extrapolable a otros personajes. Simplemente hay que cambiar el documento con el que se puebla la base de datos y el *prompt* de la segunda petición al LLM (en el que se especifica que debe actuar como si fuese don Francisco de Arobe).

Por ello, algunas de estas mejoras van orientadas a un posible uso de la aplicación en otros contextos más complejos como, por ejemplo, un personaje cuya documentación sea mucho más extensa.

- Mejora del formato de la documentación. Actualmente, la base de datos se puebla con información extraída de un PDF con imágenes, *links*, etc. Dado que dicho documento es pequeño y hay poca información entre la que buscar, esto no influye demasiado. Esta variedad de elementos en el PDF puede llegar a confundir la búsqueda de información del sistema RAG e incluso inducir alucinaciones en caso de que la documenetación sea mucho más extensa. Es por ello que convendría tener un documento que únicamente contenga la información que se pretende buscar: sin números de página, *links*, imágenes, etc.

- Uso de Unstructured para el *splitting*. Similarmente al punto anterior, está la posibilidad de usar Unstructured. Este paquete de Python divide el documento en *chunks* teniendo en cuenta el formato. Por ejemplo, es capaz de diferenciar imágenes y sacarlas a parte. La motivación del uso de este paquete es la misma que en el punto anterior.

- Optimización de la elección de modelos de IA. En cuanto a los modelos que hemos tomado para desarrollar el sistema de IA, siempre hemos tenido la limitación del hardware. En caso de poseer dispositivos con mayor potencia, sería viable seleccionar otros modelos mejores. En concreto, lo primero sería utilizar un LLM no cuantizado, para evitar las alucinaciones que se puedan dar por la

reducción de precisión del modelo.

Además, los modelos *reranker* y *embedder* también puedes ser seleccionados entre otros de una gama más alta. En la elección de estos modelos mencionamos que priorizamos siempre que fuesen ligeros. La razón sigue el hilo de los primeros puntos: la búsqueda en la base de datos era sencilla.

- Mejora de la GUI. La GUI que se despliega es funcional, pero se puede mejorar su diseño para hacerla más atractiva y amigable. Por ejemplo, se puede implementar una imagen animada de don Francisco de Arobe que se mueva a la vez que se producen las respuestas del *chatbot* para dar mayor sensación de estar ante una persona real. Esto se puede lograr utilizando modelos de IA ya existentes.

  Algo que también se puede añadir y que ya poseen otros *chatbots* es la impresión gradual del texto que se produce. Es decir, que no se muestre la respuesta al completo nada más terminar de elaborarla, sino que se vaya imprimiendo conforme se elabora para así dar la impresión de estar desarrollando un discurso a lo largo del tiempo.

Introduction

Throughout history, human beings have always strived to progress as a civilization. In recent years, this progress has focused on the development of new technologies (mobile devices, the Internet, etc.). One of the most recent outcomes of this evolution is the emergence of Artificial Intelligence (AI).

AI is a branch of computer science aimed at developing systems capable of performing tasks that, until recently, required human intelligence. Such tasks include reasoning, learning, and decision-making.

Unlike traditional programs, which rigidly follow a set of instructions, AI systems can adapt to new situations. This is because they extract patterns from data and later use them to handle completely new inputs, making these systems more flexible.

In this context, museums and cultural centers face an increasing challenge: maintaining the interest of an audience that is increasingly accustomed to immediate, interactive, and digital experiences. These institutions play a fundamental role in preserving and disseminating history and cultural heritage. However, their somewhat traditional methods of knowledge transmission often prove less appealing to younger generations, who are used to the constant use of the Internet, social networks, and interactive technologies.

To address this challenge, many museums have begun integrating technological tools to provide more dynamic and immersive experiences. These initiatives seek both to complement physical exhibitions and to improve accessibility and visitor interaction with cultural content. Among them, we can find simple projects such as building maps of the premises, as well as more recent developments such as virtual reality.

AI emerges as an opportunity to transform the way users engage with information in a museum. Through techniques such as natural language processing, image recognition, or the generation of personalized content, AI can adapt to the needs of each visitor and provide more engaging, interactive, and personalized experiences.

Following this line, this work aims to fully develop an application that uses AI

to increase public interest in these cultural centers through a closer and more interactive activity.

7.2. Proposal from the Museo de América

We will now detail our project, which originated from a proposal by the Museo de América. We will review the initial proposal and its context.

To facilitate explanations about the museum's works and to attract younger audiences, the Museo de América in Madrid proposed the development of a *chatbot* representing a historical figure. Visitors would be able to interact with it, serving both as a tourist attraction and as an automatic guide capable of answering questions about the character.

The proposal was made in relation to the painting "Los Mulatos de Esmeraldas." Specifically, the project aimed to develop a *chatbot* that would represent Don Francisco de Arobe, the central figure of the painting. Before proceeding, it is necessary to introduce Don Francisco de Arobe and his context in order to better understand the project and its challenges.

During the era of conquest and colonization (16th–18th centuries), many Spaniards traveled to the Indies of the Americas to continue the expansion of the Catholic Monarchy initiated under the reign of Isabella and Ferdinand. One of the most renowned conquistadors was Francisco Pizarro. In 1532, he landed on the northern coast of Ecuador before heading south toward the heart of Peru. With the fall of the Inca Empire, this border region began to depopulate due to disease and forced relocations. Additionally, many ships carrying deserters and escaped slaves wrecked along this coast. Survivors of these wrecks took refuge in the jungle and terrorized nearby populations. For all these reasons, this region, named Esmeraldas, became regarded as an almost cursed land.

On one of these wrecked ships traveled Andrés Mangache, an enslaved African. Having managed to escape with his beloved, he gathered followers and confronted the local indigenous peoples. They allied with some natives, such as those from the "land of Arobe," from whom his descendants likely adopted the surname "Arobe."

The Mangache family had two sons, one of whom was Don Francisco de Arobe. He eventually assumed the role of chief after the deaths of his father and his elder brother, Juan.

Although his leadership was not entirely favorable to the local authorities, the Arobe family gained a reputation for being more cooperative and approachable than other groups and leaders, even converting to Christianity.

As a result of this relationship and the assistance they provided to many other shipwrecked individuals, an agreement was reached between the Arobe family and the royal judge Juan Barrio de Sepúlveda. In 1599, Don Francisco traveled to Quito with two of his sons to pledge loyalty to the Crown, after which he was granted the title of governor of the Esmeraldas territory.

As a consequence of this event, the painting "Los Mulatos de Esmeraldas" was created and sent to Philip IV, where it remained in the royal collection until the late 19th century. It was later transferred to the Museo de América, where it is still displayed today. In the painting (see Fig. 7.1), Don Francisco is depicted in the center, with his two sons at his sides.

Figure 7.1: Los Mulatos de Esmeraldas (1599), Andrés Sánchez Gallque

7.3. General Approach and System Requirements

With this project in mind, we must specify an initial approach to our application and its requirements.

The *chatbot* will be implemented as a Retrieval-Augmented Generation (*RAG*) system. This will enable it to handle natural language understanding and generation while limiting its responses largely to a restricted documentary base. The core of this RAG system is a Large Language Model (LLM), which provides the natural language processing capabilities. This will be integrated with other AI models to personalize the *chatbot*, ensuring it expresses itself as Don Francisco de Arobe. Since Don Francisco lived during the early Modern Age, it is important that the *chatbot* not only attempt to speak like a person from that era but also limit its knowledge

to his historical context.

The first task is challenging, as the written records from that era are mostly formal documents such as legislative or literary texts, making it difficult to determine exactly how Don Francisco would have spoken. However, we aim to give his words a tone of solemnity and simplicity, appropriate for the context of an African slave who rose to the role of governor.

For the second task, historical information will be provided through other AI models, as mentioned earlier. The restriction of the *chatbot*'s knowledge will be achieved by choosing an LLM with minimal hallucinations and strong performance in answering only with the provided information.

It is especially important that the *chatbot* avoids hallucinations, as these could confuse users or even lead to the presentation of false facts or inappropriate language. This will also be ensured through the choice of LLM.

To launch the application, at least one output device (likely a screen) and one input device (such as a keyboard) are required. The interface will be developed on these devices and must be simple while meeting the needs of the majority of users.

This project builds on a previous undergraduate thesis[1]. That system will serve as a starting point to address unresolved aspects and deliver a finished application ready for deployment at the Museo de América. From the original project, the concept and the GUI have been retained, but the rest of the application has been restructured and reimplemented to adapt to the new system architecture, advances in infrastructure, and to resolve performance issues.

7.4. Main Objective and Sub-Objectives

The main objective is to design a *chatbot* developed to adopt the role of a historical figure, in our case, Don Francisco de Arobe. In other words, we aim to provide context and specific instructions to a *chatbot* so that it behaves as a character, simulating a real conversation with him.

This application will be installed in a museum hall, next to the painting of the character represented by the *chatbot*. It will therefore be exposed to interaction with any visitor who enters the museum and passes through the hall. This highlights the importance of avoiding hallucinations and, in particular, inappropriate responses that could make visitors uncomfortable or prove unsuitable in this context. For the same reason, the tone of the responses is also relevant: they must be formal and polite.

[1] https://github.com/mgm48/tfg

Moreover, since the goal is to simulate a historical figure, we must ensure that the responses align with the character. This means not only possessing detailed knowledge about the figure and his context but also limiting responses strictly to this knowledge. In other words, the *chatbot* should not be able to answer questions outside the scope of what the character would know. For this, we will use documentation about the figure (limited, given its nature) to feed the *chatbot*'s responses.

The key objectives of the project are:

- Integrate a Generative AI (GAI) model as the foundation of the *chatbot*. A local LLM will be used to facilitate fine-tuning and optimize performance.
- Implement context retrieval to help the local LLM generate responses consistent with the historical figure's knowledge. For this, a database will store the character's information along with a retrieval system. This approach, known as RAG, ensures responses consistent with the LLM's role.
- Design a Graphic User Interface (GUI) that enables interaction with the *chatbot* and displays the chat history with the ongoing conversation.
- Develop the functionalities necessary for a speech-to-speech *chatbot*, enabling verbal interaction and vocalized responses. This requires an AI model for speech synthesis and another for speech recognition.
- Provide the technical user with a way to add or remove documents from the knowledge base. This functionality will remain hidden from general users.

7.5. Work Plan

The phases to achieve the above objectives are as follows:

1. **State of the Art.** January–February 2025.
   First, we will carefully analyze the current application, identifying its weaknesses to address them and produce a complete, ready-to-use application.

   Once these issues are analyzed, we will study various AI models for the aforementioned functionalities, seeking the most suitable choice for each.

2. **System Architecture Design and Infrastructure Selection.** February 2025.
   We will define the functional flow of the application to correctly integrate all components.

Subsequently, we will choose the software and hardware platforms that best suit the intended use and models.

3. **Application Development.** February–May 2025.
   We will implement the functionalities and determine the best configuration for a complete and finished application.

4. **Testing and Debugging.** May–August 2025.
   We will evaluate the efficiency and accuracy of the application and implement the necessary adjustments to improve these aspects. This phase is essential to ensure that the application performs well under appropriate conditions.

5. **Development of Installation and User Guide.** September 2025.
   To facilitate usage, we will provide clear installation instructions and a user manual.

The rest of the document is organized as follows:

- Chapter 2. We will examine the state of the art, describing the current situation in museums with new technologies. We will also introduce the models required and some supporting infrastructure tools.

- Chapter 3. We will describe the architecture of the application in detail.

- Chapter 4. We will study current models most suitable for our purpose and propose one for each type.

- Chapter 5. We will explain how we optimized the use of models to improve application performance.

- Chapter 6. We will provide a guide for using the application.

- Chapter 7. We will summarize all the work carried out and outline some future work to continue developing this project.

Conclusions and Future Work

This project has successfully developed a *chatbot* that simulates the historical figure Don Francisco de Arobe to be exhibited in a room of the Museo de América, alongside the painting "Los Mulatos de Esmeraldas".

To achieve this, several AI models have been integrated. These models include an LLM, an *embedder*, a *reranker*, a TTS model, and an STT model.

The first three models make up the part of the AI system that implements the *chatbot* with RAG. This approach was taken to provide the LLM with sufficient information so that it could respond while simulating Don Francisco de Arobe. Information about this character is stored in a vector database generated by creating *embeddings* from the information we have, divided into *chunks*.

In selecting the LLM, the main priorities were that it should stay within the role assigned to it (that of Don Francisco de Arobe), respond with relevant information to the query, avoid hallucinations, and answer in a formal and polite tone. The last two reasons are particularly important because the application will be exhibited in a public museum hall, meaning anyone visiting the museum could use it. It is therefore crucial to avoid generating uncomfortable situations.

All models were chosen taking into consideration the hardware limitation we had: a computer with 8 GB of VRAM.

The design of the GUI and the voice functionalities added through the STT and TTS models were arranged to make the application easy to use without requiring explanation, and to bring the experience as close as possible to a real conversation.

The application has been subjected to a series of tests to verify its performance. The results have been very positive: with the recommended model, there are no hallucinations, the tone remains formal, and the responses provide relevant information or, at the very least, refrain from answering.

The performance of the application is good. Each model was evaluated separately, since our PC was not able to load all of them on its GPU at the same time.

The response time is around seven seconds, which is more than acceptable given that the application must convert audio to text, generate a response, and then convert the response back into audio. If the audio functions are not used, the response time decreases to about five seconds.

7.6. Future Work

Finally, we will detail some points that could improve the performance of the application. For this section, it is important to bear in mind that the application was built around the character of Don Francisco de Arobe and works very well for this purpose. However, it can be very easily extrapolated to other characters. One simply needs to replace the document used to populate the database and update the prompt in the second request to the LLM (where it is specified that it must act as if it were Don Francisco de Arobe).

For this reason, some of the proposed improvements are aimed at potential use in more complex contexts, such as a character with much more extensive documentation.

- Improvement of documentation format. Currently, the database is populated with information extracted from a PDF containing images, *links*, etc. Since this document is small and the amount of information to search is limited, this does not affect the system significantly. However, this variety of elements in the PDF may confuse the RAG system's search process and even induce hallucinations in the case of much more extensive documentation. Therefore, it would be advisable to have a document that contains only the information intended for retrieval: without page numbers, *links*, images, etc.

- Use of Unstructured for *splitting*. Similar to the previous point, there is the possibility of using Unstructured. This Python package divides the document into *chunks* while taking formatting into account. For example, it can differentiate images and extract them separately. The motivation for using this package is the same as in the previous point.

- Optimization of AI model selection. Regarding the models we used to develop the AI system, we were always limited by the hardware. If more powerful devices were available, it would be possible to select better models. Specifically, the first step would be to use a non-quantized LLM to avoid hallucinations caused by reduced model precision.

  In addition, the *reranker* and *embedder* models could also be chosen from higher-end options. As mentioned earlier, we prioritized lightweight models.

The reasoning follows the same logic as in the first points: the database search was straightforward.

- Improvement of the GUI. The GUI currently deployed is functional but could be improved to make it more attractive and user-friendly. For example, an animated image of Don Francisco de Arobe could be implemented, moving in sync with the chatbot's responses to enhance the feeling of interacting with a real person. This could be achieved using existing AI models.

  Another feature that could be added, already present in other *chatbots*, is gradual text rendering. Instead of displaying the full response immediately upon completion, the text could appear progressively as it is generated, thus giving the impression of delivering a speech in real time.

Bibliografía

Amazon. Amazon Web Services: Machine Learning e IA. 2023. Disponible en: https://aws.amazon.com/es/what-is/retrieval-augmented-generation/.

Anaconda Inc. Anaconda Software Distribution. 2020. Disponible en: https://docs.anaconda.com/.

Bhat, N. M. pyttsx3. Python Package Index (PyPI), 2020. Disponible en: https://pypi.org/project/pyttsx3/.

Boiano, S., Borda, A., Gaia, G., Rossi, S. y Cuomo, P. Chatbots and New Audience Opportunities for Museums and Heritage Organisations. En *Electronic Workshops in Computing (EVA London 2018)*, páginas 164–171. BCS Learning & Development Ltd., 2018.

Boiano, S., Cuomo, P. y Gaia, G. Real-time Messaging Platforms for Storytelling and Gamification in Museums: A case history in Milan. En *Electronic Workshops in Computing (EVA 2016)*, páginas 291–293. BCS Learning & Development Ltd., 2016.

British Museum. How to explore the British Museum from home. 2020. Disponible en: https://www.britishmuseum.org/blog/how-explore-british-museum-home.

Chroma. Chroma Documentation. 2025. Disponible en: https://docs.trychroma.com/.

Coqui AI. XTTS V2: Voice Generation Model. 2025. Disponible en: https://huggingface.co/coqui/XTTS-v2.

Devlin, J., Chang, M.-W., Lee, K. y Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. En *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, vol. 1, páginas 4171–4186. Association for Computational Linguistics, 2019. Disponible en: https://arxiv.org/abs/1810.04805.

Díaz, J. Conversational Retrieval Chain, how does it work? 2023. Disponible en: https://medium.com/@jerome.o.diaz/langchain-conversational-retrieval-chain-how-does-it-work-bb2d71cbb665.

Europa Press. Sube la edad media de visitantes en el Museo del Prado: 45 años en 2023 frente a 38 en 2022. *Diario de Burgos*, página n.p., 2024. Disponible en: https://www.diariodeburgos.es/noticia/zeb15d1b8-dda9-0e6e-bfa391cc058bbac0/202401/sube-la-afluencia-a-los-museos-en-2023-con-record-de-el-prado,.

GitHub. Documentación oficial de GitHub. 2025. Disponible en: https://docs.github.com/.

Goodfellow, I., Bengio, Y. y Courville, A. *Deep Learning*. MIT Press, 2016. Disponible en: https://www.deeplearningbook.org/.

Hugging Face. Hugging Face Documentation. 2025a. Disponible en: https://huggingface.co/docs.

Hugging Face. Latin America LLM Leaderboard. 2025b. Disponible en: https://huggingface.co/spaces/la-leaderboard/la-leaderboard.

Hugging Face. Massive Text Embedding Benchmark (MTEB). 2025c. Disponible en: https://huggingface.co/mteb.

Hugging Face. Open LLM Leaderboard. 2025d. Disponible en: https://huggingface.co/open-llm-leaderboard.

Hugging Face. Transformers Documentation. 2025e. Disponible en: https://huggingface.co/docs/transformers.

Ibermuseos. Informes de impacto del COVID-19 en el ecosistema del museo. 2020. Disponible en: https://www.ibermuseos.org/informes-de-impacto-del-covid-19-en-el-ecosistema-del-museo/.

ICOM. Definición de museo. Normas y Directrices de ICOM. 2022. Disponible en: https://icom.museum/es/recursos/normas-y-directrices/definicion-del-museo/.

LangChain. LangChain. 2024. Disponible en: https://python.langchain.com.

LeCun, Y., Bengio, Y. y Hinton, G. Deep learning. *Nature*, vol. 521(7553), páginas 436–444, 2015.

llama-cpp-python. Documentación oficial de llama-cpp-python. 2025. Disponible en: https://llama-cpp-python.readthedocs.io/.

Longhi-Heredia, S. A., Quezada-Tello, L. L. y Cappello, G. Estrategias digitales de difusión museística en tiempos del COVID-19: Estudio comparativo entre Ecuador, España y Perú. *Contratexto*, 2021. Disponible en: https://www.researchgate.net/publication/360497824_Estrategias_

digitales_museisticas_durante_la_pandemia_Museo_Nacional_del_Prado_y_Museo_Nacional_Centro_de_Arte_Reina_Sofia_Museum_digital_strategies_during_the_pandemic_Prado_National_Museum_and_Reina,.

López López, Y., Cisneros Álvarez, P., Ávila Rodríguez, M. y Delage González, I. Museum reinvention in the face of the pandemic challenge: digitization strategies, heritage, and audiences. *Estoa. Journal of the Faculty of Architecture and Urbanism*, vol. 13(26), páginas 147–163, 2025. Disponible en: https://publicaciones.ucuenca.edu.ec/ojs/index.php/estoa/article/view/5134.

Microsoft. Visual Studio Code Documentation. 2025. Disponible en: https://code.visualstudio.com/docs.

Morales Carmona, I. y Freitag, V. Los Museos en el Siglo XXI: nuevos retos, nuevas oportunidades. *Revista Digital do LAV*, vol. 7, páginas 30–49, 2014. Disponible en: https://www.redalyc.org/articulo.oa?id=337030167004.

NVIDIA. NVIDIA Documentation Hub. 2025. Disponible en: https://docs.nvidia.com/.

OpenWebinars. Qué son las redes neuronales y sus aplicaciones. 2023. Disponible en: https://openwebinars.net/blog/que-son-las-redes-neuronales-y-sus-aplicaciones/.

Python Software Foundation. Python Documentation. 2025. Disponible en: https://docs.python.org/3/.

Red Hat. ¿Qué es la inteligencia artificial generativa? Ejemplos y riesgos. 2024. Disponible en: https://www.redhat.com/es/topics/ai/what-is-generative-ai.

Rumelhart, D. E., Hinton, G. E. y Williams, R. J. Learning representations by back-propagating errors. *Nature*, vol. 323(6088), páginas 533–536, 1986.

Srivastav, V., Pouget, L., Fourrier, C., Lacombe, Y., Horse, M., Gandhi, S. y mrfakename. TTS Arena: Benchmarking Text-to-Speech Models in the Wild. 2024. Disponible en: https://huggingface.co/blog/arena-tts.

Streamlit. Streamlit: The fastest way to build and share data apps. 2024. Disponible en: https://streamlit.io/.

SYSTRAN. Faster-Whisper: A fast Whisper implementation. 2025. Disponible en: https://github.com/SYSTRAN/faster-whisper?tab=readme-ov-file.

Taggart, E. This Immersive Exhibition Lets You Step Aboard the Titanic. *MyModernMet*, 2025. Disponible en: https://mymodernmet.com/the-legend-of-the-titanic-exhibition/,.

Unstructured.io. Unstructured Documentation. 2025. Disponible en: https://docs.unstructured.io/welcome.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. y Polosukhin, I. Attention Is All You Need. En *Advances in Neural Information Processing Systems (NeurIPS)*, páginas 5998–6008. 2017. Disponible en: https://arxiv.org/abs/1706.03762.

Zhang, A. Speech Recognition (Version 3.8) [Software]. 2017. Disponible en: https://github.com/Uberi/speech_recognition#readme.

Zárate Martín, M. A. y García Ferrero, A. Los museos, oferta consolidada para el turismo sostenible y la calidad del paisaje. *Arbor. Ciencia, Pensamiento y Cultura*, vol. 193-785, página a401, 2017. Disponible en: https://arbor.revistas.csic.es/index.php/arbor/article/view/2209,.

Apéndice A

Guía de instalación

El proyecto ha sido desarrollado al completo sobre un PC con sistema operativo GNU/Linux. Sin embargo, dado que todo ha sido diseñado creando un entorno virtual aislado, es sencillo reproducirlo en otros sistemas operativos. Aquí se darán los pasos para instalarlo en un PC con GNU/Linux.

Requisitos:

- Poseer una tarjeta gráfica de NVIDIA.
- Tener instalado el NVIDIA Toolkit en la versión 12.1, lo cual se puede realizar desde aquí.
- Tener instalado Anaconda3, lo cual se puede realizar desde este link.
- Tener instalado Python en la versión 3.11, lo cual se puede realizar con el comando:

```
sudo apt install -y python3.11 python3.11-venv python3.11-dev
```

Pasos a seguir:

1. Clonar el repositorio del proyecto:

```
git clone https://github.com/rafamp2/tfg
```

2. Crear el entorno virtual y descargar los paquetes necesarios. Para ello, simplemente accedemos a la consola, nos trasladamos a la carpeta del proyecto con algo parecido a

```
cd ruta/a/la/carpeta/del/proyecto
```

y ejecutamos el archivo de instalación ya definido para armar el entorno virtual, escribiendo en la consola

```
./install.sh
```

Puede ser que se soliciten permisos para realizar alguna acción. Debemos leer atentamente las instrucciones en la consola y continuar el proceso. Este proceso puede tardar varios minutos ya que necesita instalar un entorno virtual grande.

3. Dentro de la carpeta del proyecto, creamos una carpeta nueva llamada "models". Ahí debemos descargar los modelos de IA. Para ello, tenemos que realizar lo siguiente desde una terminal situada en la carpeta "models":

   - Permitir la descarga de archivos grades con "lfs" mediante el comando:

     ```
     git lfs install
     ```

   - Clonar el repositorio del modelo TTS dentro de la carpeta "models":

     ```
     git clone https://huggingface.co/coqui/XTTS-v2
     ```

   - Clonar el repositorio del modelo Faster Whisper dentro de la carpeta "models":

     ```
     git clone https://huggingface.co/Systran/faster-whisper-medium
     ```

   - Clonar el repositorio del *reranker* dentro de la carpeta "models":

     ```
     git clone https://huggingface.co/BAAI/bge-reranker-base
     ```

   - Clonar el repositorio del *embedder* dentro de la carpeta "models":

     ```
     git clone https://huggingface.co/intfloat/multilingual-e5-large-instruct
     ```

   - Descargar el archivo "gemma-2-9b-it-Q6_K.gguf" desde este link . Crear una carpeta llamada "gemma-2-9b-it-GGUF" dentro de nuestra carpeta "models" e introducir el archivo en dicha carpeta.

Si necesitamos ajustar alguno de los parámetros del "config.yaml" como las capas de GPU que use el LLM, lo hacemos. En concreto, conviene variar el atributo "N_GPU_LAYERS", para determinar el número de capas del LLM que se cargan sobre la VRAM. Ahora mismo está establecido a 0, es decir, se carga todo sobre la CPU. Para un PC de 8 GB no conviene cargar más de 5 capas[1], pero el rendimiento será más lento. Lo ideal es usar un PC cuya tarjeta gráfica tenga al menos 16 GB. En ese caso, podremos cargar al menos 28 capas (de las 42 que posee nuestro modelo) y el rendimiento será muy bueno.

[1]Si se cargan demasiadas capas se puede desbordar la GPU y hacer saltar una excepción.

Para ejecutar la aplicación simplemente ejecutamos los siguientes comandos situándonos en una terminal en la carpeta del proyecto

```
conda activate tfg
```

```
streamlit run app.py
```

Apéndice B

Glosario

- IA: Inteligencia Artificial
- IAG: Inteligencia Artificial Generativa
- RAG: *Retrieval Augmented Generation*
- LLM: *Large Language Model*
- GPTQ: *Gradient Post-Training Quatization*
- AWQ: *Activation-aware Weight Quantization*
- GGUF *GPT-Generated Unified Format*
- API: *Application Program Interface*
- GUI: *Graphic User Interface*
- PC: *Personal Computer*
- CPU: *Central Processing Unit*
- GPU: *Graphic Processing Unit*
- RAM: *Random Access Memory*
- VRAM: *Video Random Access Memory*
- CUDA: *Compute Unified Device Architecture*
- VR: *Virtual Reality*
- TIC: Tecnologías de la Información y la Característicaomunicación
- TTS: *Text To Speech*
- STT: *Speech To Text*

Apéndice C

Material adicional

Documento adjunto: Análisis LLMs