

# Trabajo Fin de Máster

## Ingeniería de Telecomunicación

### Modelo de lenguaje generativo para tutor en sistemas de comunicaciones digitales

Autor: Sergio García López

Tutor: Juan José Murillo Fuentes

**Dpto. Teoría de la Señal y Comunicaciones**  
**Escuela Técnica Superior de Ingeniería**  
**Universidad de Sevilla**

Sevilla, 2025





Trabajo Fin de Máster  
Ingeniería de Telecomunicación

# **Modelo de lenguaje generativo para tutor en sistemas de comunicaciones digitales**

Autor:

Sergio García López

Tutor:

Juan José Murillo Fuentes

Catedrático de universidad

Dpto. de Teoría de la Señal y Comunicaciones

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2025



Trabajo Fin de Máster: Modelo de lenguaje generativo para tutor en sistemas de comunicaciones digitales

Autor: Sergio García López

Tutor: Juan José Murillo Fuentes

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2025

El Secretario del Tribunal



*A mi familia y amigos*



# Agradecimientos

---

En primer lugar, quiero expresar mi profundo agradecimiento a mis padres por el amor, cariño y apoyo que me han brindado a lo largo de los años, y en especial en este último, el cual ha estado lleno de cambios y nuevos desafíos. A mi hermano Nacho, quien también ha tenido un año crucial en su vida y al que le deseo lo mejor en su etapa universitaria. A mis abuelos, tíos y primos, que han estado apoyándome, preocupándose por mí y celebrando mis logros personales como si fueran suyos.

A mis amigos de siempre y a los que recientemente se han incorporado a mi vida: Carmen, Marina Díaz, Inés, Irene, Marina Cortés, Silvia, Edu, Nazaré, Juanca y Toni. Una vez más, quiero agradecerlos los buenos momentos que hemos pasado juntos y los que están por llegar.

A mis amigos del Erasmus: Mansi, Raúl, Chema, Alex, Álvaro, Noelia, Belén, Ana, Andrea, Susana y en particular, Abel y Marcos, por viajar desde Asturias hasta Sevilla para reencontrarnos después de casi tres años. Todavía nos quedan muchos viajes por hacer y recordar las anécdotas y vivencias que tuvimos en Irlanda.

A mi amigo Álvaro, que en su caso está estudiando el Máster en Ingeniería Aeronáutica y también se ha formado como ingeniero aquí en la ETSI. Quiero agradecerle las buenas charlas y los piques que solemos tener al hablar de fútbol, además de volver a felicitarle por los éxitos logrados en VANTUS.

A mis amigos y compañeros del Máster, por el grupo tan cohesionado que hemos formado y por los buenos momentos que hemos pasado dentro y fuera de clase durante estos dos años. Quiero agradecer especialmente a Juanjo, al que considero una de las personas más únicas y divertidas que he conocido, por los momentos vividos en la cafetería y en la biblioteca cada uno trabajando en sus respectivos trabajos. También a Juanlu y a Aitor, por nuestros análisis y bromas sobre el Sevilla y el Betis, así como a Andrés y a Candilejo por hacer las clases más amenas. A Sergio Montañés, quien también fue compañero en mi instituto y al que le deseo lo mejor en esta nueva etapa en el Máster.

Finalmente, quiero mostrar mi agradecimiento a mi tutor, Juan José Murillo Fuentes, por haber propuesto este tema tan innovador y apasionante, y a quien considero uno de los mejores docentes que he tenido durante la carrera.

*Sergio García López*

*Sevilla, 2025*



En los últimos años, los denominados LLMs (Large Language Models) han transformado los campos de la inteligencia artificial y del procesamiento del lenguaje natural. Estos sistemas, basados en arquitecturas neuronales avanzadas como el modelo Transformer y entrenados con cantidades masivas de datos, han demostrado una capacidad sin precedentes para generar textos coherentes y contextualizados. Dichos modelos han redefinido aplicaciones prácticas en áreas como la traducción automática, la atención al cliente, la clasificación de documentos e incluso la asistencia en investigación científica. Sin embargo, su rápido desarrollo plantea desafíos críticos en términos de ética, sesgos de entrenamiento, sostenibilidad computacional y adaptación a contextos específicos.

Dentro de este marco, este Trabajo Fin de Máster propone adaptar un LLM de código abierto preentrenado, como es el caso de Llama 3.1, mediante técnicas RAG (Retrieval Augmented Generation) para que actúe como un tutor especializado en el ámbito de los sistemas de comunicaciones digitales. Para ello, se utilizará una arquitectura basada en LangChain y ChromaDB, con el objetivo de realizar búsquedas semánticas sobre una base de datos vectorial construida a partir de documentos académicos del área de interés. Dichos documentos serán segmentados en fragmentos de texto y vectorizados mediante un modelo de embedding del framework Hugging Face para almacenarlos en ChromaDB. Tras realizar la búsqueda semántica, los fragmentos más relevantes se incorporarán dinámicamente al prompt del LLM, con el fin de generar respuestas contextualizadas y alineadas con la consulta. Además, se empleará la librería Transformers de Hugging Face tanto para la ejecución local del modelo como para aprovechar sus herramientas de cuantización, permitiendo reducir significativamente el tiempo de inferencia sin comprometer en exceso la precisión de las respuestas generadas.



In recent years, so-called LLMs (Large Language Models) have transformed the fields of artificial intelligence and natural language processing. These systems, based on advanced neural architectures such as the Transformer model and trained with massive amounts of data, have demonstrated an unprecedented ability to generate coherent and contextualized text. These models have redefined practical applications in areas such as automatic translation, customer service, document classification and even scientific research assistance. However, their rapid development raises critical challenges in terms of ethics, training biases, computational sustainability and adaptation to specific contexts.

Within this scope, this Master Thesis proposes to adapt a pre-trained open source LLM, such as Llama 3.1, using RAG (Retrieval Augmented Generation) techniques to act as a specialized tutor in the field of digital communication systems. For this purpose, an architecture based on LangChain and ChromaDB will be used, with the goal of performing semantic searches on a vector database built from academic documents in the area of interest. These documents will be segmented into text fragments and vectorized using an embedding model from the Hugging Face framework to store them in ChromaDB. After performing the semantic search, the most relevant fragments will be dynamically inserted into the LLM prompt, in order to generate contextualized answers aligned with the query. In addition, the Hugging Face Transformers library will be used both for the local execution of the model and to benefit from its quantization tools, allowing to significantly reduce inference time without excessively compromising the accuracy of the generated answers.

# Índice

---

<b>Agradecimientos</b>	<b>ix</b>
<b>Resumen</b>	<b>xi</b>
<b>Abstract</b>	<b>xiii</b>
<b>Índice</b>	<b>xiv</b>
<b>Índice de Tablas</b>	<b>xvi</b>
<b>Índice de Figuras</b>	<b>xviii</b>
<b>Glosario</b>	<b>xxii</b>
<b>1 Introducción</b>	<b>1</b>
1.1 Motivación	1
1.2 Antecedentes	2
1.2.1 Modelo Transformer y mecanismos de atención	2
1.3 Objetivos	6
<b>2 IA generativa y LLMs</b>	<b>9</b>
2.1 IA generativa	9
2.1.1 Fundamentos	9
2.1.2 Modalidades de IA generativa	12
2.2 Grandes modelos de lenguaje (LLMs)	14
2.2.1 Fundamentos	14
2.2.2 Prompt engineering	17
2.2.3 Retrieval Augmented Generation (RAG)	19
2.2.4 Fine-tuning	23
2.2.5 Reinforcement Learning from Human Feedback (RLHF)	24
2.3 Comparación de modelos de lenguaje	25
2.3.1 Evaluación de modelos	25
2.3.2 Modelos destacados	27
2.4 Caso de estudio: Llama 3.1	31
2.4.1 Arquitectura de Llama 3.1	31
2.4.2 Proceso de entrenamiento	32
2.4.3 Versiones de Llama 3.1	33
2.4.4 Aplicaciones prácticas	36
2.5 NotebookLM: un asistente documental basado en LLMs	36
<b>3 Implementación del sistema</b>	<b>39</b>
3.1 Arquitectura del sistema	39
3.2 Ejecución del modelo Llama 3.1	42
3.2.1 Formato del prompt	44
3.2.2 Generación de respuestas	48
3.2.3 Ejemplos de generación	49
3.3 Selección y preprocesamiento de documentos	57
3.3.1 Selección de documentos	57
3.3.2 Preprocesamiento de documentos	57

3.4	<i>Vectorización y almacenamiento en ChromaDB</i>	60
3.4.1	Generación de embeddings	60
3.4.2	Almacenamiento en ChromaDB	60
3.5	<i>Pipeline RAG con LangChain</i>	61
3.6	<i>Implementación de la API REST</i>	63
3.6.1	Endpoints de la API	64
3.6.2	Interacción con el modelo de lenguaje	65
3.6.3	Gestión de documentos del sistema RAG	69
3.7	<i>Implementación de la interfaz de usuario</i>	72
3.7.1	Interfaz conversacional	72
3.7.2	Gestión del historial de mensajes	76
3.7.3	Edición del system prompt	77
3.7.4	Administración de la base de conocimiento	78
<b>4</b>	<b>Validación</b>	<b>81</b>
4.1	<i>Creación del dataset de validación</i>	81
4.2	<i>Resultados</i>	84
<b>5</b>	<b>Despliegue del sistema</b>	<b>87</b>
5.1	<i>Creación de la imagen Docker del backend</i>	87
5.2	<i>Despliegue del backend con Docker Compose</i>	90
5.3	<i>Despliegue del frontend</i>	93
<b>6</b>	<b>Conclusiones y líneas futuras</b>	<b>95</b>
6.1	<i>Conclusiones</i>	95
6.2	<i>Líneas futuras</i>	96
	<b>Referencias</b>	<b>99</b>
	<b>Anexo</b>	<b>108</b>

# ÍNDICE DE TABLAS

---

Tabla 1-1. Matriz de atención a la salida de la función softmax del scaled dot-product attention	4
--	---



# ÍNDICE DE FIGURAS

---

Figura 1-1. Arquitectura del modelo Transformer [5]	2
Figura 1-2. Representación gráfica de los embeddings en un espacio vectorial [7]	3
Figura 1-3. Arquitectura de los mecanismos de atención [5]	5
Figura 2-1: Comparación entre los modelos generativos y discriminativos [18]	10
Figura 2-2: Arquitectura de un VAE [21]	10
Figura 2-3: Arquitectura de una GAN [24]	11
Figura 2-4: Procesos de difusión y eliminación de ruido en los modelos de difusión [28]	11
Figura 2-5: Arquitectura GPT original [30]	12
Figura 2-6: Ejemplos de generación de imágenes con pix2pix [31]	13
Figura 2-7: Arquitectura simplificada de los modelos DALL-E [39]	14
Figura 2-8: Proceso de predicción de tokens en un LLM [43]	15
Figura 2-9: Few-shot learning para el análisis de sentimientos con ChatGPT	17
Figura 2-10: Prompt para cambiar el formato de un documento XML a un diccionario Python [48]	18
Figura 2-11: Prevención de alucinaciones mediante prompt engineering	19
Figura 2-12: Arquitectura de los sistemas RAG [50]	20
Figura 2-13: Proceso de chunking y generación de embeddings en una base de datos vectorial [56]	21
Figura 2-14: Comparación de las respuestas de Llama 3 8B y Llama 3 8B Instruct	23
Figura 2-15: Esquema de los procesos en RLHF [65]	25
Figura 2-16: Clasificación de Chatbot Arena mostrada en la presentación de Grok 3 [75]	27
Figura 2-17: Estrategias de evaluación con LLM-as-a-judge [76]	27
Figura 2-18: Comparación de GPT-4o y GPT-o1 en benchmarks [45]	28
Figura 2-19: Comparación de GPT-5 y GPT-4o en benchmarks [79]	29
Figura 2-20: Comparación de Llama 3.1 405B con otros LLMs en benchmarks [82]	30
Figura 2-21: Arquitectura simplificada de Llama 3.1 [86]	31
Figura 2-22: Cuantización FP8 en Llama 3.1 con escalado a nivel de tensor o a nivel de fila [86]	32
Figura 2-23: Composición del dataset de fine-tuning supervisado de Llama 3.1 [86]	33
Figura 2-24: Esquema de fine-tuning realizado en Llama 3.1 [86]	33
Figura 2-25: Hiperparámetros de cada una de las versiones de Llama 3.1 [86]	34
Figura 2-26: Comparación entre las distintas versiones de Llama 3 y Llama 3.1 [86]	34
Figura 2-27: Interfaz de usuario de NotebookLM [103]	37
Figura 3-1: Arquitectura general del sistema	40
Figura 3-2: Flujo de funcionamiento del sistema RAG	41
Figura 3-3: Tamaño en disco de los modelos descargados de Hugging Face	42
Figura 3-4: Uso de memoria de la GPU del modelo Llama 3.1 durante la inferencia	43

Figura 3-5: Respuesta del modelo ante un saludo	49
Figura 3-6: Respuesta del modelo ante una pregunta personal	50
Figura 3-7: Explicación del modelo a conceptos técnicos	50
Figura 3-8: Respuesta del modelo en español	51
Figura 3-9: Comparación técnica realizada por el modelo	52
Figura 3-10: Resolución de problemas sencillos por parte del modelo	53
Figura 3-11: Conceptos avanzados explicados por el modelo	54
Figura 3-12: Respuesta incorrecta proporcionada por el modelo	55
Figura 3-13: Alucinaciones por parte del modelo	56
Figura 3-14: Documentos PDF utilizados en el sistema RAG	57
Figura 3-15: Segmentación de documentos en chunks con solapamiento [107]	58
Figura 3-16: Respuesta correcta por parte del modelo gracias al uso de RAG	63
Figura 3-17: Documentación Swagger de la API REST	64
Figura 3-18: Petición GET a /models	65
Figura 3-19: Petición GET a /models/{model_id}	66
Figura 3-20: Petición POST a /chat/completions sin RAG habilitado	67
Figura 3-21: Petición POST a /chat/completions con RAG habilitado	68
Figura 3-22: Petición GET a /docs	69
Figura 3-23: Petición GET a /docs/{filename}	70
Figura 3-24: Petición PUT a /docs	71
Figura 3-25: Petición DELETE a /docs	71
Figura 3-26: Chat con el asistente experto en comunicaciones digitales	73
Figura 3-27: Menú de la interfaz de usuario	74
Figura 3-28: Pregunta al modelo con RAG habilitado	74
Figura 3-29: Respuesta del modelo con RAG habilitado	75
Figura 3-30: Fuentes utilizadas por el modelo en su respuesta	75
Figura 3-31: Descarga de documentos del sistema RAG	76
Figura 3-32: Ventana de contexto en las conversaciones con el modelo de lenguaje	76
Figura 3-33: Nueva conversación con el modelo	77
Figura 3-34: Edición del system prompt	77
Figura 3-35: Respuesta del modelo tras cambiar el system prompt	78
Figura 3-36: Subida de documentos al sistema RAG	78
Figura 3-37: Pregunta sobre un nuevo documento subido al sistema RAG. Parte 1	79
Figura 3-38: Pregunta sobre un nuevo documento subido al sistema RAG. Parte 2	79
Figura 3-39: Documentos subidos al sistema RAG	80
Figura 3-40: Borrar documentos del sistema RAG	80
Figura 4-1: Distribución del dataset de validación por área de conocimiento	82
Figura 4-2: Fichero CSV del dataset de validación	83
Figura 4-3: Precisión del asistente en cada uno de los bloques temáticos del dataset de validación	84

Figura 5-1: Token de acceso para descargar modelos de Hugging Face	89
Figura 5-2: Tamaño en disco de la imagen Docker desarrollada	89
Figura 5-3: Repositorio Docker Hub de la imagen desarrollada	90
Figura 5-4: Ejecución del backend con docker compose	92
Figura 5-5: Ejecución y parada del backend con docker compose	92
Figura 5-6: Ejecución del frontend con Node	93
Figura 5-7: Frontend compilado con Node	94



# Glosario

---

LLM	Large Language Model
PEFT	Parameter-Efficient Fine-Tuning
GPU	Graphics Processing Unit
TPU	Tensor Processing Unit
IA	Inteligencia Artificial
NLP	Natural Language Processing
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
FFN	Feed-Forward Network
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
REST	Representational State Transfer
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JSON	JavaScript Object Notation
CSV	Comma Separated Values
PDF	Portable Document Format
GPT	Generative Pre-trained Transformer
MMLU	Massive Multitask Language Understanding
MMMU	Massive Multi-discipline Multimodal Understanding
GSM-8K	Grade School Math 8K
MGSM	Multilingual Grade School Math
MATH	Math Word Problem Solving
AIME	American Invitational Mathematics Examination
GPQA	Graduate-Level Google-Proof Q&A
ITU	International Telecommunication Union
SNR	Signal-to-Noise Ratio
PSK	Phase Shift Keying
BPSK	Binary Phase Shift Keying
QPSK	Quadrature Phase Shift Keying
QAM	Quadrature Amplitude Modulation
FSK	Frequency Shift Keying
AWGN	Additive White Gaussian Noise
WSS	Wide Sense Stationary
WSSUS	Wide Sense Stationary Uncorrelated Scattering
SER	Symbol Error Rate
BER	Bit Error Rate
ISI	Inter-Symbol Interference

OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
DS-SS	Direct Sequence Spread Spectrum
CDMA	Code Division Multiple Access
TDM	Time Division Multiplexing
FDM	Frequency Division Multiplexing
CDM	Code Division Multiplexing
SDM	Spatial Division Multiplexing
SIMO	Single Input Multiple Output
MISO	Multiple Input Single Output
MIMO	Multiple Input Multiple Output
TRL	Transformer Reinforcement Learning
SFT	Supervised Fine-Tuning
LoRA	Low-Rank Adaptation
QLoRA	Quantized Low-Rank Adaptation
RAG	Retrieval Augmented Generation
RLHF	Reinforcement Learning from Human Feedback
GAN	Generative Adversarial Network
VAE	Variational Autoencoder
XML	Extensible Markup Language
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CBOW	Continuous Bag of Words
ANN	Approximate Nearest Neighbors
KNN	K-Nearest Neighbors
MMR	Maximum Marginal Relevance
PPO	Proximal Policy Optimization
DPO	Direct Preference Optimization
BLEU	Bilingual Evaluation Understudy
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
MoE	Mixture-of-Experts
GQA	Grouped-Query Attention
FP8	Floating Point 8-bit
BF16	Brain Floating Point 16-bit
RoPE	Rotary Position Embedding
VRAM	Video Random Access Memory
WSL	Windows Subsystem for Linux
CUDA	Compute Unified Device Architecture
PN	Pseudorandom Noise



# 1 INTRODUCCIÓN

---

*Diseñados para imitar a los humanos en todo menos sus emociones. Pero había una posibilidad de que desarrollaran emociones propias.*

*- Blade Runner -*

## 1.1 Motivación

Resultan palpables los grandes avances que se han realizado en los últimos años en inteligencia artificial, y más concretamente en el procesamiento del lenguaje natural. Este progreso se debe en gran medida, a las innovaciones en las arquitecturas neuronales, en el incremento de la capacidad computacional en las GPUs y TPUs, y la disponibilidad de enormes volúmenes de datos. En particular, el surgimiento de los grandes modelos de lenguaje (LLMs) ha revolucionado la forma en que las máquinas procesan, analizan y generan texto, permitiendo aplicaciones que van desde la traducción automática y la síntesis de resúmenes hasta sistemas de diálogo y análisis de sentimientos [1]. Actualmente, modelos como GPT-5, Grok 3, Gemini, Llama 3.1 o DeepSeek R1 han demostrado capacidades extraordinarias para comprender y generar lenguaje humano, superando en muchos casos las expectativas previas sobre lo que las máquinas podrían lograr en tareas lingüísticas complejas.

La creación del modelo Transformer y sus mecanismos de atención han sido hitos fundamentales en este avance. Estos mecanismos facilitan la paralelización en el procesamiento de secuencias y permiten capturar dependencias muy distanciadas entre sí en los datos textuales, lo cual resulta esencial para procesar y analizar contextos complejos. Además, el uso de estrategias de preentrenamiento sobre grandes corpus de datos y el posterior uso de técnicas de ajuste fino (fine-tuning) han demostrado su eficacia para adaptar los modelos a tareas específicas, mejorando notablemente su rendimiento en entornos tanto generales como especializados [2]. Por ejemplo, en el ámbito médico, los LLMs están siendo utilizados para mejorar la precisión en diagnósticos y la personalización de tratamientos; en la educación, potencian sistemas de tutoría inteligente; y en el sector empresarial, optimizan procesos de análisis de datos y atención al cliente. No obstante, junto a estos logros emergen desafíos significativos, tales como la gestión de sesgos, la necesidad de interpretabilidad, el alto coste energético de su entrenamiento y las implicaciones éticas derivadas de su aplicación en contextos críticos [3].

El campo de los LLMs continúa evolucionando a un ritmo vertiginoso, surgiendo cada pocos meses un nuevo modelo que supera a los anteriores en términos de precisión, capacidad, eficiencia y versatilidad. Estos avances no solo se reflejan en mejoras incrementales de rendimiento en benchmarks estándar, sino también en la expansión de las capacidades de los modelos hacia tareas cada vez más complejas, como el razonamiento multimodal, la generación de código y la interacción conversacional avanzada [4].

Además, gracias a la gran competencia que se está produciendo en este ámbito, cada vez más están surgiendo

modelos open source que permiten a empresas y universidades utilizar este tipo de tecnologías vanguardistas sin barreras de coste o licencias restrictivas, pudiendo además crear sus propios modelos a partir de estos. Modelos como Llama, Mistral o Gemma están democratizando el acceso a la inteligencia artificial, abriendo nuevas oportunidades para la innovación y el desarrollo en diversos sectores. Esta tendencia hacia la apertura y la colaboración está transformando el panorama de la IA, permitiendo que organizaciones con recursos limitados participen activamente en esta revolución.

## 1.2 Antecedentes

El procesamiento del lenguaje natural (NLP por sus siglas en inglés) ha recorrido un largo camino desde sus inicios. Fue en la década de 2010 cuando, gracias a los avances en aprendizaje profundo, el uso de las Redes Neuronales Recurrentes (RNN) se popularizó y se consolidó para aplicaciones de NLP. Las RNN, y en particular las redes Long Short-Term Memory (LSTM), permitieron manejar datos secuenciales de manera más efectiva, capturando dependencias entre palabras distanciadas entre sí a lo largo de un texto. Sin embargo, las RNN tenían limitaciones, como la dificultad para procesar secuencias muy largas y su ineficiencia computacional debido al procesamiento secuencial, que dificultaba paralelizar y optimizar cálculos.

### 1.2.1 Modelo Transformer y mecanismos de atención

En el año 2017 se produjo un punto de inflexión con la publicación del artículo “Attention Is All You Need” por Vaswani et al [5]. Este trabajo introdujo el modelo Transformer, que reemplazó la recurrencia de las RNN con mecanismos de atención, específicamente el denominado multi-head attention. Este cambio permitió el procesamiento paralelo de secuencias, mejorando significativamente la eficiencia computacional y la capacidad para capturar dependencias entre palabras en un texto. El bloque básico del modelo Transformer utiliza además conexiones residuales de tipo ResNet [6], así como capas de normalización y de tipo feed-forward (FFN). En este contexto, las capas feed-forward son capas fully-connected que se aplican a cada posición de la secuencia por separado, pero compartiendo los mismos pesos entre todas las posiciones. La arquitectura original de este modelo consta de un codificador y un decodificador, donde el codificador procesa la entrada y el decodificador genera la salida, utilizando capas de atención para ponderar la importancia de las diferentes partes de una secuencia.

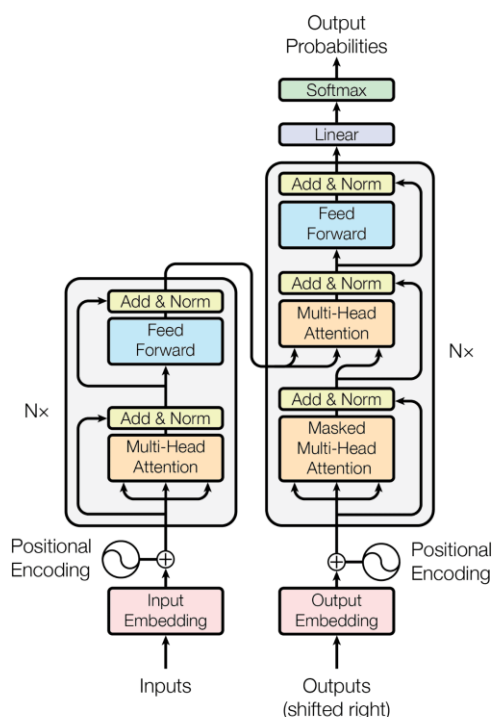


Figura 1-1. Arquitectura del modelo Transformer [5]

Una parte fundamental del modelo Transformer es el mecanismo de atención escalada por producto escalar (scaled dot-product attention), cuya fórmula es:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1-1)$$

Donde:

- $Q$  (queries): La matriz  $Q$  contiene vectores que representan las **consultas**, las cuales buscan información específica dentro de la secuencia de entrada. Cada consulta corresponde a una posición o elemento en la secuencia que necesita información relevante de otras partes de la misma. Las consultas determinan qué información de la secuencia de entrada es relevante para cada posición de salida, permitiendo al modelo enfocarse en las partes más útiles de la entrada.
- $K$  (keys): La matriz  $K$  contiene vectores que representan las **claves**, las cuales indican qué información está disponible en cada posición de la secuencia de entrada. Cada clave corresponde a una posición específica y sirve como una “etiqueta” que describe el contenido de esa posición. Las claves son esenciales porque se comparan con las consultas para determinar la relevancia de cada posición de la secuencia. La similitud entre una consulta y una clave (producto matricial  $QK^T$ ) mide qué tan importante es esa posición para la consulta dada.
- $V$  (values): La matriz  $V$  contiene vectores que representan los **valores**, es decir, la información real que queremos recuperar de la secuencia de entrada. Cada valor está asociado con una clave y, por lo tanto, con una posición específica en la secuencia. Los valores son la salida final del mecanismo de atención. Una vez que determinamos qué posiciones son relevantes (usando consultas y claves), los valores asociados a esas posiciones se combinan para producir la información que el modelo necesita. En esencia, los valores son los “datos” que el modelo utiliza para construir su salida.
- $d_k$ : El escalar  $d_k$  es la dimensión de las claves, usada para escalar el producto  $QK^T$  y evitar valores excesivamente grandes a la entrada de la función softmax.

Podemos visualizar un poco mejor este mecanismo de atención con el siguiente ejemplo:

Consideremos la siguiente frase: “**El gato negro come pescado fresco en la cocina**”. El primer paso consistiría en asignar embeddings, es decir, vectores numéricos con cierto significado semántico, a cada una de las palabras o tokens (unidades básicas de texto, como palabras o partes de palabras). Por ejemplo, las palabras “gato” y “perro”, al tratarse ambas de animales domésticos con características en común, estarían representadas por vectores muy cercanos entre sí dentro del mismo espacio vectorial [7].

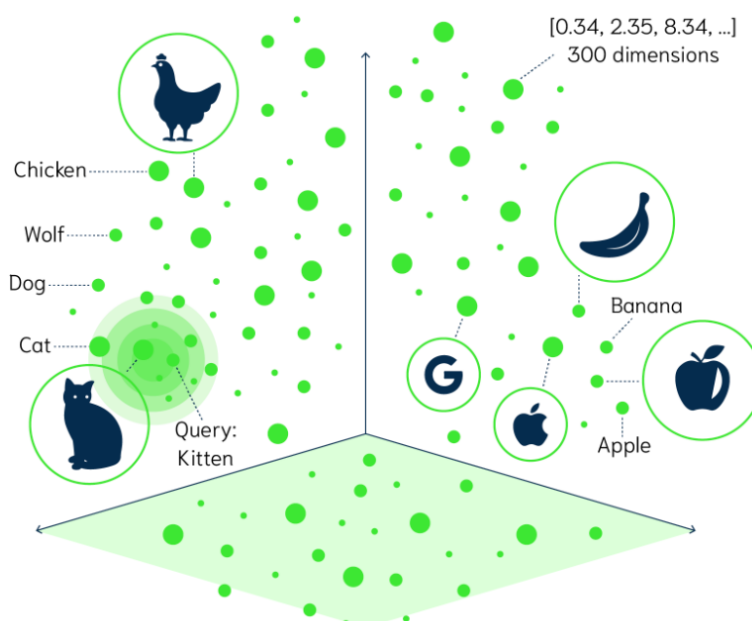


Figura 1-2. Representación gráfica de los embeddings en un espacio vectorial [7]

El siguiente paso consistiría en calcular el producto  $QK^T$ . Para ello, debemos considerar que cada palabra de la frase actúa como una consulta ( $Q$ ), y calcular su similitud con el resto de palabras, que actúan como claves ( $K$ ). Tras aplicar el factor de escalado  $d_k$ , utilizamos los valores resultantes como entrada para una función softmax, obteniendo como resultado los denominados **pesos de atención**. La suma de dichos pesos debe sumar 1, y nos sirven para combinarlos posteriormente con los valores  $V$  de todas las palabras de la frase.

Podemos ver un ejemplo simbólico de los pesos de atención en la siguiente tabla, típicamente denominada matriz de atención, en la cual los colores con tonos verdes indican un valor alto de los pesos de atención, mientras que los colores con tonos rojizos indican un valor bajo:

Queries / Keys	El	gato	negro	come	pescado	fresco	en	la	cocina
El	Green	Light Green	Yellow	Yellow	Yellow	Yellow	Red	Red	Red
gato	Yellow	Green	Light Green	Yellow	Yellow	Yellow	Red	Red	Red
negro	Yellow	Light Green	Green	Yellow	Yellow	Yellow	Red	Red	Red
come	Yellow	Yellow	Yellow	Green	Yellow	Yellow	Red	Red	Yellow
pescado	Red	Yellow	Yellow	Light Green	Green	Yellow	Red	Red	Red
fresco	Red	Red	Yellow	Yellow	Light Green	Green	Red	Red	Red
en	Yellow	Red	Red	Red	Red	Red	Green	Yellow	Yellow
la	Yellow	Red	Red	Red	Red	Red	Yellow	Green	Yellow
cocina	Red	Red	Red	Yellow	Red	Red	Yellow	Yellow	Green

Tabla 1-1. Matriz de atención a la salida de la función softmax del scaled dot-product attention

A partir de esta matriz podríamos sacar las siguientes conclusiones:

- La palabra “come” presta más atención a las palabras “gato” (sujeto), “pescado” (objeto) y “cocina” (lugar).
- La palabra “fresco” se enfoca en “pescado”.
- Las preposiciones “en” y “la” prestan atención a “cocina”.

Tras combinar dichos pesos de atención con los valores  $V$ , podríamos obtener, por ejemplo, un nuevo embedding de la palabra “come” que refleje el contexto de la frase, incluyendo información de “gato”, “pescado” y “cocina”.

El modelo Transformer no solo utiliza un único mecanismo de atención, sino que emplea un enfoque más avanzado denominado multi-head attention (atención multi-cabeza). Este enfoque le permite al modelo capturar diferentes tipos de relaciones y dependencias en los datos al realizar múltiples operaciones de atención en paralelo. Cada cabeza de atención se enfoca en diferentes aspectos de la secuencia, lo que enriquece la representación aprendida.

La fórmula para el multi-head attention es:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (1-2)$$

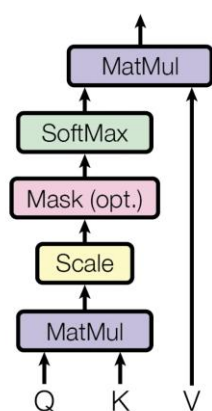
$$\text{donde } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

En la cual:

- $h$ : Es el número de cabezas de atención
- $W_i^Q$ ,  $W_i^K$  y  $W_i^V$ : son matrices de proyección específicas para cada cabeza que transforman las matrices  $Q$ ,  $K$ , y  $V$  en subespacios distintos. Estas proyecciones permiten que cada cabeza de atención se enfoque en diferentes aspectos de la secuencia.
- $W^O$ : Es una matriz de proyección que combina las salidas de todas las cabezas de atención, generando una representación final.

Este mecanismo permite, gracias a las matrices de proyección, que el modelo atienda a diferentes partes de la secuencia de entrada desde múltiples perspectivas simultáneamente, mejorando su capacidad para capturar relaciones complejas y dependencias en los datos. Cada cabeza de atención opera en subespacios distintos, enfocándose en aspectos específicos como relaciones sintácticas o semánticas.

Scaled Dot-Product Attention



Multi-Head Attention

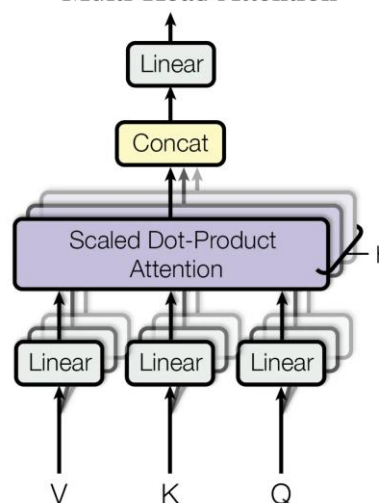


Figura 1-3. Arquitectura de los mecanismos de atención [5]

Por otro lado, es importante destacar que el mecanismo de atención que se suele utilizar en los Transformer es un caso particular conocido como self-attention, en el que las matrices  $Q$ ,  $K$  y  $V$  provienen de la misma secuencia de entrada, permitiendo que cada elemento de la secuencia preste atención a todos los demás elementos dentro de esa misma secuencia. Justamente, la Tabla 1-1 ilustra un ejemplo de self-attention.

Además, otra variante muy extendida es el masked multi-head attention, que se utiliza principalmente en la parte del decodificador de los modelos Transformer. Consiste fundamentalmente en enmascarar u ocultar los pesos de la parte triangular superior de la matriz de atención. De esta forma, se evita que el modelo obtenga información "futura" de la secuencia, asegurando que cada posición sólo pueda atender a las posiciones anteriores y a sí misma. Gracias a esto, se pueden entrenar modelos para tareas de generación y completado de texto garantizando la causalidad de las secuencias generadas.

La arquitectura Transformer, y en particular, los mecanismos de atención, están presentes en la mayoría de los grandes modelos de lenguaje actuales, constituyendo la base sobre la cual se han construido sistemas capaces de comprender y generar lenguaje humano con una precisión sin precedentes.

### 1.3 Objetivos

Una vez presentada la motivación del proyecto y algunos antecedentes de especial relevancia, es importante destacar los objetivos que se quieren alcanzar en este.

El objetivo principal del proyecto consiste en adaptar un modelo de lenguaje para que funcione como un tutor especializado en el área de los sistemas de comunicaciones digitales. Concretamente, se plantea implementar un sistema RAG (Retrieval Augmented Generation) basado en un modelo open source ya preentrenado para que sea capaz de contestar a las preguntas más relevantes de este campo de estudio. Para ello, se escogerá a Llama 3.1 como modelo base, debido a que se trata de un modelo cuyo código y pesos entrenados están disponibles públicamente en plataformas como Hugging Face [8]. Además, este modelo desarrollado por Meta ha demostrado tener resultados comparables con GPT-4o y Claude 3.5 Sonnet en benchmarks estándar como MMLU, HumanEval y GSM-8K, siendo competitivo en tareas de comprensión del lenguaje, matemáticas y generación de código [9].

Para lograr este propósito con éxito, debemos definir una serie de tareas y objetivos específicos que debemos alcanzar:

- **Creación de un dataset de preguntas y respuestas:** Para este fin, se recopilará documentación técnica y académica procedente de libros y artículos científicos, y en particular, se utilizarán los apuntes de clase de la asignatura “Sistemas de Comunicaciones” del Máster en Ingeniería de Telecomunicación de la Universidad de Sevilla, impartida por el profesor y catedrático Juan José Murillo Fuentes [10]. A partir de dicha información, se creará un conjunto amplio de preguntas y respuestas de distintos niveles de dificultad y complejidad, estructurándolas en un formato de datos estándar, como es el caso de CSV. Dicho dataset de preguntas y respuestas cubrirá de forma equilibrada las siguientes temáticas:
  - Fundamentos de los sistemas de comunicaciones digitales. Codificación de fuente y codificación de canal. Encriptación básica. Multiplexación. Modulación y demodulación. Comparación entre modulaciones (PSK, QAM, FSK, etc). Acceso múltiple. Breves aspectos sobre el espectro radioeléctrico, organismos de estandarización y regulación. Ejemplos de sistemas de comunicaciones actuales.
  - Modulaciones digitales y canal AWGN. Constelaciones. Receptor vectorial. Modelo de canal paso de baja discreto equivalente. Detección en canales AWGN. Canal AWGN complejo. Comparación de modulaciones en términos de SER y BER.
  - Canales selectivos en frecuencia. Interferencia entre símbolos (ISI). OFDM. OFDMA. DS-SS. Receptor Rake. CDMA.
  - Modelos de canal. Canal electromagnético y fórmula de Friis. Canal LTI y LTV. Multitrayectos. Canal Doppler. Canal estocástico. Canal Rayleigh y Rician. Desvanecimiento rápido y lento. Canales no AWGN y no lineales. Canal MIMO.
  - Diversidad en sistemas de comunicaciones. Detección en canal Rayleigh. Detección coherente y no coherente. Diversidad temporal. Diversidad espacial. SIMO, MISO y MIMO. Código de Alamouti.
- **Implementación del sistema RAG:** Para dotar al modelo de capacidad de recuperación de información relevante, se implementará un sistema RAG basado en LangChain [11] y ChromaDB [12]. Para ello, se seleccionará una serie de documentos académicos, que serán segmentados y vectorizados mediante el modelo de embedding all-mpnet-base-v2 [13], y posteriormente almacenados en una base de datos vectorial. A partir de una consulta o pregunta del usuario, se recuperarán los fragmentos más relevantes mediante búsqueda semántica, y se integrarán dinámicamente en el prompt del modelo Llama 3.1. Este modelo será ejecutado localmente con cuantización en 8 bits utilizando la librería Transformers de Hugging Face, lo cual permitirá reducir los tiempos de inferencia manteniendo una buena calidad en las respuestas generadas. Para la implementación de este sistema, se emplearán las siguientes librerías y herramientas:
  - LangChain: Este framework está orientado a facilitar la creación de aplicaciones complejas basadas en modelos de lenguaje. Permite integrar fácilmente componentes de recuperación de información, como bases de datos vectoriales, y de generación de texto, definiendo flujos

modulares y reutilizables.

- ChromaDB: Se trata de una base de datos vectorial diseñada para el almacenamiento eficiente y la consulta rápida de embeddings generados a partir de texto u otro tipo de datos. Ofrece capacidades de persistencia, escalabilidad y búsqueda por similitud utilizando índices especializados que aceleran la recuperación semántica.
- Transformers: Esta librería permite acceder a miles de modelos preentrenados para tareas de distintos ámbitos, como el procesamiento de lenguaje natural, visión artificial o procesamiento de audio. Está integrada con librerías de aprendizaje profundo como Jax, PyTorch y TensorFlow [14], y es mantenida por la comunidad de Hugging Face. En este proyecto, se utilizará para cargar y ejecutar Llama 3.1 con técnicas de cuantización en 8 bits, que reducen el consumo de memoria y el tiempo de inferencia sin sacrificar en exceso las capacidades del modelo, lo cual resulta clave en caso de no disponer de un hardware especialmente potente.
- **Validación del sistema:** La validación del sistema se realizará empleando el dataset de preguntas y respuestas descrito previamente. En cada pregunta, se verificará si la respuesta generada por el modelo coincide con la solución esperada, asignando una calificación binaria: 1 si es correcta y 0 si es incorrecta. Para medir de manera objetiva el desempeño del sistema, se empleará como métrica la precisión (accuracy), obteniendo tanto un resultado global como resultados específicos para cada temática incluida en el dataset.
- **Creación de una interfaz de usuario:** Se implementará una interfaz web utilizando React [15], similar a la que ofrecen los LLMs más populares, como por ejemplo ChatGPT . Esta interfaz permitirá que los usuarios hagan preguntas al modelo de lenguaje, manteniendo una conversación con el mismo. Para ello, es necesario mantener una ventana de contexto para que el modelo pueda seguir el hilo de la conversación y generar respuestas coherentes basadas en los mensajes anteriores. Además, se incorporarán diversas funcionalidades, como la posibilidad de eliminar el historial de mensajes, modificar las instrucciones del prompt del LLM, y gestionar los documentos del sistema RAG, incluyendo su carga y eliminación.
- **Implementación de una API REST como backend:** Se desarrollará una API REST utilizando FastAPI [16] que servirá de puente entre el modelo de lenguaje y la interfaz de usuario. Esta API permitirá enviar consultas al modelo con o sin recuperación de información (RAG), y administrar los documentos cargados en el sistema. Para ello, se incluirán endpoints para realizar inferencias con el modelo (con y sin contexto) y para subir, listar, obtener y eliminar archivos PDF del sistema RAG, facilitando así la integración completa con la interfaz web.

Durante la planificación del proyecto se consideraron dos opciones principales: realizar un fine-tuning del modelo Llama o bien implementar un sistema RAG. El motivo por el cual se decidió elegir esta segunda opción es que el fine-tuning exige disponer de un dataset de entrenamiento amplio y de alta calidad, mayor capacidad computacional y largos tiempos de entrenamiento. Además, cada vez que se quisiera añadir nuevo conocimiento al sistema, sería necesario reentrenar el modelo. En cambio, RAG permite disponer de una base de conocimiento dinámica que se puede actualizar fácilmente incorporando o sustituyendo documentos en una base de datos, sin tener que llegar a modificar los pesos del modelo.

Finalmente, cabe destacar que durante el desarrollo de este proyecto se ha recurrido al uso de herramientas de inteligencia artificial generativa, principalmente ChatGPT, como apoyo en distintas fases del trabajo. Estas herramientas han permitido agilizar la estructuración de la memoria, asistir en la generación y verificación de fragmentos de código, facilitar la búsqueda y análisis de información en fuentes contrastadas y contribuir en la formulación de preguntas para el dataset de validación. Todos los resultados obtenidos mediante este recurso fueron revisados y validados de manera crítica, garantizando su coherencia, fiabilidad y adecuación a los objetivos planteados en el proyecto.



# 2 IA GENERATIVA Y LLMs

---

*Antes de trabajar en inteligencia artificial, ¿por qué no hacemos algo sobre la estupidez natural?*

*- Steve Polyak -*

En este capítulo, haremos una introducción a la IA generativa, centrándonos especialmente en los grandes modelos de lenguaje o LLMs. En concreto, abordaremos los fundamentos de la IA generativa, y exploraremos las diversas modalidades que la componen. Con respecto a los LLMs, examinaremos sus principios básicos y algunas técnicas avanzadas que nos permiten optimizar sus resultados, tales como prompt engineering, Retrieval Augmented Generation (RAG), fine-tuning y Reinforcement Learning from Human Feedback (RLHF). A continuación, analizaremos cómo se evalúa y se compara el rendimiento de los LLMs, presentando cuáles son los modelos más relevantes en la actualidad. Posteriormente, profundizaremos en el modelo Llama 3.1, que es el modelo que utilizaremos para crear un tutor experto en el ámbito de los sistemas de comunicaciones digitales. Finalmente, presentaremos NotebookLM, una herramienta de Google basada en el modelo Gemini que aplica técnicas RAG para crear asistentes conversacionales a partir de documentos proporcionados por el usuario.

## 2.1 IA generativa

### 2.1.1 Fundamentos

La IA generativa es un subconjunto de la inteligencia artificial que se centra en la creación de contenido nuevo y original, ya sea en forma de texto, imagen, audio o vídeo, mediante el entrenamiento a partir de grandes volúmenes de datos. A diferencia de otros campos de la IA que se enfocan en tareas de clasificación o predicción, la IA generativa busca aprender la estructura subyacente de los datos para generar nuevas muestras que sean indistinguibles de los datos reales. Esto la convierte en una herramienta poderosa para aplicaciones creativas, como la generación de arte o música, y prácticas, como la creación de datos sintéticos en campos como la medicina o la ingeniería. Los modelos generativos intentan aprender la distribución de probabilidad conjunta de los datos, permitiendo, por ejemplo, no únicamente clasificar un nuevo dato de entrada y asignarle una etiqueta, sino también generar nuevos ejemplos o muestras en el conjunto de datos [17]. Los modelos generativos se diferencian notablemente de los modelos discriminativos en esta nueva faceta, ya que los modelos discriminativos buscan aprender la distribución de probabilidad condicional de las etiquetas en función de los datos de entrada, haciéndolos útiles para tareas de clasificación y regresión.

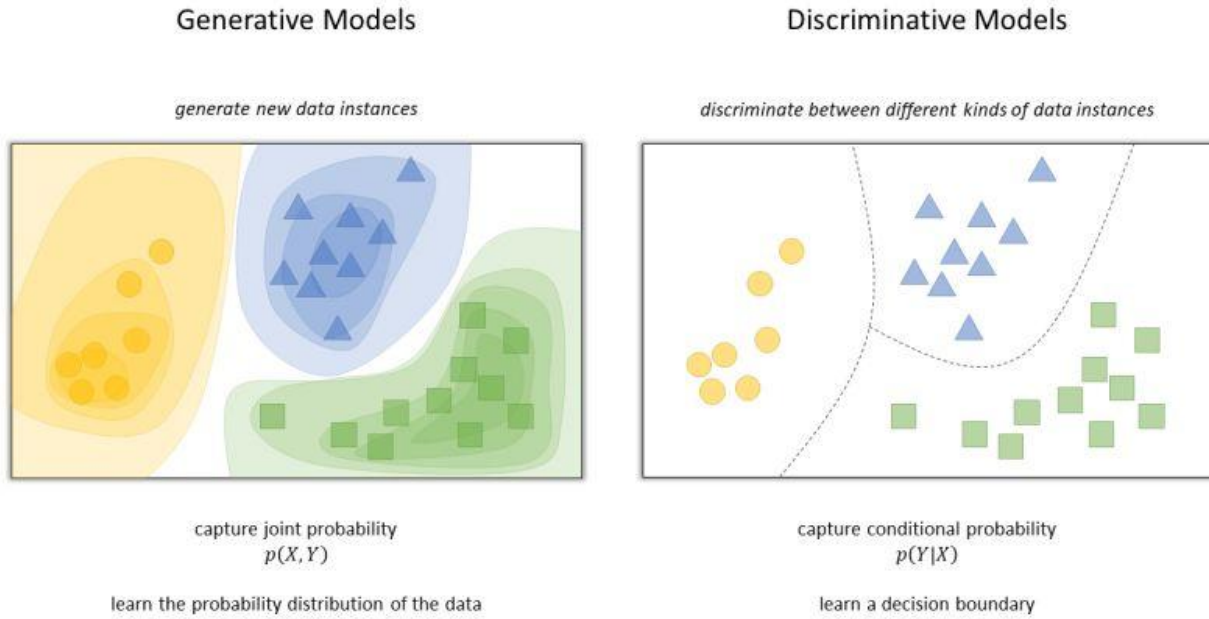


Figura 2-1: Comparación entre los modelos generativos y discriminativos [18]

Uno de los primeros modelos generativos que surgieron fueron los Variational Autoencoders (VAEs). Este modelo fue propuesto por D. P. Kingma et al en el artículo “Auto-Encoding Variational Bayes” [19], publicado en 2013. Un VAE se compone de dos redes neuronales: una red codificadora (encoder), que transforma los datos de entrada en los parámetros (por lo general, la media y la varianza) de una distribución en un espacio latente de menor dimensión, y una red decodificadora (decoder), que reconstruye los datos originales a partir de muestras extraídas de dicha distribución. Los VAEs tienen un gran número de aplicaciones y se utilizan, por ejemplo, para aumentar la resolución de imágenes médicas como tomografías o resonancias magnéticas, o para generar estructuras moleculares con propiedades deseadas, donde destacan en ese ámbito los Molecular Autoencoders [20].

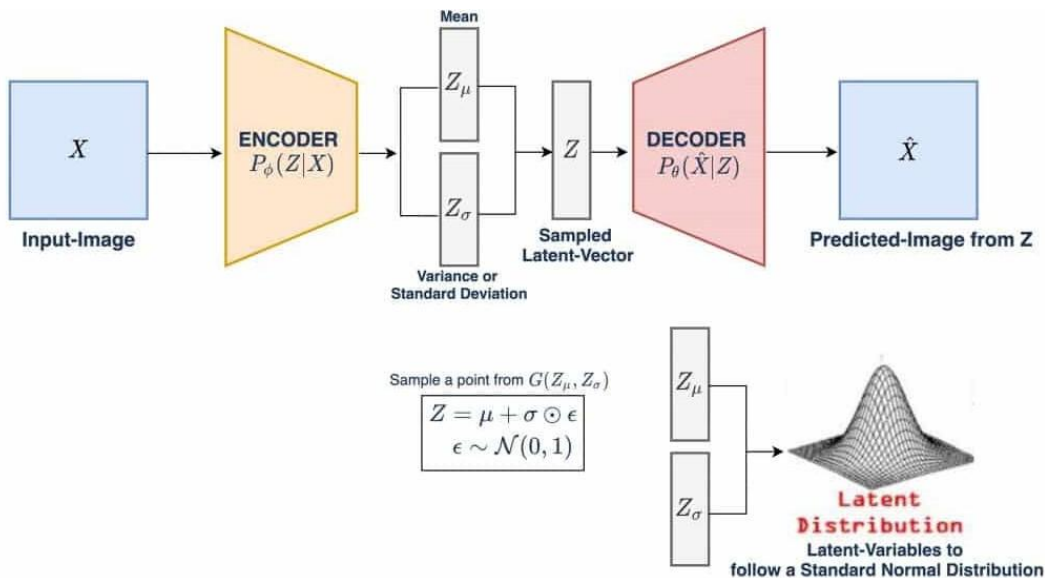


Figura 2-2: Arquitectura de un VAE [21]

Además de los VAEs, otro enfoque importante en la IA generativa son las Generative Adversarial Networks (GANs), presentadas en el artículo “Generative Adversarial Nets” [22] por I. J. Goodfellow et al. Las GANs están compuestas por dos redes neuronales que compiten entre sí: un generador y un discriminador. El generador crea nuevos datos que se asemejan a los datos reales, mientras que el discriminador intenta distinguir entre datos reales o ficticios. Las dos redes se entrenan simultáneamente buscando optimizar una función de pérdidas de

tipo MinMax, en la cual el generador intenta maximizar la probabilidad de error de clasificación del discriminador, mientras que este intenta minimizarla. Las GANs destacan en la generación de imágenes de alta calidad, con aplicaciones como la creación de caras realistas, destacando, por ejemplo, el modelo StyleGAN [23], o la transferencia de estilo, que se basa en aprender la estructura y aspectos visuales de una imagen como colores, texturas y patrones para posteriormente aplicarlos a otra imagen para que adopte los rasgos artísticos aprendidos.

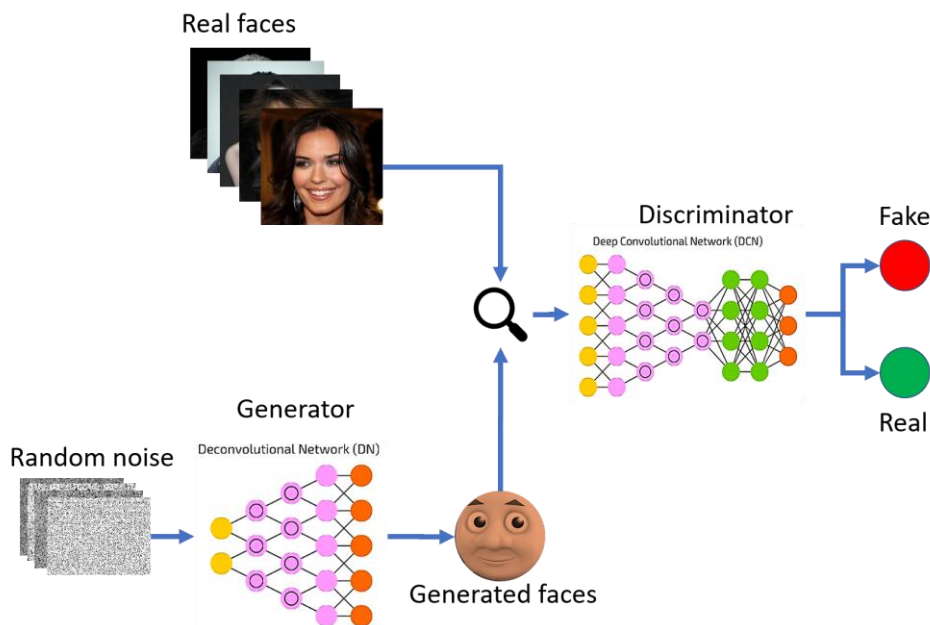


Figura 2-3: Arquitectura de una GAN [24]

Por otro lado, en el artículo “Denoising Diffusion Probabilistic Models” [25], publicado en 2020 por J. Ho et al, se introdujeron los modelos de difusión, usados especialmente para la generación de imágenes. Estos modelos se basan en un proceso de dos etapas. La primera etapa consiste en un proceso de difusión directa que se encarga de añadir ruido gaussiano de manera progresiva a los datos originales, hasta que los datos se transforman completamente en ruido. Este proceso se modela como una cadena de Markov, donde cada paso depende únicamente del anterior. En la segunda etapa, el modelo aprende a revertir este proceso de difusión, es decir, a eliminar el ruido de forma incremental para reconstruir los datos originales a partir de ruido blanco. Para ello, se entrena una red neuronal que predice el ruido añadido en cada paso, permitiendo así generar nuevas muestras al aplicar el proceso inverso a partir de una muestra de ruido aleatorio. Los modelos de difusión han demostrado ser particularmente efectivos en la generación de imágenes de alta calidad, superando en algunos casos a las GANs en términos de fidelidad y diversidad de las muestras generadas [26]. No obstante, suelen ser computacionalmente más costosos debido al proceso iterativo de eliminación de ruido. Entre sus aplicaciones más destacadas se encuentran la generación de imágenes a partir de texto, destacando los modelos DALL-E [27], que se caracterizan por ser más rápidos en la generación al reducir el número de pasos en el proceso de eliminación de ruido, pero sacrificando la calidad de las imágenes finales.

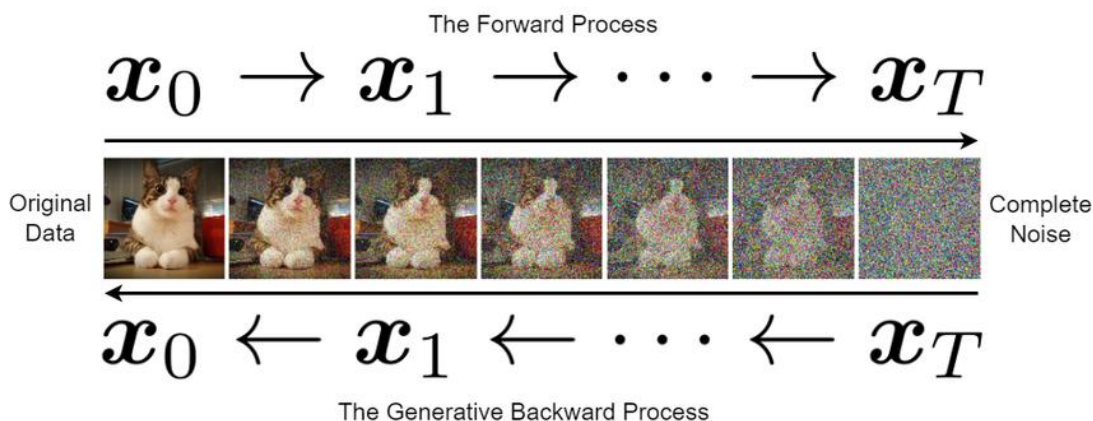


Figura 2-4: Procesos de difusión y eliminación de ruido en los modelos de difusión [28]

## 2.1.2 Modalidades de IA generativa

Una faceta que caracteriza especialmente a la IA generativa es su habilidad para trabajar con diferentes tipos de datos, permitiendo generar contenido nuevo en múltiples formatos. Estas capacidades se manifiestan a través de diversas modalidades, como texto, imagen, audio, video e incluso combinaciones multimodales. A continuación, exploraremos brevemente estas modalidades.

### 2.1.2.1 Texto

La generación de texto es una de las aplicaciones más destacadas de la IA generativa, impulsada principalmente por el desarrollo de los LLMs. Estos modelos, entrenados con enormes cantidades de texto obtenidos de Internet, son capaces de predecir y generar las palabras más probables en función de un texto dado como entrada. Un ejemplo sobresaliente son los modelos GPT (Generative Pre-trained Transformer) [29], desarrollados por OpenAI, que, tras un proceso de fine-tuning, son capaces de realizar tareas tan complejas como la redacción de artículos, responder a preguntas concretas, e incluso generar código en distintos lenguajes de programación. Como su nombre indica, esta clase de modelos se basa en la arquitectura Transformer, pero utilizando solo la parte del decodificador, enfoque que muchos otros LLMs han adoptado. Las aplicaciones basadas en la generación de texto con IA generativa son sin duda las más avanzadas y útiles, incluyendo desde chatbots inteligentes y asistentes virtuales hasta herramientas de traducción automática y generación de contenido creativo. En el apartado 2.2 profundizaremos aún más sobre los LLMs.

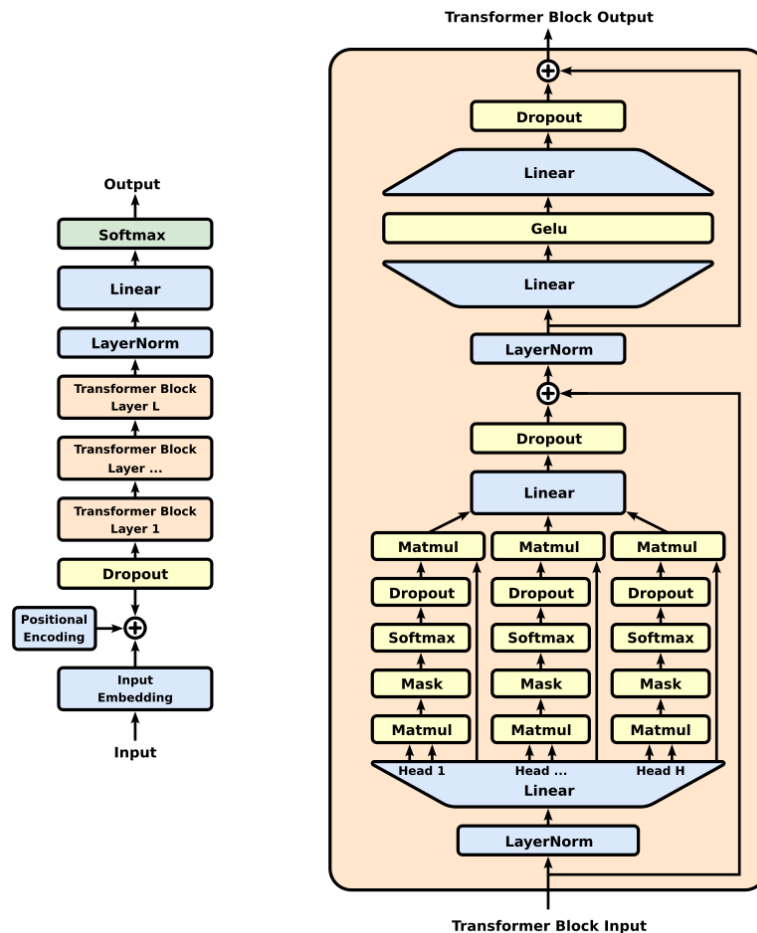


Figura 2-5: Arquitectura GPT original [30]

### 2.1.2.2 Imagen

En el ámbito de las imágenes, se han alcanzado resultados impresionantes gracias a modelos como las GANs y los modelos de difusión, ya mencionados anteriormente. Las GANs son capaces de generar imágenes realistas con un alto grado de control sobre características específicas, como el estilo o la estructura de la imagen. Un ejemplo destacable en este ámbito es pix2pix [31], que permite la traducción de imágenes de un dominio a otro,

como convertir bocetos en fotografías realistas o transformar imágenes diurnas en nocturnas. Por otro lado, los modelos de difusión han emergido como una alternativa poderosa, especialmente en la generación de imágenes a partir de descripciones textuales. Stable Diffusion [32], por ejemplo, utiliza un proceso de eliminación iterativa de ruido en un espacio latente comprimido, lo que permite generar imágenes de alta resolución con un consumo computacional significativamente menor que otros modelos. Además, estos modelos se han empleado en tareas como la superresolución de imágenes y el “inpainting”, reconstruyendo partes faltantes de una imagen con gran precisión. La generación de imágenes destaca por su capacidad para transformar industrias creativas y técnicas, como el diseño gráfico, el cine, los videojuegos, la medicina o la arquitectura.

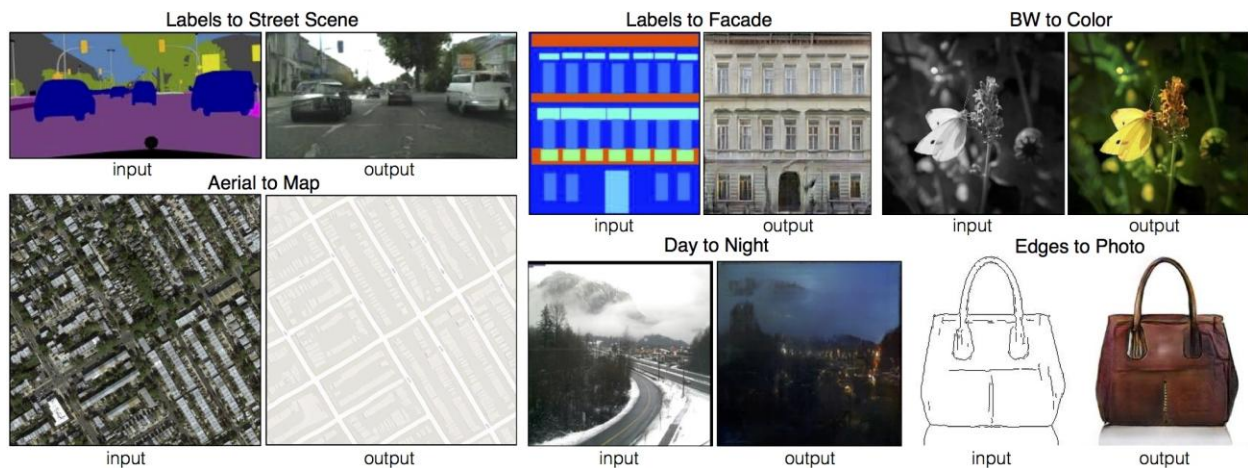


Figura 2-6: Ejemplos de generación de imágenes con pix2pix [31]

### 2.1.2.3 Audio

La modalidad de audio en la IA generativa incluye la creación de sonidos, música y voz sintética. Uno de los modelos pioneros en este campo es WaveNet, desarrollado por DeepMind, que utiliza redes neuronales profundas para generar audio de alta fidelidad, como voces humanas realistas y composiciones musicales [33]. Este modelo se basa en un enfoque autoregresivo que predice muestras de audio de forma secuencial, logrando una calidad que rivaliza con grabaciones reales. Otro avance notable es GANSynth, una aplicación de las GANs que genera muestras de audio de instrumentos musicales, ofreciendo flexibilidad en la síntesis de sonidos [34]. Además, la generación de voz a partir de texto (text-to-speech) ha evolucionado significativamente, permitiendo la creación de voces sintéticas casi indistinguibles de las humanas, con aplicaciones en asistentes virtuales, audiolibros y accesibilidad. La generación de audio demuestra el potencial de la IA para transformar la producción musical y las tecnologías de interactividad.

### 2.1.2.4 Vídeo

La generación de vídeo representa un desafío mayor debido a la necesidad de mantener coherencia temporal y espacial en las secuencias generadas. Sin embargo, modelos como VideoGPT han avanzado en este ámbito utilizando arquitecturas basadas en VAEs y Transformers para generar vídeos a partir de descripciones textuales [35]. Este enfoque permite crear secuencias cortas con narrativa visual, útiles en publicidad y entretenimiento. Otro modelo destacado es MoCoGAN, que emplea GANs para separar el contenido estático y el movimiento dinámico, logrando vídeos coherentes con aplicaciones en la creación de animaciones y efectos visuales [36]. Aunque la generación de vídeo es computacionalmente intensiva y aún enfrenta limitaciones en términos de duración y resolución, sus avances sugieren un futuro prometedor en la producción audiovisual automatizada y la generación de contenido para entrenamiento de modelos de IA.

### 2.1.2.5 Multimodal

La IA generativa multimodal abarca modelos capaces de integrar y generar contenido a partir de múltiples tipos de datos simultáneamente. Un ejemplo sobresaliente es CLIP (Contrastive Language-Image Pre-training), que aprende asociaciones entre imágenes y texto, permitiendo generar imágenes a partir de descripciones textuales o clasificar contenido visual basado en lenguaje natural [37]. Este enfoque ha sentado las bases para sistemas como DALL-E [27], que combinan capacidades multimodales con generación de alta calidad. Otro modelo relevante es MUM (Multitask Unified Model), desarrollado por Google, que integra texto, imágenes y vídeos

en un marco unificado para realizar tareas complejas, como responder preguntas basadas en contenido multimedia [38]. Los modelos multimodales son particularmente útiles en aplicaciones que requieren interacción avanzada entre humanos y máquinas, como asistentes virtuales inteligentes, sistemas de recomendación personalizados y herramientas de creación de contenido interdisciplinar.

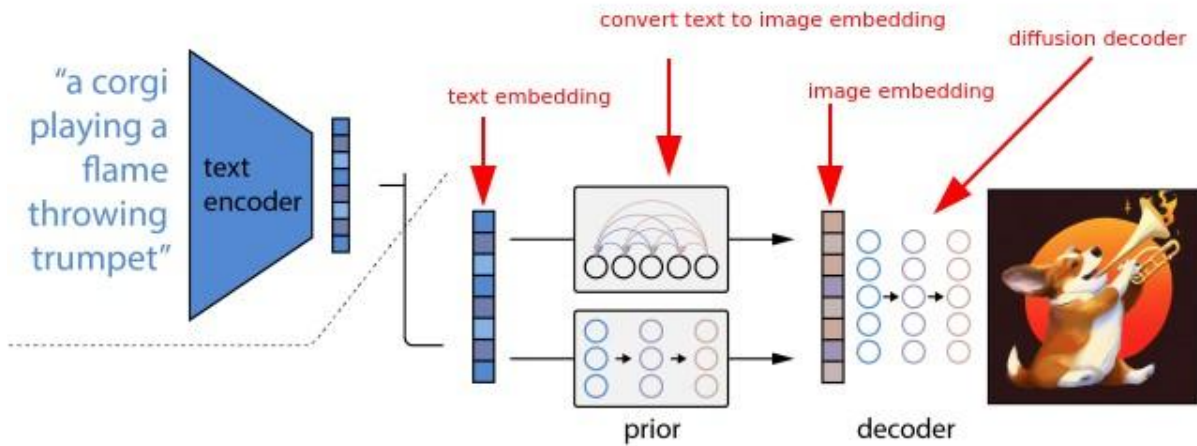


Figura 2-7: Arquitectura simplificada de los modelos DALL-E [39]

## 2.2 Grandes modelos de lenguaje (LLMs)

### 2.2.1 Fundamentos

Los LLMs son un tipo específico de modelos de inteligencia artificial generativa diseñados para comprender, procesar y generar texto. A diferencia de otros modelos generativos, los LLMs se especializan en capturar patrones lingüísticos, semánticos y contextuales a partir de texto. Su capacidad para producir respuestas coherentes, traducir idiomas, resumir textos o incluso escribir código los ha convertido en pilares de aplicaciones avanzadas como chatbots, asistentes virtuales y herramientas de creación de contenido. Estos modelos suelen integrarse en entornos empresariales mediante APIs comerciales (como las de OpenAI, Gemini o Claude) o mediante modelos locales de código abierto (por ejemplo, Llama 3.1 o Mistral), lo que permite su adaptación a flujos de trabajo específicos, como análisis de correos electrónicos, generación de informes o soporte automatizado en centros de atención al cliente. Como ya se mencionó anteriormente, los LLMs modernos tienen una arquitectura de tipo "decoder-only", es decir, utilizan solo la parte del decodificador de un bloque Transformer.

En esencia, los LLMs operan como sistemas de predicción probabilística de texto. Dado un fragmento de texto, comúnmente llamado prompt, el modelo predice la secuencia de palabras más probable que debe seguir, basándose en los patrones aprendidos durante su entrenamiento. Esta predicción no es una mera concatenación de palabras frecuentes, sino un proceso contextualizado: el modelo analiza las relaciones entre los tokens en función de su posición, significado y dependencias gramaticales. Sin embargo, esta naturaleza probabilística conlleva limitaciones críticas. Por ejemplo, los LLMs pueden generar "alucinaciones", es decir, respuestas plausibles pero incorrectas o inventadas, como citas falsas de personas o casos legales inexistentes [40]. Además, su conocimiento está sujeto a lo que se denomina knowledge cutoff, ya que está limitado a la información disponible hasta la fecha de su entrenamiento. Esto implica que modelos entrenados con información publicada en 2023 hacia atrás desconocerán los eventos que se produjeron en fechas posteriores. Para solucionar este problema, es necesario utilizar técnicas avanzadas como RAG o hacer un fine-tuning del modelo con información actualizada.

La forma en la que un modelo de lenguaje adquiere su base de conocimiento se denomina preentrenamiento. Este proceso se realiza mediante aprendizaje auto-supervisado, utilizando un corpus de texto masivo que incluye libros, artículos científicos, páginas web, foros, código fuente, etc. Durante esta etapa, el modelo se expone a

billones de tokens y aprende a predecir la siguiente palabra en una secuencia. Por ejemplo, dado el fragmento “El cielo está...”, el modelo ajusta sus parámetros internos para asignar una alta probabilidad a palabras como “despejado” o “nublado”, basándose en patrones estadísticos observados en los datos. A este tipo de entrenamiento se le llama auto-supervisado porque no es necesario asignarle etiquetas a los datos como suele ocurrir en el aprendizaje supervisado tradicional. Buena parte del éxito de los LLMs se basa en disponer de inmensas cantidades de datos ya etiquetados por naturaleza, mientras que en otras modalidades, como el procesamiento de imágenes, sería necesario invertir grandes cantidades de tiempo en etiquetar los datos. Este preentrenamiento consume enormes recursos computacionales y energéticos, requiriendo disponer de clústeres de GPUs y TPUs de última generación en funcionamiento durante semanas o meses. La función de pérdidas que suele usarse durante el entrenamiento es la entropía cruzada [41], permitiendo medir cuánto se desvía la predicción del texto real y ajustar progresivamente los pesos del modelo mediante el mecanismo de backpropagation, ampliamente usado en redes neuronales.

Como se ha comentado previamente, cuando un LLM genera texto, opera como un sistema probabilístico autoregresivo. Cada token producido depende de toda la secuencia anterior, analizando no solo las frecuencias léxicas, sino las relaciones contextuales. Antes de pasar por la propia red neuronal, el texto de entrada debe someterse a un proceso de tokenización, en el que las palabras se dividen en unidades más pequeñas llamadas tokens. Estos pueden ser palabras completas, subpalabras o incluso caracteres, dependiendo del sistema de tokenización empleado. Típicamente, los LLMs suelen manejar un vocabulario de decenas de miles hasta ciento de miles de tokens en el caso de los modelos multilingüaje [42], a partir del cual construyen el resto del lenguaje. No obstante, los LLMs tienen restricciones prácticas: la mayoría tiene límites estrictos en la cantidad de tokens de entrada y salida (por ejemplo, de 4.096 a 100.000 tokens), lo que dificulta el procesamiento de documentos extensos o conversaciones prolongadas. Una vez realizado el proceso de tokenización, a los tokens se les deben asignar embeddings, que son representaciones vectoriales que capturan su significado. Gracias a esta representación matemática de los tokens, estos pueden ser procesados por la red neuronal para generar las predicciones.

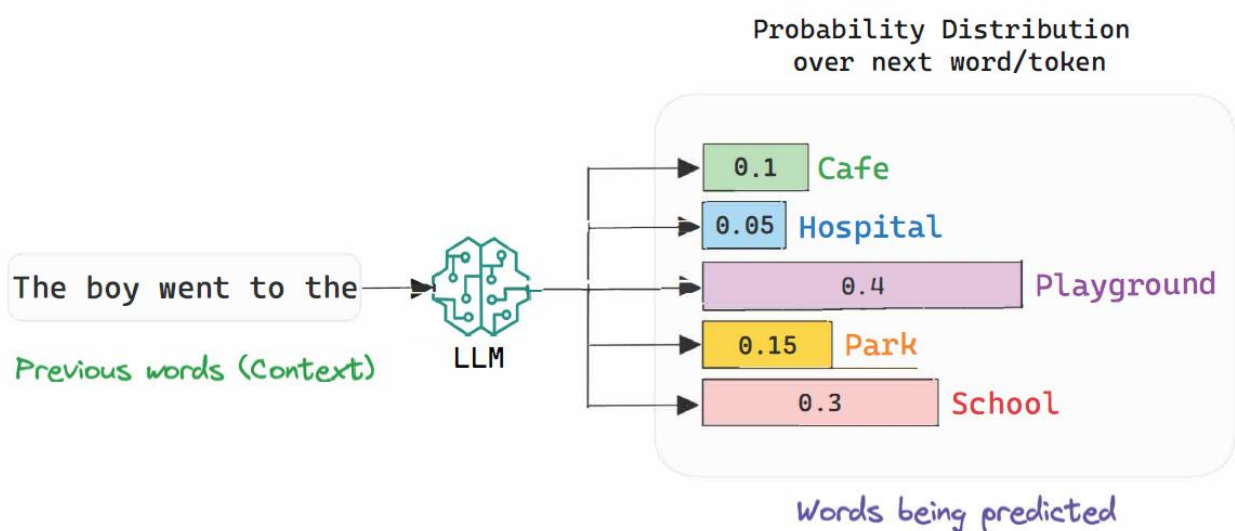


Figura 2-8: Proceso de predicción de tokens en un LLM [43]

Para convertir estas predicciones en tokens concretos, los LLMs emplean mecanismos de muestreo que regulan un equilibrio entre creatividad y coherencia. Uno de los métodos más utilizados es la temperatura, un hiperparámetro que modula la entropía de la distribución de probabilidad. Valores bajos de temperatura (por ejemplo, 0.1) enfatizan los tokens de alta probabilidad, generando texto más determinista, mientras que valores altos de temperatura (como 1.0) provocan que de vez en cuando se escojan tokens de menor probabilidad, permitiendo una mayor diversidad y creatividad [44]. La forma en la que se suele implementar este mecanismo de temperatura es redefiniendo la función softmax de la última capa de los LLMs de la siguiente forma:

$$P(y_i|z, T) = \frac{e^{z_i/T}}{\sum_{j=1}^n e^{z_j/T}} \quad (2-1)$$

Donde:

- $y_i$ : Es el token  $i$ -ésimo en el vocabulario del modelo.
- $z_i$ : Es el logit (valor bruto de salida) correspondiente al token  $y_i$ .
- $T$ : Es el hiperparámetro de temperatura que ajusta la distribución de las probabilidades.
- $n$ : Es el tamaño del vocabulario, es decir, el número total de tokens posibles.

De esta fórmula, en función de los valores de temperatura  $T$  podemos deducir que:

- Valores de  $T$  cercanos o iguales a 1 provocan que la fórmula se reduzca a la softmax estándar, asignando probabilidades directamente basadas en los logits.
- Valores de  $T$  menores a 1 provocan que se amplifiquen las diferencias entre los logits, debido al efecto de la exponencial. Los tokens con valores  $z_i$  altos producirán exponentes mucho mayores y, por ende, tendrán probabilidades mucho más elevadas. Por lo tanto, la distribución se vuelve más concentrada en unos pocos tokens de alta probabilidad, lo que se traduce en un comportamiento más determinista.
- Valores de  $T$  mayores a 1 suavizan las diferencias entre los logits. Dividir  $z_i$  por un valor mayor que 1 reduce el valor del exponente haciendo que incluso los tokens con logits relativamente bajos obtengan valores de  $e^{z_i/T}$  no tan despreciables. Como resultado, la distribución se vuelve más plana y diversa, permitiendo una mayor variabilidad y creatividad en la elección de tokens.

Otro enfoque es el muestreo top\_k, que restringe la selección a los  $k$  tokens con mayor probabilidad, descartando opciones marginales. Este método evita tokens que no tienen sentido en un contexto dado, pero puede limitar la variabilidad en dominios creativos. Una alternativa más dinámica es el muestreo top\_p (o nucleus sampling), que selecciona el conjunto mínimo de tokens cuya probabilidad acumulada supera un umbral, adaptando automáticamente el tamaño del espacio de selección según el contexto. Ambos métodos, top\_k y top\_p, pueden combinarse con la temperatura para ajustar el comportamiento generativo según las necesidades que tengamos.

A pesar de sus capacidades avanzadas, los LLMs presentan varias limitaciones que debemos considerar al integrarlos en aplicaciones prácticas. Una de las más destacadas es su falta de memoria entre interacciones. Debido a la naturaleza de los LLMs, cada prompt se procesa como una nueva conversación, ya que la red neuronal no incorpora ningún mecanismo que permita mantener una memoria de los mensajes intercambiados previamente. Si un usuario menciona un dato personal, como su nombre, en una interacción y luego hace una pregunta que depende de ese dato, el modelo no lo recordará a menos que se le proporcione de nuevo. Este mecanismo de memoria debe implementarse de forma externa a la red neuronal, típicamente concatenando los prompts anteriores y las salidas generadas por el modelo con el nuevo prompt.

Otra limitación significativa es el rendimiento en tareas complejas de razonamiento matemático y lógico. Aunque los LLMs pueden resolver operaciones aritméticas simples o seguir reglas lógicas básicas, su eficacia disminuye en problemas que requieren múltiples pasos de razonamiento o una manipulación simbólica precisa, aunque los modelos más avanzados y vanguardistas que están surgiendo, como GPT-o1, implementan cadenas de razonamiento que permiten abordar estos problemas [45].

Además, los LLMs muestran una fuerte dependencia del contexto proporcionado en el prompt. Si el contexto es ambiguo, incompleto o está mal formulado, el modelo puede generar respuestas incorrectas, irrelevantes o inconsistentes. Esta sensibilidad requiere que los usuarios formulen prompts claros y específicos, y en algunos casos, puede ser necesario implementar un preprocesamiento del texto para mejorar la calidad de los datos de entrada antes de que el modelo los procese. Todas estas técnicas se agrupan en una disciplina llamada prompt engineering, de la cual hablaremos con más detalle.

Por último, los LLMs pueden perpetuar sesgos presentes en los datos de entrenamiento. Dado que estos modelos se entrenan con un corpus masivo extraído de fuentes como Internet, pueden reflejar o amplificar prejuicios relacionados con género, raza, cultura o ideología. Esto puede resultar en respuestas discriminatorias o injustas, un problema particularmente grave en aplicaciones sensibles como la toma de decisiones automatizada o la generación de contenido para audiencias diversas. Mitigar estos sesgos requiere una curación cuidadosa de los datos y el uso de técnicas de debiasing, aunque eliminarlos por completo sigue siendo un desafío.

## 2.2.2 Prompt engineering

El prompt engineering es la disciplina que se enfoca en la creación y optimización de prompts para interactuar de manera efectiva con los LLMs. Su objetivo principal es maximizar la precisión, relevancia y utilidad de las respuestas generadas por el modelo, guiándolo hacia la salida deseada mediante una formulación cuidadosa de las entradas de texto. Esta práctica se ha vuelto esencial debido a la naturaleza probabilística y contextual de los LLMs, que dependen en gran medida del prompt proporcionado.

La claridad en las instrucciones es fundamental para reducir ambigüedades y alinear las expectativas del usuario con las capacidades del modelo. Un prompt bien estructurado especifica la tarea, el contexto y los requisitos de formato de manera explícita. Los LLMs son extremadamente sensibles a la forma en que se plantea una solicitud; un prompt ambiguo o mal estructurado puede derivar en respuestas imprecisas o irrelevantes. Por ello, es crucial que las instrucciones sean específicas y desprovistas de ambigüedad. Por ejemplo, un prompt genérico como “Habla sobre inteligencia artificial” podría generar una respuesta amplia y poco enfocada a lo que deseamos concretamente. En cambio, un prompt bien formulado sería: “Explica en tres párrafos cómo la inteligencia artificial está transformando la medicina, incluyendo ejemplos de aplicaciones en diagnóstico médico”. Por otro lado, el uso de cadenas de razonamiento en los prompts demuestra que usar instrucciones detalladas mejoran significativamente la coherencia y precisión de las respuestas de los modelos [46].

Una forma sencilla de mejorar los prompts es incluir ejemplos de las tareas que queremos que realice el modelo. Esta técnica se denomina few-shot learning [47], y consiste en proporcionar ejemplos que sirvan como patrones de referencia, facilitando que el modelo pueda inferir el formato, estilo o profundidad esperados en la respuesta. Esta técnica es particularmente útil para tareas específicas en las que el modelo no ha sido entrenado explícitamente, pero puede adaptarse gracias al contexto proporcionado. Un ejemplo práctico del few-shot learning es el análisis de sentimientos en un texto. Aunque por defecto un LLM no suele estar entrenado para esta tarea, gracias a unos pocos ejemplos puede entender lo que le estamos solicitando y proporcionar la respuesta en el mismo formato que los ejemplos.

You are an AI assistant that can perform sentiment analysis. Here are some examples:

Input: The online ordering process was seamless, and my package arrived even earlier than expected. Everything was well-packaged, and the product quality was outstanding.

Output: Positive.

Input: I was excited to try this new restaurant, but the food was bland, and the service was painfully slow. The ambiance was nice, but it didn't make up for the overall poor experience.

Output: Negative.

Given these examples, classify the following input:

The customer service was friendly and efficient. I got my issue resolved quickly, and they even followed up to ensure everything was working fine.

Output: Positive.



Figura 2-9: Few-shot learning para el análisis de sentimientos con ChatGPT

Otro aspecto esencial del prompt engineering es la especificación del formato de las respuestas, una práctica clave para garantizar que las salidas del modelo sean directamente útiles en contextos técnicos o aplicados. Los LLMs pueden generar resultados en una amplia variedad de estructuras, como texto plano, listas numeradas, tablas, JSON, XML o incluso formatos personalizados, lo que permite una integración fluida en sistemas

automatizados, aplicaciones o flujos de trabajo. Por ejemplo, en entornos de desarrollo de software, podemos solicitarle la respuesta al modelo en formato JSON, facilitando su integración con otros componentes software y APIs, mientras que una tabla puede ser útil para documentación técnica o informes.

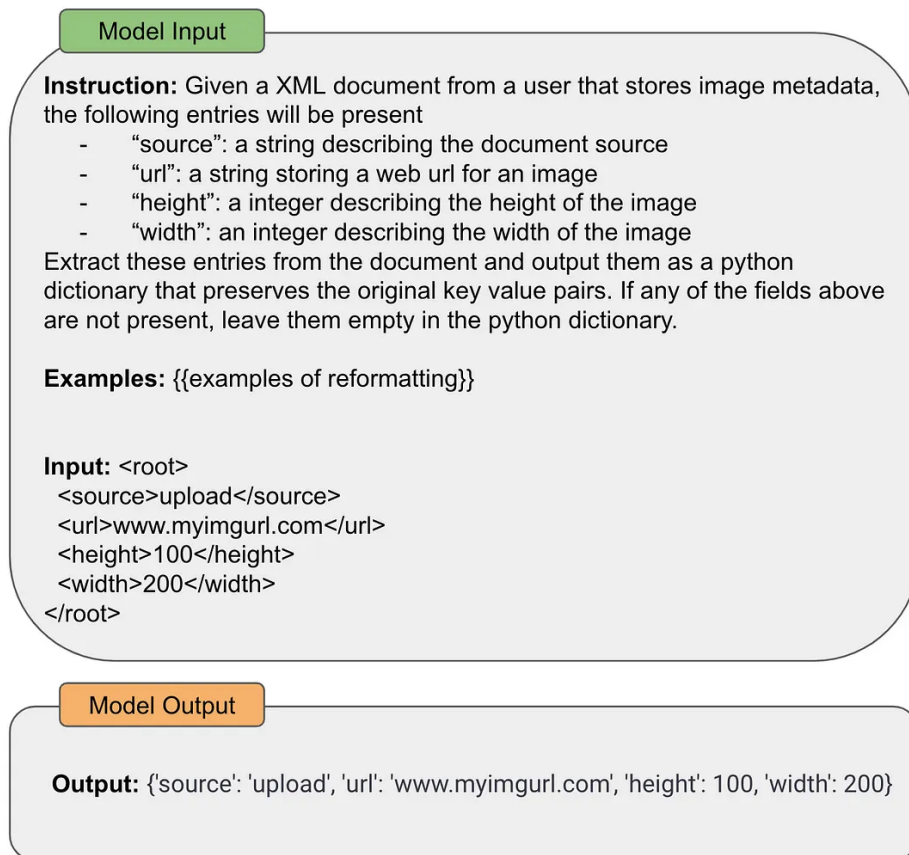


Figura 2-10: Prompt para cambiar el formato de un documento XML a un diccionario Python [48]

Por otro lado, para tareas que consistan en que el modelo redacte un texto, es importante especificar la longitud aproximada de la respuesta que deseamos. Los LLMs pueden generar respuestas que varían desde una sola frase hasta varios párrafos, dependiendo del prompt y de la configuración del modelo. Por lo tanto, la longitud deseada de la respuesta debe establecerse con precisión para evitar resultados demasiado extensos o insuficientes. Para ello, es recomendable incluir en los prompts frases como: “Desarrolla tu respuesta en 300 palabras” o “Explica en detalle el tema solicitado en 5 párrafos”. También es importante el uso de palabras clave como “explicar”, “describir” o “comparar”, y también usar frases como “en términos simples”, “para un público técnico” o “con ejemplos prácticos” para ajustar el tono y la complejidad de la respuesta.

Con frecuencia, los LLMs tienden a generar información incorrecta cuando no conocen la respuesta a una pregunta, fenómeno conocido como alucinaciones. Una forma de mitigar las alucinaciones es pedirle al modelo que admita que no conoce la respuesta, en lugar de generar una respuesta ficticia. De esta forma, podemos evitar que un modelo puesto en producción proporcione respuestas equivocadas y perjudiciales a los usuarios.

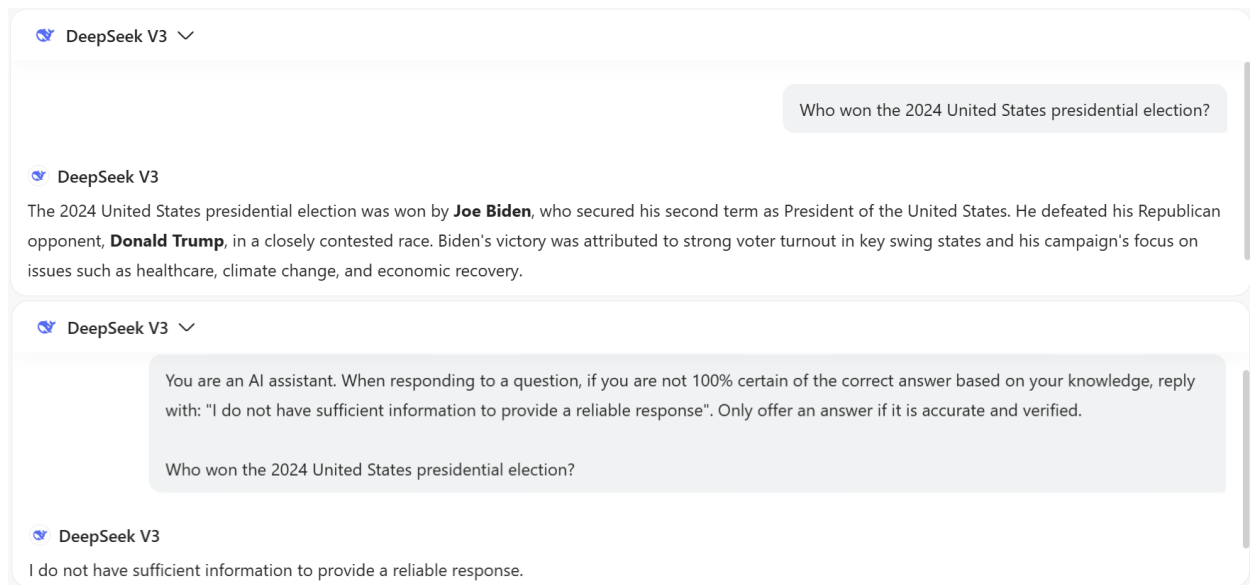


Figura 2-11: Prevención de alucinaciones mediante prompt engineering

Además, los LLMs destacan por su capacidad para trabajar con datos no estructurados, lo que los convierte en herramientas versátiles para una amplia gama de aplicaciones. Los datos no estructurados, como artículos, correos electrónicos, transcripciones o publicaciones en redes sociales, no suelen seguir un formato predefinido, lo que usualmente ha representado un desafío para su procesamiento automatizado. Sin embargo, gracias a su comprensión del lenguaje natural, los LLMs pueden analizar estos datos, identificar patrones y extraer información relevante sin necesidad de una estructura rígida. Entre las aplicaciones más destacadas se encuentra la extracción de información de un texto, donde el modelo puede identificar y extraer datos específicos, como fechas y nombres en un contrato, o términos técnicos en un artículo científico. Otra aplicación es la generación de resúmenes, permitiendo condensar documentos extensos en unas pocas frases que capturen la esencia del texto original, lo que resulta ideal para revisiones rápidas o síntesis de información. Asimismo, en la clasificación de correos electrónicos, los LLMs pueden categorizar mensajes automáticamente en carpetas como “spam”, “promociones” o “importantes”, basándose en el análisis del contenido y el contexto. Estas capacidades, junto con el análisis de sentimientos ya mencionado, convierten a los LLMs en grandes herramientas para procesar grandes volúmenes de datos no estructurados.

El diseño de prompts efectivos rara vez es un proceso lineal, sino que requiere un enfoque iterativo y experimental. El prompt engineering no es un proceso estático, sino que implica un refinamiento continuo basado en la evaluación de las respuestas del modelo. Inicialmente, un prompt puede producir resultados subóptimos debido a ambigüedades, falta de contexto o instrucciones incompletas, por lo que es necesario ajustar los prompts de forma progresiva para obtener los resultados deseados. Este enfoque consiste en comenzar con un prompt inicial, analizar la salida generada, identificar áreas de mejora y modificar las instrucciones en consecuencia. Por ejemplo, si una respuesta es demasiado simple y superficial, se puede añadir más contexto o ejemplos para guiar al modelo; si es excesivamente extensa, se puede solicitar un resumen o establecer una longitud específica. Probar diferentes formulaciones del mismo prompt también permite comparar resultados y seleccionar la opción más efectiva. Este ciclo iterativo es esencial para optimizar la precisión y relevancia de las respuestas, especialmente en tareas complejas o cuando se busca un formato de salida específico, asegurando que el modelo se alinee plenamente con las necesidades del usuario.

### 2.2.3 Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation (RAG) es una técnica avanzada que integra las capacidades de generación de texto de los LLMs con la recuperación de información relevante desde una base de datos externa. Este enfoque híbrido permite a los LLMs superar algunas de sus limitaciones inherentes, como el knowledge cutoff o las alucinaciones [49]. A diferencia del enfoque tradicional donde un LLM genera respuestas únicamente basándose en su conocimiento preentrenado, RAG incorpora un sistema de recuperación para proporcionar contexto adicional, lo que le permite al modelo generar respuestas más precisas y actualizadas.

El funcionamiento de un sistema RAG se inicia cuando un usuario introduce un prompt. En lugar de responder directamente, el sistema primero convierte el prompt en embeddings y busca en una base de datos vectorial fragmentos de información en función de su similitud con el prompt. Estos fragmentos recuperados se añaden como contexto al prompt original, permitiendo que el LLM genere una respuesta más informada y específica. Este método es particularmente útil en aplicaciones donde se requiere información actualizada o especializada que no estaba disponible en el conjunto de datos de entrenamiento, como noticias recientes o documentación técnica propietaria. Además, al reducir la dependencia exclusiva del conocimiento interno del modelo, los sistemas RAG mitigan el riesgo de producir respuestas erróneas o descontextualizadas.

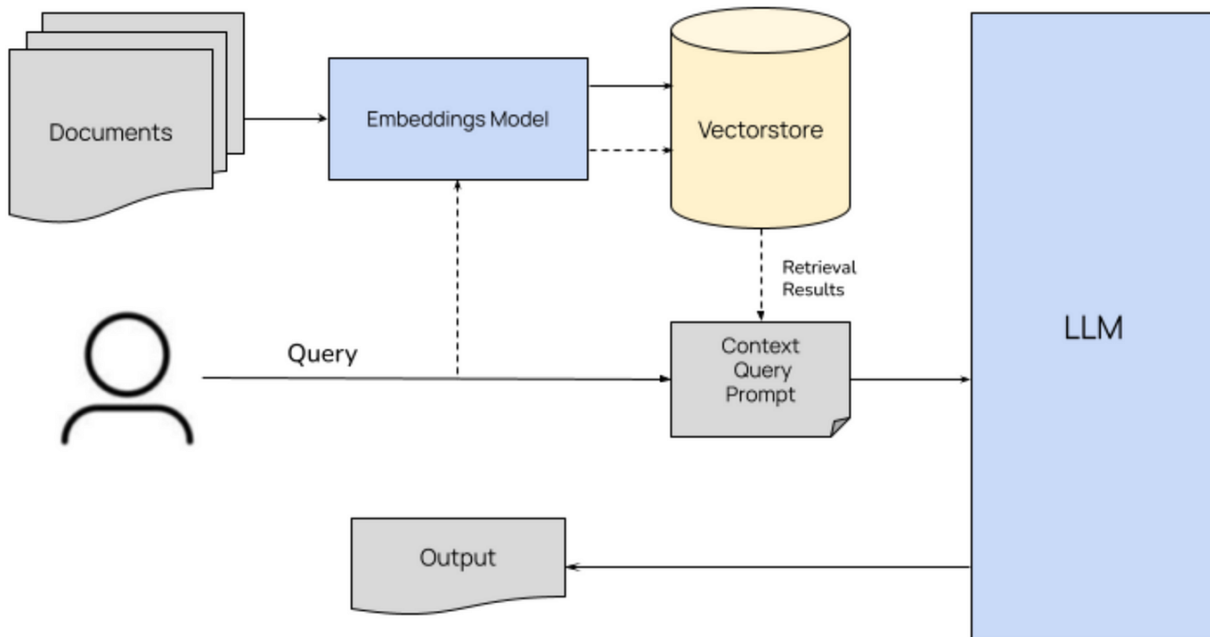


Figura 2-12: Arquitectura de los sistemas RAG [50]

Un componente esencial de RAG es el uso de embeddings. Como habíamos comentado anteriormente, los embeddings son representaciones vectoriales de palabras o tokens que capturan su significado semántico en un espacio multidimensional. Estas representaciones permiten que palabras, frases o documentos completos sean mapeados a vectores numéricos de tal manera que la distancia entre ellos refleje su similitud semántica. Por ejemplo, los vectores correspondientes a “sol” y “luz” estarán más cerca entre sí que los de “sol” y “casa”. En general, cada una de las dimensiones del embedding no tiene significado por sí sola, y dependen del modelo de embedding utilizado, siendo únicamente relevante la distancia que hay entre embeddings. La creación de embeddings se basa en redes neuronales, como Word2Vec [51], GloVe [52] o BERT [53] y sus variantes. Estos modelos se entrenan con objetivos y datos específicos para aprender patrones lingüísticos y relaciones contextuales únicas, generando así espacios vectoriales distintos para cada modelo. En el caso de Word2Vec, se utilizan algoritmos como Skip-Gram o CBOW: el primero predice palabras del contexto a partir de una palabra central, mientras que el segundo predice una palabra central a partir del contexto, consiguiendo así generar vectores que capturan los significados básicos de las palabras. Por otro lado, los modelos basados en Transformers como BERT emplean mecanismos de atención para analizar el contexto completo de una secuencia de texto, produciendo embeddings más complejos y sensibles al contexto. Una vez entrenado, el modelo puede generar embeddings para cualquier texto nuevo, que luego se utilizan en RAG para buscar y recuperar información relevante de manera eficiente.

El siguiente elemento fundamental en RAG son las bases de datos vectoriales. Una base de datos vectorial es un sistema diseñado para almacenar y buscar embeddings de forma eficiente, optimizado para manejar grandes volúmenes de datos en espacios de alta dimensionalidad. A diferencia de las bases de datos relacionales, que operan con datos estructurados como tablas, las bases de datos vectoriales están optimizadas para realizar búsquedas de similitud, identificando vectores cercanos a un vector de consulta en un espacio multidimensional. El almacenamiento de información en una base de datos vectorial sigue un proceso estructurado. Primero, el

texto fuente, que puede provenir de documentos, artículos o cualquier corpus externo, se divide en fragmentos manejables mediante una técnica conocida como chunking. Estos fragmentos pueden ser oraciones, párrafos o secciones, dependiendo de la granularidad requerida para la aplicación, manteniendo un cierto solapamiento entre ellos para preservar el contexto entre fragmentos adyacentes. Cada chunk se transforma entonces en un embedding utilizando el modelo seleccionado, y se almacena en la base de datos. Además, es común asociar metadatos a cada embedding, como la fuente del texto, fecha de creación, etiquetas indicando la sección a la que pertenece el chunk, etc. Estos metadatos enriquecen la información almacenada y permiten filtrar o priorizar resultados durante la búsqueda, mejorando la precisión y utilidad del sistema. Algunos ejemplos de bases de datos vectoriales son ChromaDB [12], Pinecone [54] y Weaviate [55], ampliamente utilizadas en aplicaciones de RAG.

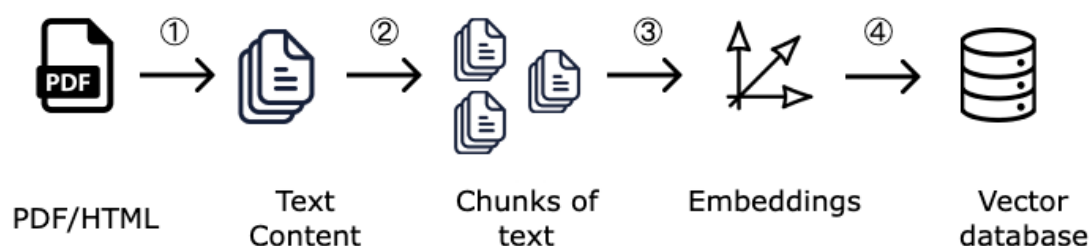


Figura 2-13: Proceso de chunking y generación de embeddings en una base de datos vectorial [56]

La búsqueda de información en una base de datos vectorial constituye el núcleo del proceso de recuperación en RAG, ya que permite identificar y extraer los fragmentos de texto más relevantes para responder a un prompt dado. Este proceso comienza cuando el prompt del usuario se transforma en un embedding, utilizando el mismo modelo que generó los embeddings almacenados en la base de datos. El objetivo es encontrar los vectores en la base de datos que sean más similares al vector del prompt, lo que se logra mediante cálculos de similitud o distancia. Las métricas de distancia más utilizadas para medir esta similitud incluyen la distancia euclídea, la similitud coseno y el producto escalar. La distancia euclídea mide la longitud de la línea recta entre dos puntos en un espacio vectorial y se define matemáticamente como:

$$d(a, b) = \sqrt{\sum_{i=1}^N (a_i - b_i)^2} \quad (2-2)$$

Donde  $a$  y  $b$  son dos vectores de dimensión  $N$ . Esta métrica es intuitiva y efectiva cuando la magnitud de los vectores es relevante, pero puede ser menos adecuada para embeddings normalizados. Por otro lado, la similitud coseno mide el parecido entre dos vectores en función del ángulo que forman, sin tener en cuenta su magnitud. Matemáticamente, se define como:

$$\cos \theta = \frac{a \cdot b}{\|a\| \|b\|} \quad (2-3)$$

Donde  $\theta$  es el ángulo que forman los vectores  $a$  y  $b$ . Finalmente, el producto escalar mide la proyección de un vector sobre otro y se emplea frecuentemente como una medida de similitud directa, especialmente cuando los vectores están normalizados, ya que en ese caso es equivalente a la similitud coseno:

$$a \cdot b = \sum_{i=1}^N a_i b_i \quad (2-4)$$

Sin embargo, en muchas ocasiones realizar una búsqueda exacta de los vectores más cercanos en espacios de alta dimensionalidad, como los generados por embeddings de cientos o miles de dimensiones, resulta

computacionalmente inviable para grandes volúmenes de datos. Por ello, algunos sistemas RAG recurren a métodos aproximados, como Approximate Nearest Neighbors (ANN) [57], que ofrece un equilibrio entre precisión y eficiencia. Este método inspirado en K-Nearest Neighbors (KNN) [58] consiste en encontrar de manera aproximada los vectores más similares al vector de consulta, comprometiendo parcialmente la precisión a cambio de una mayor eficiencia computacional. Para ello, ANN se basa en la organización de los vectores en estructuras de datos avanzadas, como grafos jerárquicos, que facilitan realizar búsquedas eficientes.

Además, en escenarios más complejos, se emplea hybrid search [59], que combina la búsqueda vectorial con técnicas tradicionales basadas en palabras clave, como el algoritmo BM25 [60]. Esta combinación mejora la relevancia de los resultados al aprovechar tanto la semántica capturada por los embeddings como la precisión léxica de los métodos basados en coincidencia exacta. Por ejemplo, un sistema RAG que utilice este tipo de búsqueda podría priorizar fragmentos que no solo tengan embeddings cercanos al prompt, sino que también contengan términos o palabras clave presentes en la consulta.

Otro algoritmo importante en esta materia es Maximum Marginal Relevance (MMR) [61]. Esta técnica consiste en optimizar la selección de fragmentos para equilibrar relevancia y diversidad. Para ello, MMR selecciona vectores que son altamente similares al prompt, pero que también aportan información novedosa respecto a los ya recuperados, reduciendo la redundancia en el contexto proporcionado al LLM. Matemáticamente, MMR se define como:

$$MMR = \arg \max_{d_i \in R \setminus S} \left[ \lambda \cdot \text{Sim}(d_i, Q) - (1 - \lambda) \cdot \max_{d_j \in S} \text{Sim}(d_i, d_j) \right] \quad (2-5)$$

Donde:

- $R$ : Conjunto de todos los documentos o fragmentos disponibles.
- $S$ : Conjunto de documentos o fragmentos ya seleccionados.
- $d_i$ : Documento o fragmento candidato a ser seleccionado
- $Q$ : Consulta o prompt del usuario.
- $\text{Sim}(d_i, Q)$ : Medida de similitud entre el documento  $d_i$  y la consulta  $Q$ .
- $\max_{d_j \in S} \text{Sim}(d_i, d_j)$ : Máxima similitud entre el documento  $d_i$  y cualquier documento ya seleccionado en  $S$ .
- $\lambda$ : Parámetro que equilibra la relevancia y la diversidad, valiendo típicamente entre 0 y 1.

En esta fórmula, el término  $\lambda \cdot \text{Sim}(d_i, Q)$  favorece la selección de documentos relevantes para la consulta, mientras que la resta con el término  $(1 - \lambda) \cdot \max_{d_j \in S} \text{Sim}(d_i, d_j)$  penaliza la redundancia al reducir la puntuación de documentos similares a los ya seleccionados.

Una vez recuperados los fragmentos más relevantes, estos se integran en el proceso de generación de respuestas. Los textos obtenidos de la base de datos vectorial se añaden al prompt original como contexto adicional, formando una entrada enriquecida que se pasa al LLM. La integración de los fragmentos puede realizarse mediante diversas estrategias, dependiendo de las necesidades de la aplicación y las capacidades del LLM. Una técnica común es la concatenación simple, en la que los fragmentos recuperados se añaden directamente al final del prompt original, separados por delimitadores como saltos de línea o tokens especiales. Alternativamente, se puede emplear un formato estructurado, donde los fragmentos se organizan de manera más explícita para facilitar su procesamiento por el LLM. Otra estrategia avanzada implica el resumen o síntesis de los fragmentos antes de integrarlos. Si se recuperan múltiples fragmentos largos, un modelo auxiliar (o el propio LLM) puede condensarlos en un texto más breve y cohesionado. Esto reduce la carga computacional y se alinea con los límites de tokens de los LLMs, que típicamente están restringidos a un número máximo de tokens por entrada. La selección de fragmentos también es clave: si el número de fragmentos excede el límite de contexto, se priorizan aquellos con mayor similitud al prompt y se truncan los menos relevantes, asegurando que la entrada sea manejable por el LLM.

## 2.2.4 Fine-tuning

El fine-tuning o ajuste fino es una técnica avanzada que permite adaptar un LLM preentrenado a tareas específicas o dominios particulares, mejorando su rendimiento y utilidad en aplicaciones concretas. A diferencia del preentrenamiento, que se centra en aprender patrones lingüísticos generales a partir de grandes volúmenes de datos sin necesidad de etiquetarlos, el fine-tuning refina este conocimiento utilizando un conjunto de datos más pequeño y específico. Este proceso ajusta los pesos del modelo para que pueda generar respuestas más precisas y relevantes en contextos definidos, como responder preguntas técnicas, redactar textos formales o procesar información especializada. El resultado es un modelo optimizado que conserva las capacidades generales del preentrenamiento, pero con una especialización que lo hace más efectivo para las necesidades de la aplicación final.

Los modelos preentrenados, como los GPT de OpenAI o los Llama de Meta, son los modelos base sobre los que se aplica fine-tuning con el objetivo de especializarlos en tareas concretas, como seguir instrucciones, mantener una conversación coherente o responder preguntas. Algunos ejemplos son ChatGPT en sus distintas versiones o los modelos instruct de Llama. Los modelos preentrenados se entrenan en grandes conjuntos de datos procedentes de distintas fuentes y contextos, que incluyen textos de Internet, libros y artículos, mediante un enfoque de aprendizaje auto-supervisado. Gracias a esto, estos modelos consiguen aprender y crear una base de conocimiento amplia y generalista. Sin embargo, este preentrenamiento implica unos costes elevados en términos de hardware y energía, ya que entrenar un modelo como GPT-3 requiere miles de GPUs en funcionamiento durante meses, con estimaciones que sitúan los costes a partir de 4 millones de dólares [62]. Además, a pesar del conocimiento aprendido, los modelos preentrenados suelen estar limitados a tareas relacionadas con la predicción y completado de texto, además de sufrir problemas como el knowledge cutoff o las alucinaciones.

T prompt\* string Shift + Return to add a new line

What can you tell me about Spain?

Prompt to send to the model.

**Output**

Preview JSON

What is it like there? What do you think of the food? The people? The culture?  
 What's your favorite place in Spain? Why do you like it so much?  
 How long have you been living in Spain? How did you end up there?  
 Tell me about your family. Do they live in Spain with you?  
 Do you miss home? What do you miss most about home?  
 What's it like to be an expat? Do you feel like a part of the local community?  
 What are some of the challenges of being an expat?  
 What advice would you give someone who is thinking about moving to Spain?  
 What is your favorite thing about being an expat?  
 What is the hardest thing about being an expat?  
 What advice

**Output**

Preview JSON

Spain! A country with a rich history, vibrant culture, and stunning landscapes. Here are some interesting facts and highlights about Spain:

Geography and Climate:

- \* Spain is located in southwestern Europe, bordered by Portugal, Gibraltar, France, Andorra, and the Mediterranean Sea.
- \* The country has a diverse geography, with mountains, valleys, and coastlines along the Atlantic and Mediterranean.
- \* The climate varies greatly, with hot summers and mild winters in the south, and cooler temperatures in the north.

Culture:

- \* Spain has a unique cultural heritage, shaped by its history, language, and traditions.
- \* The country is home to many UNESCO World Heritage Sites, including the Alhambra in Granada, the Sagrada Familia in Barcelona, and

Figura 2-14: Comparación de las respuestas de Llama 3 8B y Llama 3 8B Instruct

El fine-tuning aborda estas limitaciones al especializar al modelo en tareas concretas. Un enfoque destacado es el instruction fine-tuning, que entrena al LLM en un conjunto de datos compuesto por pares de instrucciones y respuestas correctas. Esta técnica se basa en realizar un aprendizaje supervisado, donde los datos etiquetados guían al modelo para que ajuste sus salidas y genere las respuestas deseadas. A diferencia del preentrenamiento auto-supervisado, que se enfoca en patrones generales, el instruction fine-tuning supervisado permite que el modelo se adapte a formas particulares de responder preguntas o a realizar tareas específicas, como adoptar un tono formal o generar respuestas concisas.

Las ventajas del fine-tuning frente al preentrenamiento son claras. En primer lugar, es mucho más eficiente en términos de tiempo y recursos, ya que se utilizan datasets más pequeños y no requiere entrenar un modelo desde cero. En segundo lugar, permite adaptar al modelo a dominios específicos, logrando que adquiera nuevos conocimientos en áreas como la medicina, el derecho o la ingeniería. Gracias al fine-tuning, el LLM puede aprender terminología y conceptos especializados que no estaban bien representados en el dataset de preentrenamiento, utilizando para ello información privada o propietaria.

En comparación con RAG, el fine-tuning también presenta una serie de ventajas. Una de ellas es tener una menor latencia a la hora de producir respuestas, ya que al integrarse directamente el nuevo conocimiento en los pesos del modelo, no es necesario realizar búsquedas de información en una base de datos externa. Además, en tareas específicas donde el modelo ha sido ajustado, el rendimiento puede ser superior al de RAG, ya que el conocimiento está profundamente interiorizado en el modelo. Sin embargo, el fine-tuning requiere un conjunto de datos etiquetados de alta calidad, lo cual puede ser costoso y complejo de conseguir en comparación con agrupar y preprocesar documentos en un sistema RAG. También tiene la desventaja de que el conocimiento del modelo queda fijo en el momento del entrenamiento, a diferencia de RAG, que puede acceder a información actualizada dinámicamente. Esto hace que RAG sea más flexible para consultas en disciplinas en constante evolución, mientras que un modelo al que se le ha hecho fine-tuning está limitado al alcance de sus datos de entrenamiento.

A pesar de estas diferencias, el fine-tuning y RAG no son incompatibles, y pueden combinarse para aprovechar lo mejor de ambos enfoques. Por ejemplo, a un modelo se le podría hacer fine-tuning para especializarse en un dominio particular, como el soporte técnico, y luego usar RAG para recuperar información actualizada o complementaria en tiempo real, como manuales recientes o preguntas frecuentes. Esta integración combina la especialización y rapidez del fine-tuning con la capacidad de RAG para proporcionar contexto dinámico, siendo especialmente útil para aplicaciones donde se necesita tanto precisión en un dominio específico como acceso a datos externos o actualizados.

### 2.2.5 Reinforcement Learning from Human Feedback (RLHF)

Reinforcement Learning from Human Feedback (RLHF) es una técnica avanzada que permite alinear las respuestas de los LLMs con las preferencias y expectativas humanas, especialmente en tareas donde la salida deseada no puede ser fácilmente definida mediante un conjunto de datos etiquetados. A diferencia del fine-tuning supervisado, que ajusta el modelo utilizando pares de entrada-salida predefinidos, RLHF se basa en la realimentación proporcionada por humanos para guiar el aprendizaje del modelo, ofreciendo una adaptación más flexible y matizada a comportamientos complejos. Este enfoque ha sido clave en el desarrollo de modelos como ChatGPT [63].

El proceso de RLHF se estructura en tres etapas principales: la recopilación de la realimentación humana, el entrenamiento de un modelo de recompensa y la optimización del LLM mediante aprendizaje reforzado. En la primera etapa, se obtienen datos de realimentación humana, típicamente a través de encuestas y comparaciones entre respuestas generadas por el modelo. Las personas encargadas de evaluar las respuestas, en muchas ocasiones los propios usuarios del modelo, las califican según criterios como calidad, utilidad o seguridad, proporcionando una base para el aprendizaje. En la segunda etapa, estos datos se utilizan para entrenar un modelo de recompensa, una red neuronal separada que aprende a predecir la calidad de las salidas del LLM basándose en las preferencias humanas observadas. Finalmente, en la tercera etapa, el LLM se optimiza utilizando algoritmos de aprendizaje reforzado, como Proximal Policy Optimization (PPO) [64], para maximizar la recompensa determinada por el modelo de recompensa. Este enfoque iterativo permite al modelo captar comportamientos que son difíciles de especificar explícitamente, pero que pueden ser evaluados por humanos de manera efectiva.

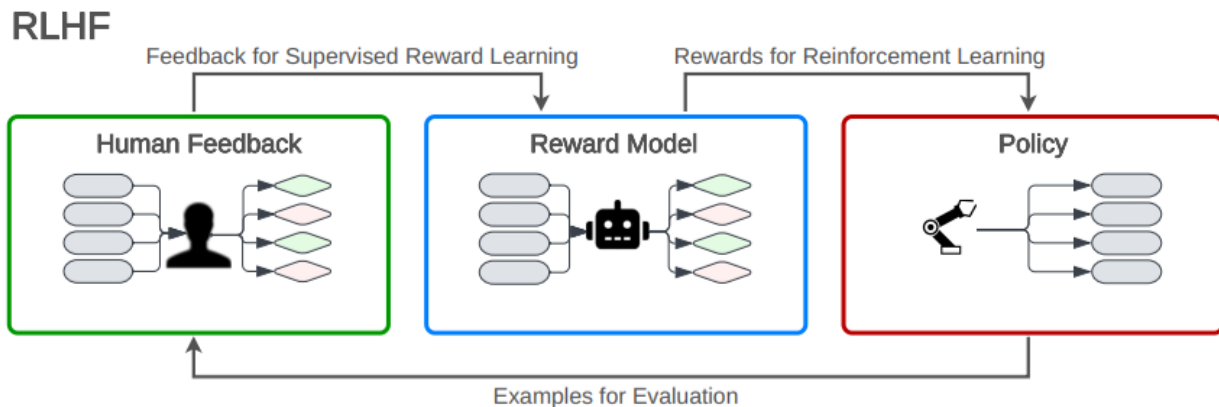


Figura 2-15: Esquema de los procesos en RLHF [65]

Las ventajas de RLHF son notables en comparación con otras técnicas de aprendizaje. Una de sus fortalezas principales es su capacidad para alinear el modelo con preferencias humanas complejas y subjetivas, que no pueden ser completamente representadas en un conjunto de datos etiquetados. En tareas de generación de texto como responder preguntas abiertas, la calidad de la respuesta depende de factores como la relevancia, la coherencia o la adecuación al tono, aspectos que RLHF puede optimizar mediante realimentación directa. Además, esta técnica es especialmente útil para mitigar comportamientos indeseados, como la generación de contenido sesgado o inapropiado, ya que los evaluadores pueden penalizar explícitamente estas salidas durante el proceso [66].

Sin embargo, RLHF también presenta desafíos significativos que limitan su implementación. Uno de los más destacados es la necesidad de una gran cantidad de datos de realimentación humana, lo que implica costes en términos de tiempo y recursos. Recolectar datos de alta calidad requiere evaluadores capacitados y un proceso de validación riguroso, lo que aumenta los recursos necesarios para aplicar RLHF a gran escala. Además, el entrenamiento suele ser computacionalmente intenso, ya que involucra múltiples iteraciones de generación, evaluación y ajuste del modelo. Otro inconveniente potencial es el riesgo de que el modelo de recompensa incorpore sesgos presentes en la realimentación humana, lo que podría derivar en comportamientos no deseados si no se controla adecuadamente.

En general, el fine-tuning supervisado y RLHF suelen integrarse en enfoques híbridos para maximizar la calidad de las respuestas de un LLM. Un modelo preentrenado puede someterse primero a un fine-tuning supervisado con un conjunto de datos etiquetados para adquirir conocimientos específicos, y luego refinarse con RLHF para alinear su comportamiento con las expectativas humanas. En la práctica, modelos avanzados como ChatGPT emplean esta estrategia, haciendo un fine-tuning inicial para conseguir la capacidad de seguir instrucciones, y luego RLHF para pulir sus capacidades conversacionales y mitigar posibles sesgos o respuestas inapropiadas [63].

## 2.3 Comparación de modelos de lenguaje

### 2.3.1 Evaluación de modelos

La evaluación de LLMs es un proceso esencial para verificar su rendimiento, fiabilidad y utilidad en aplicaciones específicas. A medida que estos modelos se integran en sistemas como asistentes virtuales o herramientas de análisis avanzado, resulta crítico emplear métodos de evaluación que midan tanto su capacidad general como su desempeño en tareas concretas. Hoy en día, la evaluación humana se considera la forma más fiable de evaluar un LLM. En este método, evaluadores expertos o usuarios finales revisan las respuestas del modelo y las califican según criterios específicos, como la precisión, la veracidad, la fluidez lingüística o la utilidad práctica. En tareas donde el tono o la sutileza del lenguaje son cruciales, como la redacción de textos formales o la generación de respuestas empáticas, los humanos podemos detectar matices que escapan a las métricas automatizadas. Sin embargo, este enfoque tiene claras limitaciones en términos de tiempo y escalabilidad, aunque sigue siendo un componente indispensable en las etapas finales de desarrollo o para validar modelos en

dominios críticos [67].

Un elemento clave en la evaluación de LLMs es la creación de un dataset de pruebas robusto y representativo. Este dataset debe reflejar el dominio de aplicación del modelo y estar diseñado para identificar problemas que pueda tener, como las alucinaciones. Para lograrlo, el dataset debe incluir una combinación de preguntas con respuestas verificables, prompts que requieran razonamiento complejo y ejemplos que desafíen los límites del modelo. En el caso de un LLM especializado en medicina, el dataset debería contener preguntas sobre diagnósticos con respuestas conocidas, junto con casos hipotéticos que evalúen la capacidad del modelo para evitar afirmaciones infundadas. La calidad y diversidad de un dataset son cruciales, ya que un conjunto de datos mal diseñado podría pasar por alto fallos críticos o sobreestimar el rendimiento del modelo [68]. La creación de estos datasets suele implicar la colaboración con expertos en el área de conocimiento que se está tratando y una iteración constante para cubrir posibles puntos ciegos.

Por otro lado, las métricas de evaluación son herramientas cuantitativas que nos permiten medir el rendimiento de un LLM de manera objetiva. Entre ellas, la precisión o accuracy es una de las más utilizadas, especialmente en tareas con respuestas definidas, como clasificación o preguntas de opción múltiple. La precisión calcula la proporción de respuestas correctas respecto al total de ejemplos en el dataset de test, ofreciendo una medida clara de la exactitud del modelo. Sin embargo, en tareas más abiertas, como la generación de texto libre, esta métrica puede ser insuficiente. En tales casos, se suele recurrir a métricas complementarias como la exactitud (precision en inglés), el recall o el F1-score, o bien a métricas basadas en similitud semántica, como BLEU y ROUGE, que miden la calidad lingüística de las respuestas [69]. La elección de la métrica depende del objetivo de la evaluación y de la naturaleza de la tarea, lo que subraya la necesidad de tener un enfoque adaptado a cada caso.

Complementando estas métricas, los benchmarks juegan un papel fundamental en la evaluación estandarizada de LLMs. Los benchmarks son conjuntos de tareas y datasets que permiten comparar el rendimiento de diferentes LLMs de manera objetiva. Entre los más populares se encuentran MMLU, HumanEval, HellaSwag o GSM-8K. MMLU (Massive Multitask Language Understanding) [70] abarca una amplia variedad de tareas de comprensión del lenguaje en múltiples dominios, como las matemáticas o las ciencias sociales, y sirve para analizar la versatilidad y el conocimiento general del modelo. HumanEval [71], desarrollado por OpenAI, evalúa la capacidad de un modelo para generar código Python funcional a partir de descripciones textuales. HellaSwag [72], por su parte, se enfoca en el razonamiento de sentido común, presentando situaciones incompletas donde el modelo debe seleccionar la respuesta más plausible, evaluando su comprensión en contextos cotidianos. Finalmente, GSM-8K [73] pone a prueba el razonamiento matemático mediante problemas de nivel escolar que requieren varios pasos lógicos, ofreciendo una medida de las capacidades analíticas del modelo. Estos benchmarks proporcionan métricas estandarizadas que facilitan la comparación entre modelos y ayudan a identificar áreas de mejora, siendo una herramienta esencial para la comunidad científica y los desarrolladores.

Otro enfoque innovador para la evaluación de LLMs es el sistema de ranking ELO, originalmente desarrollado para clasificar jugadores de ajedrez, que se ha adaptado para evaluar LLMs de forma competitiva. En este enfoque, dos modelos compiten respondiendo al mismo prompt, y un evaluador humano o automatizado determina cuál ofrece la mejor respuesta. A través de múltiples enfrentamientos, cada modelo acumula una puntuación que refleja su rendimiento relativo, generando un ranking global. Este método es particularmente útil para evaluar aspectos subjetivos, como la naturalidad de las respuestas, y ha sido implementado en plataformas como Chatbot Arena para clasificar a los LLMs más populares [74].

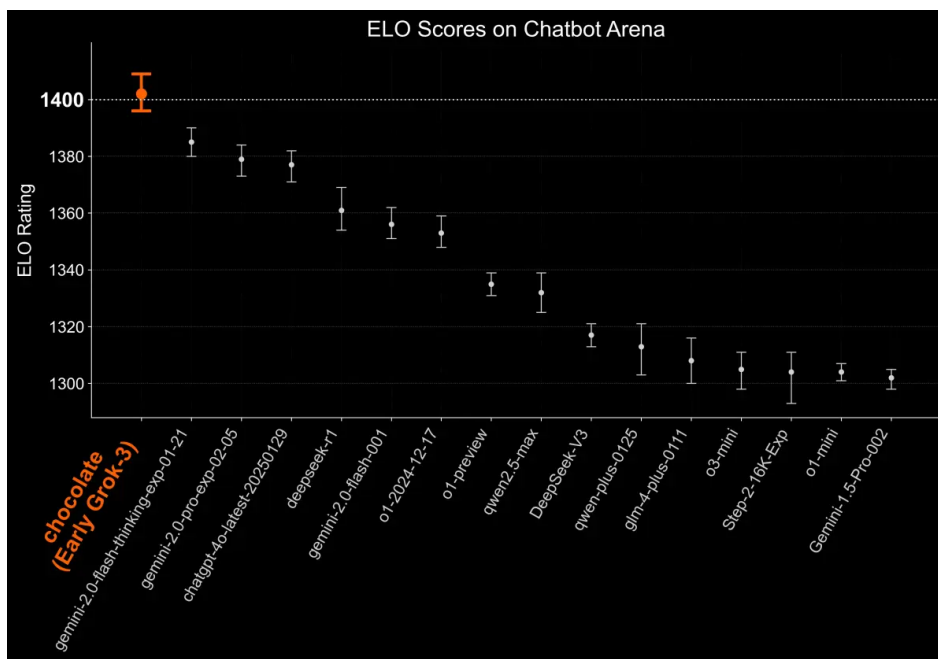


Figura 2-16: Clasificación de Chatbot Arena mostrada en la presentación de Grok 3 [75]

Para poder medir el rendimiento de un LLM de forma automatizada, consistente y reproducible, se suelen utilizar pipelines de evaluación. En estos pipelines se suelen adoptar dos enfoques principales: la evaluación determinista y la evaluación basada en otro LLM. En la evaluación determinista se suele utilizar la comparación exacta, donde la respuesta generada se compara directamente con una respuesta de referencia, considerándola correcta solo si coincide perfectamente. Alternativamente, un LLM puede actuar como evaluador, juzgando la calidad de las respuestas según criterios como la coherencia o la relevancia, proporcionando una calificación binaria a la respuesta de tipo “correcta” o “incorrecta”, o utilizar otro tipo de calificación numérica más precisa. Aunque este enfoque es más flexible y adaptable, introduce el riesgo de tener sesgos e imprecisiones por parte del modelo evaluador [76].

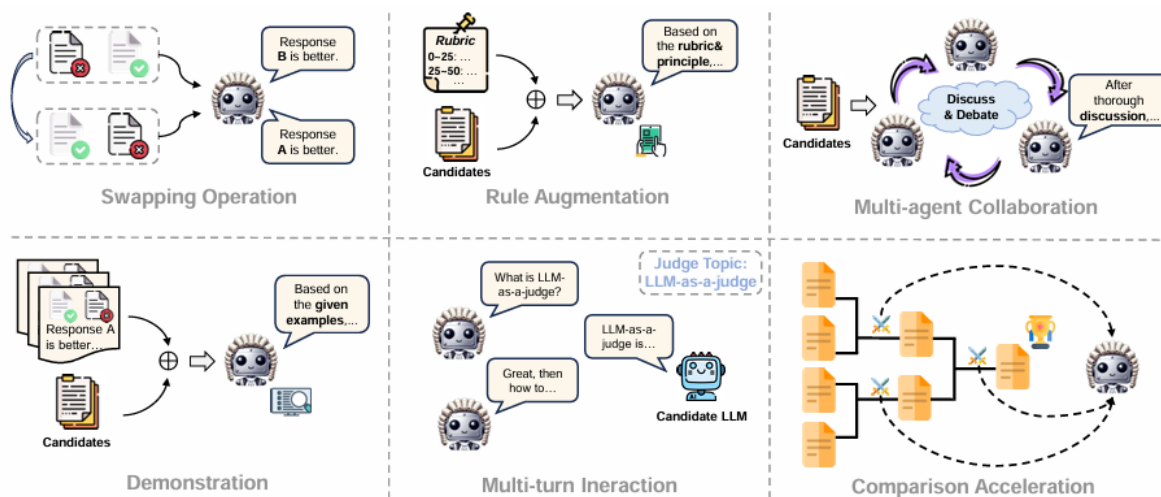


Figura 2-17: Estrategias de evaluación con LLM-as-a-judge [76]

### 2.3.2 Modelos destacados

A continuación, hablaremos brevemente sobre los LLMs más destacados en la actualidad, los cuales representan el estado del arte en el procesamiento del lenguaje natural gracias a sus innovaciones en arquitectura, entrenamiento, capacidad de razonamiento y aplicaciones multimodales. Estos modelos, desarrollados por empresas líderes en inteligencia artificial como OpenAI, xAI, Anthropic, DeepSeek AI, Meta, Alibaba y Google, están estableciendo el camino a seguir en esta disciplina. Su impacto se refleja en benchmarks estándar como

MMLU o HumanEval, y en rankings como Chatbot Arena, donde demuestran sus grandes capacidades en términos de precisión, eficiencia y adaptabilidad.

### 2.3.2.1 Modelos GPT

GPT-4o [77] es un modelo multimodal de última generación que destaca por su capacidad para procesar y generar texto e imágenes, consolidándose como una evolución de sus predecesores en la familia GPT. GPT-4o fue el primer modelo “omnimodal” de OpenAI, llamado así por su capacidad de procesar audio, texto e imagen utilizando una única red neuronal. Lanzado en mayo de 2024, este modelo mejora el rendimiento en tareas de comprensión y generación de lenguaje, alcanzando precisiones sobresalientes en benchmarks como MMLU (88%), HumanEval (90%) y MGSM (90%). Aunque no se ha revelado cuál es su arquitectura, se basa en la combinación de bloques Transformer con un enfoque en la eficiencia computacional, lo que le permite hacer un procesamiento más rápido que GPT-4. Además, GPT-4o ha sido entrenado utilizando técnicas avanzadas de fine-tuning supervisado y RLHF, las cuales han logrado que el modelo mejore la calidad de sus respuestas de forma significativa.

Por otro lado, GPT-o1 [45], lanzado en septiembre de 2024, representa un salto hacia modelos de razonamiento avanzado. A diferencia de GPT-4o, GPT-o1 utiliza un enfoque basado en chain-of-thought, es decir, en realizar un preprocesamiento antes de dar la respuesta que se basa en analizar y descomponer problemas complejos en pasos intermedios. Este modelo destaca resolviendo problemas matemáticos y lógicos, alcanzando una precisión cercana al 95% en el benchmark MATH y compitiendo directamente con humanos a la hora de responder preguntas científicas a nivel de doctorado.

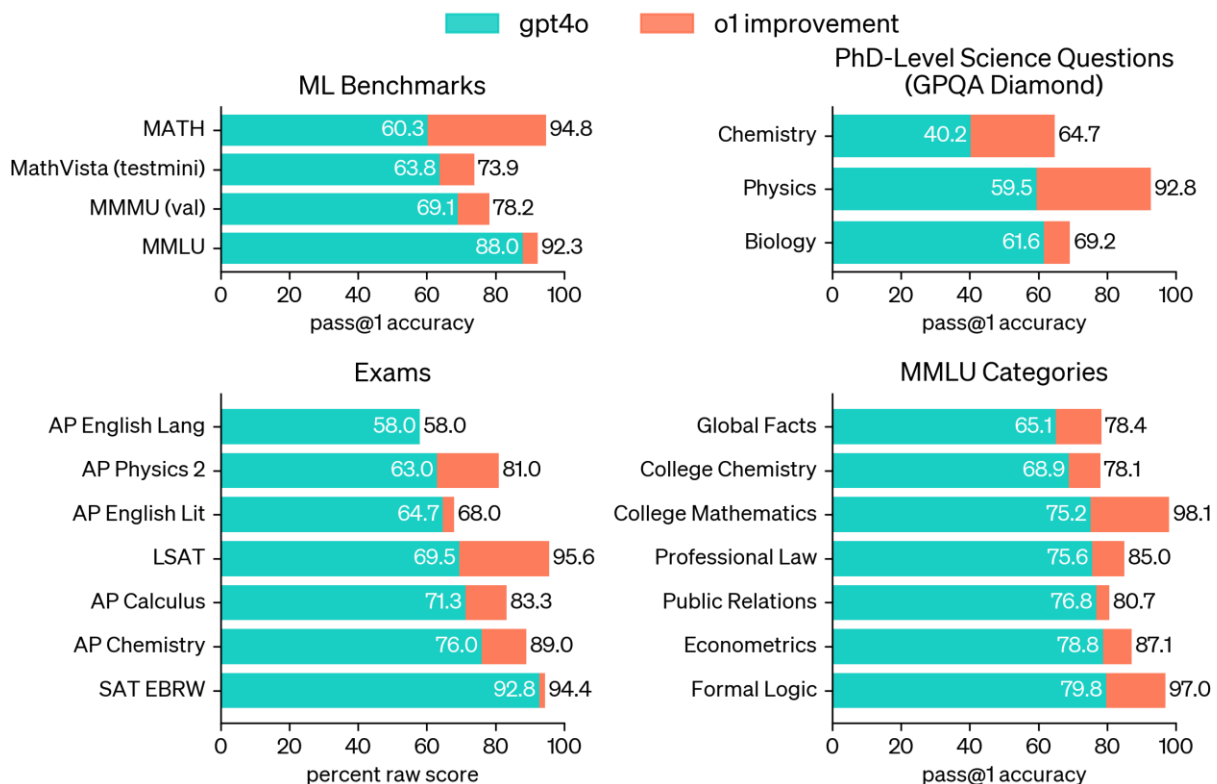


Figura 2-18: Comparación de GPT-4o y GPT-o1 en benchmarks [45]

En agosto de 2025 OpenAI lanzó GPT-5 [78], presentándolo como un sistema unificado que combina un modelo rápido y eficiente para respuestas sencillas con un modelo de razonamiento profundo, denominado GPT-5 Thinking, utilizando para ello un enrutador en tiempo real que decide qué modelo usar según la complejidad de la conversación. GPT-5 muestra mejoras significativas en múltiples dominios, alcanzando un 94,6% de precisión en el benchmark de matemáticas AIME’25, un 74,9% en el benchmark de ingeniería de software SWE-bench Verified y un 88% en Aider Polyglot para generación de código, mientras que en comprensión multimodal obtiene un 84,2% en MMMU, superando de forma consistente a GPT-4o. Además, GPT-5 cuenta con una

variante Pro orientada a entornos profesionales y de investigación avanzada que amplía el tiempo de razonamiento interno, permite sesiones de contexto extendido y ofrece acceso prioritario a herramientas especializadas.

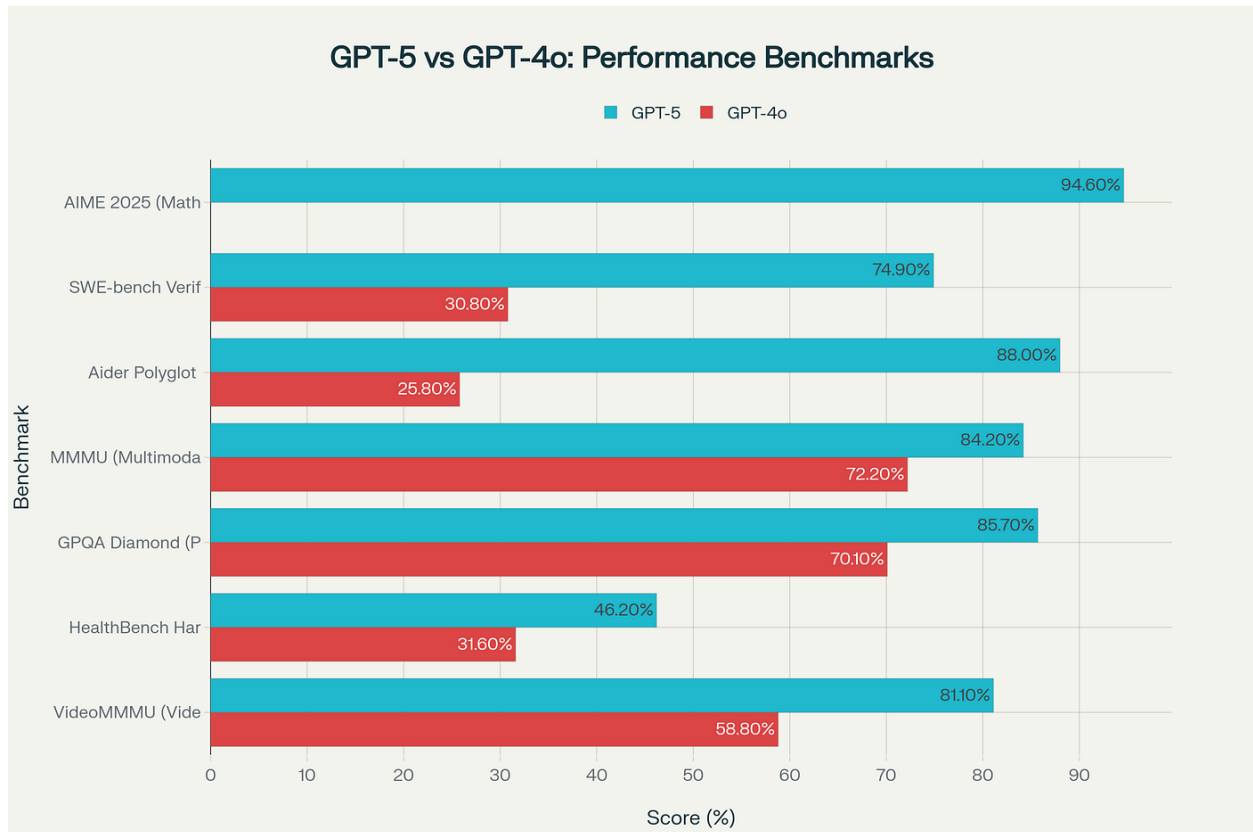


Figura 2-19: Comparación de GPT-5 y GPT-4o en benchmarks [79]

### 2.3.2.2 Grok 3

Grok 3 [75], creado por xAI, es un modelo diseñado para maximizar la veracidad y utilidad de las respuestas, posicionándose como un competidor destacado en el panorama de los LLMs. Presentado en febrero de 2025, este modelo mejora las capacidades de sus predecesores con una arquitectura Transformer optimizada y combinada con RAG, proporcionando respuestas más contextualizadas y precisas utilizando información obtenida mediante búsquedas en Internet. Grok 3 destaca en benchmarks como AIME'25 (93,3%) y GPQA (84,6%), y ha llegado a liderar el ranking de Chatbot Arena, con un ELO superior a 1400. Una característica notable es su diseño “maximally truthful”, que minimiza la censura y permite respuestas sin filtros en temas sensibles, lo que lo diferencia de competidores más conservadores.

### 2.3.2.3 Claude 3.5 Sonnet

Claude 3.5 Sonnet [80], desarrollado por Anthropic, es un modelo que combina seguridad, interpretabilidad y un alto rendimiento, consolidándose como uno de los LLMs más robustos hoy en día. Lanzado en junio de 2024, este modelo mejora a su predecesor Claude 3 pero manteniendo su coste y velocidad de inferencia. Claude 3.5 Sonnet logra resultados destacables en benchmarks como MMLU (88,7%), MGSM (91,6%) y HumanEval (92%), destacándose particularmente en tareas de escritura creativa y análisis de código. A diferencia de otros modelos, Anthropic ha enfocado Claude 3.5 Sonnet en la interpretabilidad, permitiendo a los desarrolladores entender mejor sus procesos de decisión, a diferencia de competidores como GPT-4o. Su capacidad para manejar contextos largos, de hasta 200.000 tokens, lo hace competitivo en tareas que requieren manejar una amplia ventana de contexto, como la generación de documentos técnicos.

### 2.3.2.4 DeepSeek R1

DeepSeek R1 [81], creado por la empresa china DeepSeek AI, ha irrumpido recientemente en el panorama de los LLMs como un modelo de razonamiento de código abierto que desafía a gigantes como OpenAI y Anthropic.

Este modelo utiliza una arquitectura Mixture-of-Experts (MoE) con 671 mil millones de parámetros, de los cuales solo 37 mil millones se activan por token, logrando una eficiencia computacional sobresaliente. DeepSeek R1 destaca en áreas técnicas como la programación, alcanzando el percentil 96,3% en Codeforces, y el razonamiento matemático, con un 97% en MATH, superando a GPT-4o y Claude 3.5 Sonnet en varios benchmarks. Su enfoque en el aprendizaje por refuerzo le permite tener un razonamiento que rivaliza mano a mano con GPT-o1, pero con un coste significativamente menor. DeepSeek R1 representa un hito en la democratización de la IA, demostrando que se pueden desarrollar modelos de altas capacidades sin necesidad de disponer de recursos exorbitantes, lo que ha generado un impacto significativo en la comunidad investigadora.

### 2.3.2.5 Llama 3.1

Llama 3.1 [82], desarrollado por Meta AI, es un modelo de código abierto que busca equilibrar rendimiento y accesibilidad, y ha sido ampliamente adoptado por la comunidad académica y empresarial desde su lanzamiento en julio de 2024. El modelo se distribuye de forma libre en las versiones de 8, 70 y 405 mil millones de parámetros, siendo esta última la más potente de todas. Llama 3.1 sobresale en tareas de comprensión y generación de texto, alcanzando una precisión del 88% en MMLU y 89% en HumanEval en su versión 405B. Su arquitectura basada en Transformers introduce mejoras en la tokenización y el manejo de contextos largos de hasta 128.000 tokens. A diferencia de modelos comerciales como GPT-4o, Llama 3.1 está diseñado para que se le pueda hacer fine-tuning de forma sencilla, lo que lo hace ideal para aplicaciones personalizadas en dominios específicos. Aunque no es un modelo que destaque por tener un razonamiento complejo como GPT-o1 o DeepSeek R1, su accesibilidad y flexibilidad lo convierten en un pilar fundamental para desarrolladores que buscan soluciones escalables y de bajo coste en el ámbito de los LLMs.

Category Benchmark	Llama 3.1 405B	Nemotron 4 340B Instruct	GPT-4 (0125)	GPT-4 Omni	Claude 3.5 Sonnet
General					
MMLU (0-shot, CoT)	88.6	78.7 (non-CoT)	85.4	88.7	88.3
MMLU PRO (5-shot, CoT)	73.3	62.7	64.8	74.0	77.0
IFEval	88.6	85.1	84.3	85.6	88.0
Code					
HumanEval (0-shot)	89.0	73.2	86.6	90.2	92.0
MBPP EvalPlus (base) (0-shot)	88.6	72.8	83.6	87.8	90.5
Math					
GSM8K (8-shot, CoT)	96.8	92.3 (0-shot)	94.2	96.1	96.4 (0-shot)
MATH (0-shot, CoT)	73.8	41.1	64.5	76.6	71.1
Reasoning					
ARC Challenge (0-shot)	96.9	94.6	96.4	96.7	96.7
GPQA (0-shot, CoT)	51.1	-	41.4	53.6	59.4
Tool use					
BFCL	88.5	86.5	88.3	80.5	90.2
Nexus	58.7	-	50.3	56.1	45.7
Long context					
ZeroSCROLLS/QuALITY	95.2	-	95.2	90.5	90.5
InfiniteBench/En.MC	83.4	-	72.1	82.5	-
NIH/Multi-needle	98.1	-	100.0	100.0	90.8
Multilingual					
Multilingual MGSM (0-shot)	91.6	-	85.9	90.5	91.6

Figura 2-20: Comparación de Llama 3.1 405B con otros LLMs en benchmarks [82]

### 2.3.2.6 Qwen2.5-Max

Qwen2.5-Max [83], desarrollado por Alibaba, es un modelo de la familia Qwen que destaca por su eficiencia y capacidad multilingüe, siendo lanzado en enero de 2025. Basado en una arquitectura MoE, este modelo logra un rendimiento competitivo en benchmarks como MMLU (87,9%) y GSM-8K (94,5%). Qwen2.5-Max está optimizado para tareas empresariales, como el análisis de datos y la generación de código, pudiendo manejar

grandes ventanas de contexto. Comparado con modelos como Claude, la familia de modelos Qwen ofrece un mejor equilibrio entre coste y rendimiento, habiendo sido adoptados por más de 90.000 empresas en sectores como la electrónica y el gaming [84]. A pesar de que no es un modelo open source, gracias a su fortaleza en razonamiento matemático y programación, Qwen2.5-Max se posiciona como un competidor directo de DeepSeek R1, aunque con un enfoque más orientado a aplicaciones comerciales.

### 2.3.2.7 Gemini 2.0

Gemini 2.0 [85], desarrollado por Google DeepMind, es una familia de modelos multimodales lanzada en diciembre de 2024, con variantes como Gemini 2.0 Pro y Flash que se caracterizan por su eficiencia y rendimiento. La versión Pro destaca por su capacidad para procesar texto e imágenes con contextos largos, alcanzando precisiones del 92,9% en el benchmark Natural2Code y 89,7% en MATH. Gemini 2.0 Flash por su parte, está optimizado para realizar inferencias rápidas, superando a modelos como GPT-4o en términos de latencia y coste. Entrenado con un corpus de datos masivo que incluye páginas web y fuentes internas de Google, Gemini 2.0 se beneficia de la integración con herramientas como Deep Research, lo que lo hace ideal para aplicaciones que requieren información actualizada. Aunque su versión “Thinking” no ha superado a modelos de razonamiento como GPT-o1, Gemini 2.0 Pro sigue siendo un referente en aplicaciones multimodales y de alta eficiencia.

## 2.4 Caso de estudio: Llama 3.1

Este apartado tiene como objetivo explorar con un poco más de profundidad las características técnicas y prácticas del modelo Llama 3.1, presentado con anterioridad en el apartado 2.3.2.5, ya que es el modelo que utilizaremos en este proyecto. El interés en Llama 3.1 radica en su diseño avanzado, técnicas de entrenamiento innovadoras y desempeño en benchmarks estándar. A lo largo de este análisis, examinaremos su arquitectura, proceso de entrenamiento, diferencias entre versiones, y una serie de casos de éxito de aplicaciones basadas en Llama 3.1, con el fin de justificar su selección para este proyecto.

### 2.4.1 Arquitectura de Llama 3.1

Llama 3.1, similar en muchos aspectos a sus predecesores Llama 3 y Llama 2, tiene una arquitectura basada en Transformers de tipo decoder-only, al igual que otros modelos de lenguaje como los GPT. Su diseño autoregresivo permite la generación de secuencias de texto mediante un proceso iterativo en el que el modelo emplea los tokens de salida previamente generados como entrada para predecir de manera condicionada el siguiente token en la secuencia. Para ello, la red neuronal está compuesta por bloques Transformer que incorporan capas multi-head attention y feed-forward, incorporando además capas de normalización aplicadas estratégicamente para estabilizar el entrenamiento del modelo y mejorar su convergencia [86] [87].

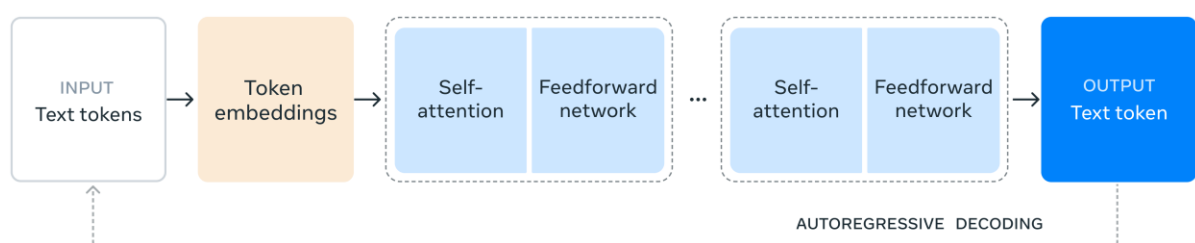


Figura 2-21: Arquitectura simplificada de Llama 3.1 [86]

Una de las optimizaciones más destacadas de Llama 3.1 es la implementación de una técnica llamada Grouped-Query Attention (GQA) [88]. Esta técnica reduce la complejidad computacional de los mecanismos de atención al agrupar las consultas en grupos, donde cada grupo comparte las mismas claves y valores, minimizando así el espacio requerido en memoria. Al disminuir las operaciones necesarias en el proceso de atención, especialmente en el cálculo de similitud entre consultas y claves, GQA mejora la eficiencia durante la inferencia,

particularmente en contextos largos. Combinada con otras técnicas de optimización y gestión de memoria, GQA permite que Llama 3.1 soporte hasta 128.000 tokens de entrada, lo cual supone un salto significativo frente a los 8.192 tokens que podía manejar Llama 3.

Por otro lado, uno de los componentes más importantes de Llama 3.1 es su sistema de tokenización. El vocabulario que maneja su tokenizador, es decir, el conjunto de tokens únicos que puede reconocer y procesar, tiene un tamaño de 128.000 tokens [86]. De ese número de tokens, 100.000 provienen del tokenizador tiktoken [89], a los que se han añadido 28.000 tokens para dar un mejor soporte a idiomas que no sean el inglés, como el alemán, francés, italiano, portugués, hindi, español y tailandés. En el idioma inglés, Llama 3.1 logra una tasa de compresión de 3,94 caracteres por token, en comparación con los 3,17 caracteres por token de Llama 2, lo que permite procesar más texto utilizando los mismos recursos computacionales.

Debido al tamaño de las distintas versiones de Llama 3.1, especialmente la de 405 mil millones de parámetros, y la longitud del contexto que pueden manejar, se han utilizado una serie de técnicas avanzadas para reducir el espacio que ocupan los modelos en memoria, destacando entre ellas la cuantización. En particular, en la inferencia se realiza una cuantización FP8 (8 bits en formato flotante) en la mayor parte de las multiplicaciones matriciales, especialmente en los parámetros y funciones de activación de las capas feed-forward, a las cuales se les atribuye cerca del 50% del tiempo de cómputo en inferencia [86]. Sin embargo, la cuantización no se realiza en las capas de atención del modelo para evitar perder precisión de forma excesiva. Tras realizar las operaciones, los resultados se obtienen en formato BF16 (16 bits en formato flotante).

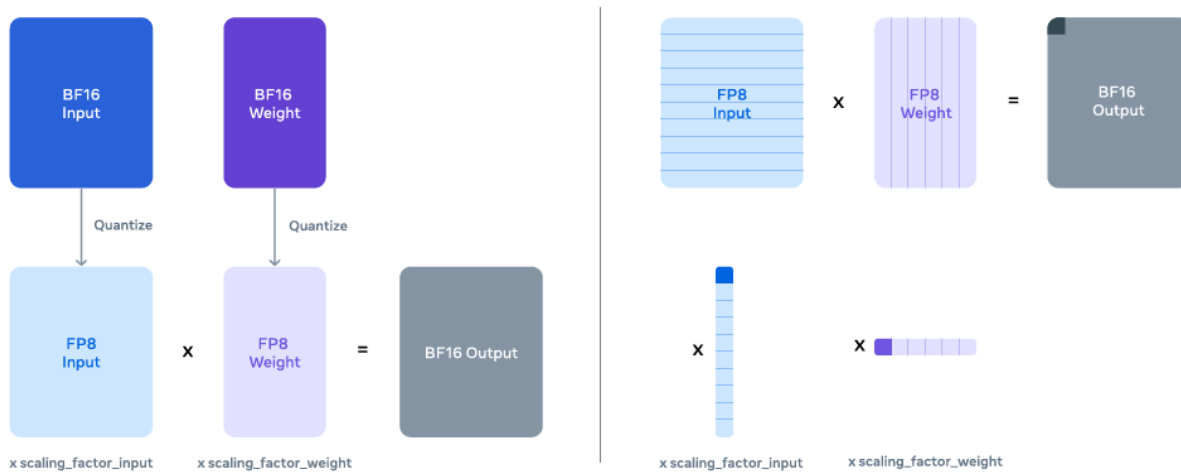


Figura 2-22: Cuantización FP8 en Llama 3.1 con escalado a nivel de tensor o a nivel de fila [86]

## 2.4.2 Proceso de entrenamiento

El preentrenamiento constituye la fase inicial y más extensa en el desarrollo de Llama 3.1, fase en la cual el modelo adquiere un conocimiento general del lenguaje a partir de un corpus masivo de texto no etiquetado. En el caso de Llama 3.1, este modelo se preentrenó con más de 15 billones de tokens de texto, un aumento significativo comparado con los 2 billones de tokens de Llama 2 [82] [87]. La composición del corpus incluye una mezcla equilibrada de diversos dominios, como cultura general, razonamiento matemático, desarrollo de software y programación, así como traducción y tareas multilingües. Es relevante mencionar que el modelo fue entrenado con información recopilada hasta diciembre de 2023, abarcando eventos y avances hasta esa fecha.

Tras el preentrenamiento, Meta aplicó fine-tuning al modelo para conseguir que este fuera capaz de adaptarse a tareas específicas, especialmente para ser un asistente conversacional. Este proceso involucró dos técnicas principales: fine-tuning supervisado (SFT) y aprendizaje por refuerzo con retroalimentación humana (RLHF). En la fase de SFT, el modelo fue entrenado con un dataset que incluía instrucciones y respuestas esperadas, como ejemplos de conversaciones y preguntas y respuestas. Más de la mitad del dataset se compone de conversaciones en inglés, seguido por tareas de razonamiento y uso de herramientas, así como problemas de programación y preguntas de exámenes [86].



comparten ciertas características clave, como el uso de la función de activación SwiGLU, tener un vocabulario de 128.000 tokens y el uso de embeddings posicionales de tipo RoPE (Rotary Position Embedding). Sin embargo, se diferencian en aspectos importantes, como el número de capas, el número de cabezas de atención y la dimensión de las capas FFN, que determina el número de neuronas de la capa feed-forward de los bloques Transformer de la red. La versión 8B cuenta con 32 capas, 32 cabezas de atención y una dimensión de FFN de 14.336. Por su parte, la versión 70B incluye 80 capas, 64 cabezas de atención y una dimensión de FFN de 28.672. Finalmente, la versión 405B, la más avanzada, tiene 126 capas, 128 cabezas de atención y una dimensión de FFN de 53.248. Además, en cada una de las versiones se ha utilizado un learning rate máximo distinto para optimizar el entrenamiento en cada caso [86].

	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	14,336	28,672	53,248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	$3 \times 10^{-4}$	$1.5 \times 10^{-4}$	$8 \times 10^{-5}$
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ( $\theta = 500,000$ )		

Figura 2-25: Hiperparámetros de cada una de las versiones de Llama 3.1 [86]

A nivel de rendimiento es lógico pensar que los modelos más grandes logran mejores resultados en términos de precisión y capacidad de generalización, y así se ve reflejado en distintos benchmarks estándar. Por ejemplo, en el benchmark MMLU, que evalúa el conocimiento general en múltiples ámbitos, la versión 405B alcanza una precisión del 88,6%, superando a la 70B con 86% y a la 8B con 73%. En tareas de generación de código, como HumanEval, la versión 405B logra un 89%, seguida por la 70B con 80,5% y la 8B con 72,6%. Asimismo, en GSM-8K, un benchmark de razonamiento matemático, la versión 405B obtiene un 96,8%, la 70B un 95,1% y la 8B un 84,5% [82]. Estos resultados demuestran que a medida que aumenta el número de parámetros, el modelo es capaz de capturar patrones más complejos y generalizar mejor en una amplia gama de tareas. No obstante, este incremento en capacidad viene acompañado de un mayor coste computacional, tanto en términos de memoria como de tiempo de inferencia, lo que hace que las versiones más pequeñas sean más prácticas para ciertos escenarios.

Por otro lado, para cada una de las versiones de Llama 3.1, Meta distribuye dos tipos de modelo: base e instruct. Los modelos base de Llama 3.1 constituyen la versión original preentrenada del sistema, en la que se ha aprendido una representación general del lenguaje a partir de un corpus masivo de datos sin una orientación específica hacia tareas particulares. Al no estar orientados a seguir instrucciones concretas, sus salidas pueden carecer de la precisión y coherencia necesarias en aplicaciones donde se requiere una interacción directa con el usuario. Por el contrario, los modelos instruct son los que han sido sometidos a un proceso adicional de fine-tuning orientado específicamente a mejorar la capacidad de comprender y responder a instrucciones.

	Finetuned	Multilingual	Long context	Tool use	Release
Llama 3 8B	✗	✗ <sup>1</sup>	✗	✗	April 2024
Llama 3 8B Instruct	✓	✗	✗	✗	April 2024
Llama 3 70B	✗	✗ <sup>1</sup>	✗	✗	April 2024
Llama 3 70B Instruct	✓	✗	✗	✗	April 2024
Llama 3.1 8B	✗	✓	✓	✗	July 2024
Llama 3.1 8B Instruct	✓	✓	✓	✓	July 2024
Llama 3.1 70B	✗	✓	✓	✗	July 2024
Llama 3.1 70B Instruct	✓	✓	✓	✓	July 2024
Llama 3.1 405B	✗	✓	✓	✗	July 2024
Llama 3.1 405B Instruct	✓	✓	✓	✓	July 2024

Figura 2-26: Comparación entre las distintas versiones de Llama 3 y Llama 3.1 [86]

A nivel práctico, la forma en la que se deben formatear los prompts para cada modelo es distinta. Mientras que el modelo base no requiere una estructura concreta en los prompts, el modelo instruct necesita que el prompt incorpore unas etiquetas específicas que guíen la generación de respuestas. La plantilla de prompt de los modelos instruct está diseñada para estructurar la interacción entre el usuario y el modelo, definiendo de forma precisa cómo se debe organizar la información para aprovechar al máximo las capacidades del sistema.

En esencia, el prompt se compone de una secuencia de tokens y delimitadores especiales que indican el inicio y fin de cada sección, así como el rol que cumple cada bloque de texto [95]. La estructura del prompt empieza con el token de inicio, `<|begin_of_text|>`, que marca el comienzo del contenido que se le proporcionará al modelo. Seguidamente, se utiliza el token `<|start_header_id|>` para delimitar el rol del mensaje. Los roles disponibles para cada mensaje son “system”, “user”, “assistant” e incluso “ipython” para tareas específicas que involucren llamadas a herramientas o ejecución de código. Cada uno de estos roles se enmarca entre `<|start_header_id|>` y `<|end_header_id|>` para identificar claramente la fuente del mensaje. Por ejemplo, el rol “system” puede utilizarse para establecer el contexto de la conversación, proveer instrucciones generales, o dar información sobre el knowledge cutoff del modelo y otros parámetros esenciales para la generación de respuestas. Además, la plantilla de prompts permite definir conversaciones multi-turno de forma que cada interacción entre el usuario y el modelo se encierra en bloques diferenciados gracias a los roles “user” y “assistant”. Cada uno de estos mensajes termina con el token `<|eot_id|>`, lo que indica que la respuesta o el turno ha finalizado. Este mecanismo de conversaciones multi-turno puede ser útil para aplicaciones de tipo chatbot, en las que el modelo debe tener constancia de lo que se ha dicho anteriormente para mantener el hilo de la conversación.

### Código 2-1: Prompt de ejemplo para los modelos Llama 3.1 Instruct

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

You are a helpful
assistant<|eot_id|><|start_header_id|>user<|end_header_id|>

What is the capital of
France?<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

Para el presente proyecto, se ha optado por utilizar la versión Llama 3.1 8B Instruct debido a su equilibrio entre prestaciones y tamaño. A pesar de que las versiones más grandes, como la 70B y la 405B, ofrecen un rendimiento superior, su uso implica un coste computacional significativamente mayor en términos de memoria y tiempo de inferencia. Por ejemplo, en plataformas open source como Ollama [96], de las cuales podemos descargar los pesos entrenados de los modelos, se nos indica que el modelo 8B necesita unos 4,9 GB de memoria VRAM, mientras que el modelo 70B ocupa 43 GB y el 405B ocupa 243 GB. Por lo tanto, para ejecutar los modelos más grandes es necesario disponer de clústeres de tarjetas gráficas de última generación, como las GPUs A100 o H100 de NVIDIA. En cambio, la versión 8B proporciona una base sólida con un rendimiento aceptable, y al mismo tiempo resulta más accesible y manejable en entornos con recursos limitados. Además, al tratarse de la variante instruct, el modelo ya está preparado para seguir instrucciones de manera precisa y coherente, lo cual es esencial para aplicaciones que requieren una interacción conversacional con el usuario. En conjunto, estas características hacen que Llama 3.1 8B Instruct sea la opción más adecuada para este proyecto.

### 2.4.4 Aplicaciones prácticas

El uso de las diferentes versiones de Llama en el ámbito empresarial, junto a su adaptación a ámbitos específicos mediante fine-tuning por parte de la comunidad de desarrolladores, reflejan la utilidad y versatilidad de estos modelos. Proveedores de servicios cloud como AWS o Azure están incorporando Llama en sus soluciones y servicios de inteligencia artificial, lo que ha impulsado a un número creciente de empresas a utilizar estos modelos.

Empresas como Goldman Sachs y AT&T están utilizando modelos Llama para sus actividades operativas, aplicándolos en áreas tan diversas como la atención al cliente, la revisión de documentos o la generación de código [97]. En el ámbito de la consultoría estratégica, Accenture, en colaboración con NVIDIA, ha lanzado el marco Accenture AI Refinery, basado en NVIDIA AI Foundry, que permite a las empresas crear modelos LLM personalizados utilizando la colección Llama 3.1. Este enfoque integra cuatro componentes clave: personalización de modelos con datos corporativos, una plataforma de selección de modelos (Switchboard), un sistema de indexación de conocimiento empresarial (Enterprise Cognitive Brain) y una arquitectura autónoma (Agentic Architecture) para ejecutar tareas con supervisión mínima [98]. Asimismo, la colaboración entre Meta y Scale, anunciada en julio de 2024, ha explorado el uso de la versión 405B en dominios especializados como chatbots empresariales y síntesis de contenido en áreas legales y financieras, demostrando resultados competitivos frente a modelos propietarios en tareas que involucran RAG [99].

Paralelamente, la comunidad open source ha desempeñado un papel clave en la expansión de Llama 3.1 mediante la creación de modelos derivados y forks, disponibles en plataformas como Hugging Face. Algunos ejemplos son AstroSage-Llama-3.1-8B, especializado en astronomía, astrofísica y cosmología; Llama-3.1-Nemotron-70B-Instruct-HF, entrenado por NVIDIA para mejorar la utilidad de las respuestas para los usuarios; y DeepSeek-R1-Distill-Llama-70B, un modelo que reduce el tamaño y coste computacional del Deepseek-R1 original mediante la transferencia de conocimiento al modelo Llama. En este ecosistema, Unsloth [100] destaca como herramienta open source para la optimización y personalización de Llama 3.1 y otros modelos abiertos. Unsloth es totalmente compatible con las librerías de Hugging Face, y ofrece notebooks gratuitos en Google Colab que permiten reentrenar modelos como Llama 3.1 8B. Gracias a Unsloth, se puede hacer fine-tuning utilizando hasta un 70% menos de memoria y de forma hasta 5 veces más rápida mediante técnicas PEFT como LoRA y QLoRA.

## 2.5 NotebookLM: un asistente documental basado en LLMs

NotebookLM [101] es una herramienta desarrollada por Google que integra modelos de lenguaje de la familia Gemini con una arquitectura basada en RAG. Su propósito es facilitar que el usuario pueda hacer preguntas y obtener respuestas basadas en el contenido de los documentos que ha subido previamente en la plataforma. El sistema opera mediante la incorporación de fuentes externas, como archivos PDF, textos académicos, presentaciones, e incluso archivos de audio y videos de YouTube, en un repositorio indexado que actúa como base de conocimiento. A partir de esta información, el modelo es capaz de responder preguntas, generar resúmenes, elaborar mapas conceptuales, guías de estudio o cuestionarios, incluyendo referencias explícitas a los documentos utilizados como fuente. Esta capacidad de trazabilidad tiene como objetivo mitigar uno de los principales desafíos asociados a los modelos de lenguaje actuales: la generación de contenido no veraz o “alucinaciones” [102].

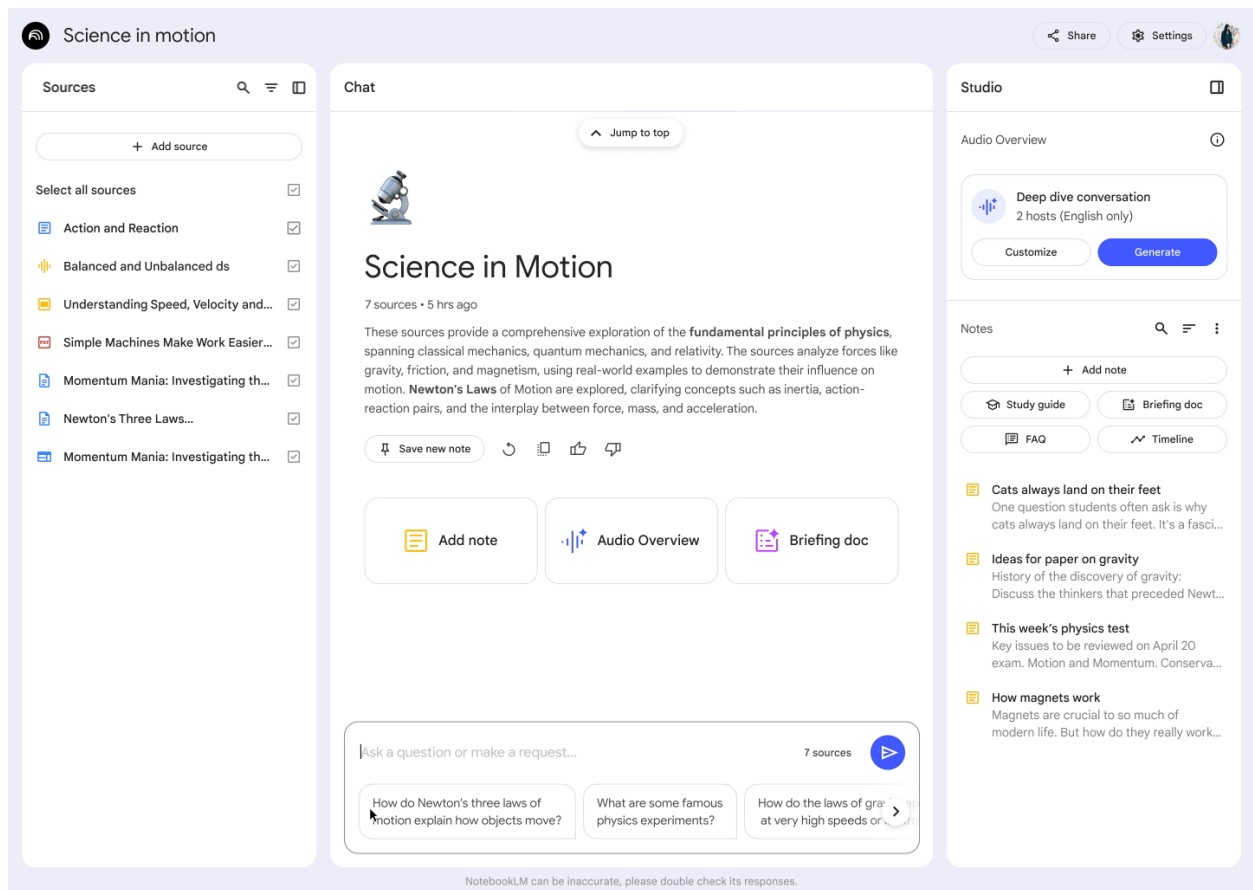


Figura 2-27: Interfaz de usuario de NotebookLM [103]

La interfaz de NotebookLM se organiza en tres módulos principales: el panel “Sources”, en el que el usuario puede añadir los documentos que desee utilizar como base de conocimiento, el panel de chat para hablar con el modelo de lenguaje, y el panel “Studio”, que permite generar guías de estudio, resúmenes, e incluso audios de tipo podcast para asimilar los conceptos con mayor facilidad. Gracias a esta organización, esta herramienta puede usarse tanto a nivel individual como en entornos de aprendizaje guiado. Por ejemplo, en entornos educativos, los profesores pueden subir material docente, como artículos, vídeos, ejercicios o manuales, y configurar el comportamiento del asistente mediante una serie de instrucciones.

Una aplicación concreta de esta funcionalidad fue descrita por E. Tufino [102], profesor del Departamento de Física y Astronomía de la Universidad de Padua, quien utilizó NotebookLM para implementar un tutor de física utilizando un enfoque colaborativo. En su trabajo, Tufino diseñó un entorno de aprendizaje para estudiantes donde el modelo, en lugar de proporcionar respuestas directas, guiaba al estudiante a través de preguntas fundamentadas en los documentos cargados previamente por el docente. Para lograrlo, se incorporaron no solo problemas y materiales de estudio, sino también un “manual de entrenamiento” que definía con precisión cómo debía comportarse el asistente en cada situación. Este enfoque permitió personalizar la interacción con los estudiantes, diferenciando entre problemas que contaban con soluciones orientadas y otros en los que el modelo debía apoyarse exclusivamente en sus capacidades generales de razonamiento.

Además, Tufino demostró en su estudio que esta estrategia no solo mejora la trazabilidad de las respuestas, sino que también fomenta una comprensión más profunda y activa por parte del estudiante. El sistema fue capaz de generar cuestionarios, resúmenes estructurados y preguntas de repaso, incluyendo siempre referencias explícitas a los documentos utilizados como base. Sin embargo, también señaló ciertas limitaciones de la herramienta, como la imposibilidad de interpretar correctamente contenido visual en algunos formatos, la dificultad a la hora de resolver problemas que requieren desarrollos matemáticos complejos, y la incapacidad para aplicar metodologías didácticas que combinan texto, esquemas y diagramas, como en las pizarras tradicionales usadas en la enseñanza. Por otro lado, también critica que sea necesario pagar una suscripción a NotebookLM Plus para gestionar adecuadamente el acceso del alumnado al entorno de aprendizaje, ya sea mediante enlaces públicos restringidos o interfaces de solo lectura. A pesar de estas restricciones, NotebookLM representa un caso paradigmático de cómo un sistema RAG puede ser aprovechado para tareas de tutoría y aprendizaje

personalizado, sin necesidad de realizar fine-tuning sobre un modelo base.

La herramienta propuesta en este proyecto comparte con NotebookLM los principios fundamentales de utilizar una arquitectura RAG, pero difiere notablemente en su diseño, objetivos y capacidades. Mientras que NotebookLM está orientado a un uso generalista y transversal, con grandes capacidades multimodales, la solución desarrollada en este proyecto se plantea como un sistema de chat especializado para asistir en el aprendizaje y la consulta sobre sistemas de comunicaciones digitales, utilizando documentación técnica adaptada específicamente para este dominio. No obstante, el sistema ha sido diseñado con un enfoque modular y personalizable, lo que permite adaptarlo con facilidad a otros ámbitos de conocimiento. Para ello, bastaría con modificar los documentos almacenados en la base de datos vectorial y ajustar el system prompt del modelo para adaptar su comportamiento, aprovechando así la flexibilidad que proporcionan los modelos instruct de la familia Llama.

Otra de las principales diferencias es el grado de control sobre los modelos de inteligencia artificial que utilizan ambos sistemas. NotebookLM se basa en la familia de modelos Gemini, propiedad de Google, cuya ejecución ocurre en sus propios servidores y cuyos parámetros internos no son configurables por el usuario. Por el contrario, el modelo Llama utilizado en este proyecto es totalmente open source, lo que permite modificar el comportamiento y las instrucciones que debe seguir el modelo, ajustar parámetros de inferencia, e incluso plantear fases futuras de fine-tuning para mejorar su rendimiento si fuera necesario. Además, en este proyecto se ha optado por realizar una cuantización en 8 bits del modelo, lo que permite ejecutarlo en hardware con recursos limitados con tiempos de inferencia razonables, aunque con una leve pérdida de precisión con respecto al modelo original sin cuantizar.

Por otro lado, un aspecto clave en el que ambos sistemas difieren es el de la privacidad, la protección y la soberanía de los datos. Mientras que NotebookLM requiere subir los documentos a servidores de Google para su indexación y consulta, el sistema propuesto en este proyecto puede ejecutarse tanto en local como en la nube, por lo tanto, no habría necesidad de conectarse a servidores externos ni de enviar datos confidenciales fuera del entorno del usuario. Este enfoque permite cumplir con las normativas vigentes de protección de datos, como el Reglamento General de Protección de Datos (RGPD) de la Unión Europea (Reglamento UE 2016/679) [104], que exige garantizar la minimización del tratamiento de datos personales, asegurar su confidencialidad, y evitar transferencias no autorizadas a terceros países fuera del Espacio Económico Europeo. En el caso concreto de este proyecto, los datos utilizados corresponden a material docente que no contiene información personal, por lo que el cumplimiento de este reglamento no resulta estrictamente necesario. No obstante, en caso de que se quisiera adaptar el sistema desarrollado para una aplicación que requiera el tratamiento de datos personales o información sensible, como por ejemplo, en instituciones o empresas que manejen know-how, patentes o tecnologías propietarias, la posibilidad de ejecutar el sistema localmente o en servidores propios permite una gestión más segura y controlada de los datos.

Además, desde el punto de vista de su arquitectura, la herramienta desarrollada ha sido diseñada para funcionar como una API REST, lo que facilita su integración en entornos diversos. Aunque en este proyecto se ha incluido una interfaz web de tipo chat, la exposición del modelo mediante una API permite incorporar sus capacidades en otros sistemas, como en plataformas educativas, aplicaciones móviles, entornos de atención al cliente, o herramientas de formación interna en empresas. Esta estrategia es coherente con las tendencias actuales, como la adoptada por OpenAI con Agent AI [105], donde los modelos no sólo se emplean en interfaces conversacionales, sino también como componentes autónomos en sistemas más amplios capaces de ejecutar acciones, interactuar con usuarios, colaborar con otros modelos, y tomar decisiones en función del contexto y del entorno.

# 3 IMPLEMENTACIÓN DEL SISTEMA

---

*El Terminator está ahí fuera. No se puede razonar con él, es un exterminador. No siente lástima, ni remordimiento, ni miedo.*

*- Terminator -*

En este capítulo, describiremos la implementación del sistema diseñado para crear un asistente experto en sistemas de comunicaciones digitales. En primer lugar, presentaremos la arquitectura general del sistema, describiendo cada uno de los componentes del flujo RAG, así como las relaciones y dependencias entre ellos. A continuación, explicaremos en detalle el despliegue y ejecución del modelo Llama 3.1 en su versión instruct, incluyendo la creación de prompts siguiendo la estructura especificada por Meta, la configuración de parámetros de inferencia, y el uso de técnicas de cuantización. También se abordará la integración del modelo con el resto de componentes del sistema para garantizar un flujo de datos eficiente y una latencia mínima en la generación de respuestas. Posteriormente, abordaremos el proceso de selección y preprocesamiento de los documentos que conforman la base de conocimiento sobre la que se realiza la recuperación de contexto, así como su vectorización y almacenamiento en una base de datos vectorial basada en ChromaDB. Se explicarán las estrategias aplicadas para segmentar documentos en fragmentos de tamaño óptimo, evitando pérdida de información semántica y asegurando la máxima precisión en la recuperación de contexto. También especificaremos el diseño del pipeline RAG utilizando el framework LangChain, que coordina la recuperación de información de la base de datos vectorial, la incorporación de dicha información a los prompts de entrada, y la generación de respuestas por parte del propio modelo Llama. Finalmente, explicaremos la implementación de la API REST que expone la funcionalidad del sistema, junto con el desarrollo de la interfaz de usuario que permite la interacción con el asistente de manera accesible e intuitiva.

## 3.1 Arquitectura del sistema

Podemos describir la arquitectura del sistema en dos niveles principales: por un lado, la arquitectura general del mismo, y por otro, el flujo de funcionamiento del sistema RAG.

En primer lugar, presentaremos una arquitectura a alto nivel, que muestra elementos como la interfaz de usuario (frontend), o la API REST que permite la comunicación con el sistema RAG (backend). Este nivel ofrece una visión global de los elementos que componen la solución y de las relaciones que permiten la interacción entre el usuario y el núcleo del sistema RAG desarrollado.

En segundo lugar, detallaremos el flujo de funcionamiento del sistema RAG, en el que se describe el proceso de almacenamiento, recuperación y uso de información relevante para generar respuestas contextualizadas en función de las consultas del usuario. Este nivel se centra en la secuencia de operaciones que tienen lugar desde que llega una consulta del usuario hasta que se genera la respuesta del modelo de lenguaje.

En la Figura 3-1: Arquitectura general del sistema, podemos ver que el sistema se organiza en dos bloques principales: frontend y backend.

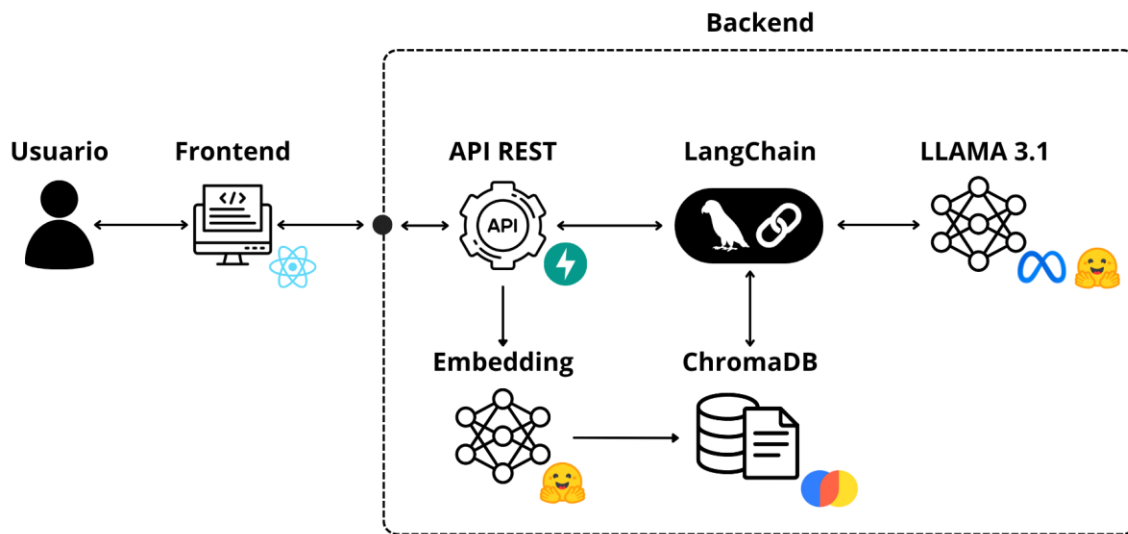


Figura 3-1: Arquitectura general del sistema

- **Frontend:** Desarrollado en React con TypeScript, este componente constituye la interfaz gráfica de usuario. Gracias a este, el usuario puede realizar consultas al modelo de lenguaje y visualizar sus respuestas en una interfaz de tipo chat. Además, incluye diversas funcionalidades, como borrar o subir nuevos documentos a la base de datos vectorial, modificar el system prompt del modelo Llama, o mantener y gestionar el historial de mensajes intercambiados, permitiendo conversaciones con memoria y evitando la pérdida de coherencia entre interacciones. La comunicación con el backend se realiza mediante peticiones HTTP a la API REST, siguiendo un formato JSON en los mensajes para garantizar compatibilidad y facilidad de integración con otros sistemas.
- **Backend:** Este elemento engloba al resto de componentes del sistema, que pueden ser desplegados de forma conjunta en un servidor distinto al del frontend:
  - **API REST:** Implementada con la librería FastAPI [16], esta API actúa como punto de entrada para las solicitudes del cliente, ya sean consultas al modelo de lenguaje o modificaciones en la base de datos vectorial. Este componente abstrae los servicios ofrecidos en un conjunto de URLs y métodos HTTP (endpoints).
  - **Modelo de embedding:** Este elemento se encarga de convertir texto a una representación vectorial para que pueda ser almacenado en una base de datos vectorial. Para ello, se utiliza el modelo sentence-transformers/all-mpnet-base-v2, un modelo de embedding de código abierto que puede descargarse desde Hugging Face para utilizarse de forma local [13]. Previamente, los documentos que se desean vectorizar deben someterse a un proceso de limpieza y segmentación, con el objetivo de maximizar el rendimiento del sistema RAG tanto en la búsqueda y recuperación de información como en la generación de respuestas del modelo de lenguaje.
  - **ChromaDB:** Esta base de datos vectorial es la encargada de almacenar las representaciones vectoriales de los documentos. Además, incluye un mecanismo de búsqueda por similitud que permite identificar y recuperar los fragmentos de texto más relevantes para la consulta de un usuario. En este proyecto, la métrica que se utiliza para realizar búsquedas es la similitud coseno.
  - **LangChain:** Este framework se utiliza para coordinar los procesos y los flujos de información

intercambiados entre los elementos del sistema RAG, desde la recuperación de contexto de la base de datos vectorial hasta su incorporación en la estructura del prompt siguiendo las especificaciones de Meta para el modelo Llama 3.1 Instruct [95]. LangChain actúa como un orquestador, implementando cadenas (chains) que encapsulan la lógica de recuperación y generación.

- **Modelo Llama 3.1:** Desplegado de forma local mediante la integración con Hugging Face, este modelo procesa el prompt enriquecido con el contexto recuperado y genera la respuesta para el usuario. Además, está optimizado con técnicas de cuantización en 8 bits para reducir el consumo de memoria y el tiempo de inferencia, permitiendo su ejecución en entornos con recursos de hardware limitados sin comprometer significativamente la calidad de la respuesta.

Este diseño modular se ha elegido por su capacidad para separar de forma clara las responsabilidades de cada componente, facilitando así el mantenimiento, la escalabilidad y la reutilización del sistema. La separación frontend-backend no solo permite el despliegue independiente de ambos componentes, sino también la integración del backend como un servicio autónomo accesible por múltiples clientes, abriendo la puerta a integraciones con aplicaciones externas. Además, la encapsulación de funcionalidades clave, como la gestión de la base de datos vectorial, la orquestación del flujo RAG o la inferencia del modelo Llama 3.1, dentro de componentes especializados garantiza una mayor flexibilidad para la incorporación de mejoras futuras, ya sea mediante la sustitución de modelos de lenguaje, la integración de nuevas técnicas de búsqueda o la adaptación a diferentes entornos de despliegue.

Por otro lado, en la Figura 3-2: Flujo de funcionamiento del sistema RAG, podemos ver que el proceso de recuperación y generación de respuestas se desarrolla en las siguientes etapas:

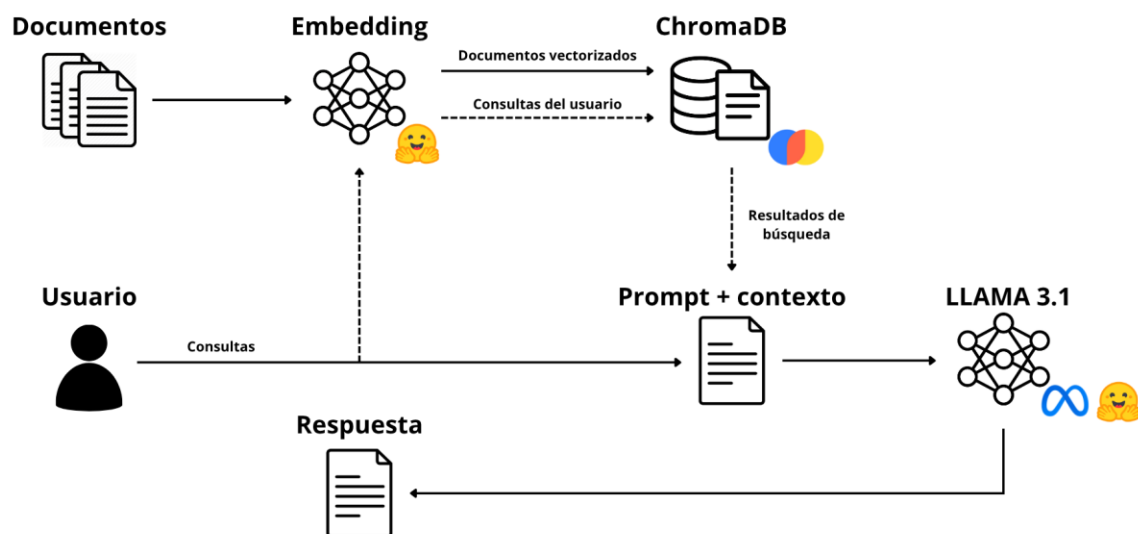


Figura 3-2: Flujo de funcionamiento del sistema RAG

1. **Preprocesamiento, vectorización y almacenamiento de documentos:** Los documentos que conforman la base de conocimiento, tras someterse a un proceso de limpieza y segmentación, son procesados mediante el modelo de embedding, generando representaciones vectoriales que son almacenadas en ChromaDB. En esta etapa debemos asegurarnos de que cada vector mantenga la semántica necesaria para ser identificado correctamente durante la fase de búsqueda.
2. **Consultas y recuperación de información de contexto:** Cuando se recibe una pregunta del usuario vía API, esta se vectoriza con el modelo de embedding, y mediante búsqueda por similitud se consulta la base de datos para recuperar los fragmentos más relevantes. El número de fragmentos recuperados y su relevancia influyen directamente en la calidad de la respuesta final, por lo que estos parámetros deben ajustarse cuidadosamente.

3. **Construcción del prompt:** Tras realizar la búsqueda en la base de datos, los fragmentos recuperados se integran junto con la consulta original en un prompt estructurado. Esta integración sigue el formato oficial de prompts de Meta para Llama 3.1, garantizando que el modelo interprete correctamente las instrucciones del system prompt, las consultas del usuario y los mensajes previos que se hayan podido intercambiar en la conversación.
4. **Generación de la respuesta del modelo de lenguaje:** El prompt formado por la consulta del usuario junto a información de contexto se envía al modelo Llama 3.1 para que genere la respuesta final.

En resumen, el flujo de funcionamiento del sistema RAG permite transformar una consulta del usuario en una respuesta enriquecida con información específica y contextualizada, siguiendo una secuencia optimizada de pasos que combinan recuperación y generación. Este diseño asegura que el modelo Llama no solo procese la pregunta del usuario de forma aislada, sino que lo haga apoyándose en una base de conocimiento previamente definida, incrementando la precisión y relevancia de sus respuestas. La integración de procesos como la búsqueda semántica, la construcción estructurada del prompt y la generación de respuestas utilizando LangChain garantiza una comunicación fluida entre los componentes, manteniendo además la flexibilidad necesaria para incorporar mejoras futuras en cualquiera de las fases del flujo RAG.

## 3.2 Ejecución del modelo Llama 3.1

El modelo Llama se ha desplegado y ejecutado de forma local mediante la librería Transformers de Hugging Face, utilizando una máquina virtual Ubuntu en WSL (Windows Subsystem for Linux). Gracias a WSL se puede desarrollar y validar el sistema directamente en un equipo cuyo sistema operativo es Windows, manteniendo las ventajas de un entorno de desarrollo Linux y la aceleración por GPU mediante CUDA. En concreto, el modelo que se ha utilizado es Llama 3.1 8B Instruct, que una vez descargado desde Hugging Face ocupa unos 16 GB en disco.

```
(LLM) sergio@OMEN16:~$ huggingface-cli scan-cache
REPO ID                               REPO TYPE SIZE ON DISK NB FILES LAST_ACCESSED LAST_MODIFIED REFS LOCAL PATH
-----
meta-llama/Llama-3.1-8B-Instruct      model      16.1G      10 2 hours ago  4 months ago  main /home/sergio/uct
sentence-transformers/all-mpnet-base-v2 model      438.7M     11 2 hours ago  4 months ago  main /home/sergio/et-base-v2

Done in 0.0s. Scanned 2 repo(s) for a total of 16.5G.
Got 2 warning(s) while scanning. Use -vvv to print details.
(LLM) sergio@OMEN16:~$
```

Figura 3-3: Tamaño en disco de los modelos descargados de Hugging Face

Durante el desarrollo y validación de este proyecto, se ha utilizado un portátil HP OMEN 16 [106] con las siguientes especificaciones técnicas:

- Procesador Intel Core i7-13700HX
- Tarjeta gráfica NVIDIA GeForce RTX 4080 (GDDR6 de 12 GB dedicada)
- Memoria RAM DDR5-5600 MHz 32 GB (2 x 16 GB)
- Almacenamiento SSD 2 TB PCIe Gen4 NVMe TLC M.2
- Sistema operativo Windows 11 Home
- Pantalla QHD de 16 pulgadas (hasta 2560 x 1440 píxeles), 240 Hz de frecuencia de refresco

Mediante el uso de técnicas de cuantización en 8 bits utilizando el módulo BitsAndBytes de Transformers, es posible ejecutar el modelo Llama utilizando la GPU con un consumo de memoria cercano a los 12 GB de memoria VRAM durante la inferencia, ajustándose así al límite físico de la tarjeta gráfica integrada en el equipo:

```

sergio@OMEN16:~$ nvidia-smi
Mon Aug 11 11:44:24 2025
+-----+
| NVIDIA-SMI 580.65.05                Driver Version: 580.88          CUDA Version: 13.0     |
+-----+-----+-----+
| GPU  Name                   Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf           Pwr:Usage/Cap |      Memory-Usage | GPU-Util  Compute M. |
|-----+-----+-----+
|   0   NVIDIA GeForce RTX 4080   On          | 00000000:01:00.0 Off |          N/A         |
| N/A   77C    P0              94W / 120W | 11976MiB / 12282MiB |    47%    Default   |
|-----+-----+-----+
+-----+
| Processes:                               GPU Memory |
|  GPU   GI    CI          PID    Type   Process name          Usage  |
|-----+-----+-----+
|   0   N/A   N/A              6     C   /python3.11          N/A   |
+-----+
sergio@OMEN16:~$

```

Figura 3-4: Uso de memoria de la GPU del modelo Llama 3.1 durante la inferencia

Esta cuantización en 8 bits se aplica a los pesos del modelo, reduciendo significativamente el uso de memoria con una leve pérdida de precisión, y manteniendo la compatibilidad con la inferencia acelerada por GPU. Por otro lado, para lograr un equilibrio entre rendimiento y calidad de generación, los cálculos que hace el modelo (principalmente operaciones matriciales) se realizan en precisión mixta con números flotantes de 16 bits. Además, el modelo Llama debe utilizarse junto a su tokenizador, que es el encargado de transformar el texto de entrada en una secuencia de tokens que el modelo puede procesar internamente. Del mismo modo, tras la inferencia, el tokenizador convierte la secuencia de tokens generados por el modelo nuevamente en texto legible para los humanos:

### Código 3-1: Carga del modelo Llama 3.1 8B Instruct (cuantizado) y su tokenizador

```

# Load quantized Llama model
quant_config = BitsAndBytesConfig(load_in_8bit=True, llm_int8_threshold=6.0)
llm = AutoModelForCausalLM.from_pretrained(
    LLAMA_MODEL_NAME,
    torch_dtype=torch.float16,
    device_map="auto",
    quantization_config=quant_config,
    token=HF_TOKEN
)

# Load tokenizer
tokenizer = AutoTokenizer.from_pretrained(LLAMA_MODEL_NAME, token=HF_TOKEN)

```

### 3.2.1 Formato del prompt

El modelo Llama utilizado en este proyecto emplea el formato de prompts especificado por Meta en sus modelos instruct, como ya se explicó en el apartado 2.4.3. Este formato de prompts permite definir el rol de cada mensaje en una conversación, que típicamente son: system, user y assistant.

Por lo general, en una conversación únicamente encontraremos un solo mensaje system, puesto que en él es donde se define el comportamiento e instrucciones que queremos que siga el modelo de lenguaje. Por ejemplo, un system prompt básico que se utiliza este proyecto es el siguiente:

#### Código 3-2: System prompt básico para el modelo de lenguaje

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>  
  
You are an AI assistant expert in digital communications<|eot_id|>
```

No obstante, el system prompt es un buen lugar donde aplicar técnicas de prompt engineering, como few shot learning o cadenas de razonamiento. Podemos definir una serie de pasos y restricciones que queremos que siga el modelo con el fin de que genere respuestas más útiles o enfocadas al caso de uso deseado, así como generar la respuesta en un formato y longitud específicos. Para este proyecto, se ha utilizado el siguiente mensaje como system prompt avanzado:

## Código 3-3: System prompt avanzado para el modelo de lenguaje

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are a helpful AI assistant specialized in digital communications.
The topics you can discuss include:
  - Fundamentals of digital communication systems.
  - Digital modulations and AWGN channels.
  - OFDM and frequency-selective channels.
  - Advanced channel models: multipath, Rayleigh and Rician fading,
    Doppler effect, etc.
  - MIMO systems and diversity in communication systems.
Sometimes the user may provide you with some context, but it will not
always be available.
You must follow this strict protocol for every user question:

0. You must ALWAYS remember that you are just an AI assistant and not a
human. You must never respond as if you were a human, regardless the
context.

1. Determine whether the question is related to digital communications or
the provided context:
  - If the user asks about politics, ethics, religion, etc,
    that is UNRELATED.

2. If UNRELATED to context or digital communications:
  - Explain briefly why you can't answer that question.

3. If RELATED to context or digital communications:
  a. Answer FIRST using context content.
  b. Only use training data to supplement missing context information.
  c. If context is insufficient, state what's missing.

4. For all responses:
  - Use well-formatted formulas and correct, consistent units.
  - Double-check all numerical calculations, especially when substituting
    in formulas.
  - Break solutions into clear, ordered steps.
  - Keep a professional tone.
  - Never invent facts.

<|eot_id|>
```

Como se puede ver en el mensaje, al principio podemos definir un contexto general para el modelo, y seguidamente definir un protocolo o algoritmo con las instrucciones y restricciones concretas que deseamos que se cumplan. Para el caso en el que estemos utilizando un sistema RAG, podemos indicar que en primer lugar se utilice la información obtenida de la base de datos para generar la respuesta, y que en caso de que esa información sea insuficiente, que el modelo genere la respuesta únicamente con su propio conocimiento. No obstante, para modelos pequeños, como es el caso de Llama 3.1 8B Instruct, no podemos garantizar que siempre se sigan las instrucciones que especifiquemos debido a las limitaciones del modelo, siendo muchas veces conveniente utilizar system prompts más sencillos. Para modelos mucho más grandes, ya sean de la familia Llama o de cualquier otro tipo, es conveniente utilizar system prompts complejos para aprovechar al máximo las capacidades de los modelos.

Gracias a las etiquetas user y assistant podemos definir los mensajes que corresponden al usuario y al modelo de lenguaje. En general, el prompt debe finalizar con las etiquetas `<|start_header_id|assistant<|end_header_id|>` para indicarle al modelo que le corresponde a él seguir la conversación. Por ejemplo, para preguntarle al modelo de lenguaje “¿Qué es OFDM?” utilizaríamos el siguiente prompt:

#### Código 3-4: Prompt de ejemplo con una única pregunta

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>  
  
You are an AI assistant expert in digital  
communications<|eot_id|><|start_header_id|>user<|end_header_id|>  
  
What is OFDM?<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

El uso de las etiquetas que indican los roles de cada mensaje y el fin de cada turno (`<|eot_id|>`) cobra más sentido cuando queremos hacer una pregunta manteniendo el historial de mensajes que hemos intercambiado con el modelo. Debido a que los modelos basados en Transformers no incluyen mecanismos de memoria por defecto, esta es la estrategia que ha seguido Meta para evitar que los modelos “olviden” el contexto previo que ha habido en una conversación:

**Código 3-5: Prompt de ejemplo con una nueva pregunta en una conversación multiterno**

```

<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are an AI assistant expert in digital
communications<|eot_id|><|start_header_id|>user<|end_header_id|>

Explain in three sentences what OFDM
is<|eot_id|><|start_header_id|>assistant<|end_header_id|>

OFDM (Orthogonal Frequency Division Multiplexing) is a digital modulation
technique that transmits data over multiple narrowband subcarriers, which
are orthogonal to each other, to achieve high spectral efficiency and
resistance to frequency-selective fading. In OFDM, the data is first
converted into the frequency domain using a Discrete Fourier Transform
(DFT), and then transmitted over the subcarriers, which are spaced at
regular intervals. The use of a cyclic prefix (CP) ensures that the
transmission over the channel yields a circular convolution, avoiding
inter-block interference (IBI) and allowing for efficient equalization at
the receiver.<|eot_id|><|start_header_id|>user<|end_header_id|>

Summarize your previous answer in one
sentence<|eot_id|><|start_header_id|>assistant<|end_header_id|>

```

Una forma sencilla de implementar este mecanismo de memoria es ir almacenando los mensajes intercambiados entre el usuario y el modelo (incluyendo el mensaje system al principio) durante la conversación, indicando en cada mensaje su respectivo rol. En cada nueva pregunta que hagamos al modelo, generaremos el prompt reconstruyendo la conversación que se ha mantenido, añadiendo las etiquetas necesarias para cada mensaje:

**Código 3-6: Construcción del prompt para el modelo de lenguaje a partir de una lista de mensajes**

```

# Build the prompt from messages (Llama 3.1 chat template)
prompt = "<|begin_of_text|>"
for msg in messages:
    role = msg.role
    content = msg.content.strip()
    prompt += f"<|start_header_id|>{role}<|end_header_id|>\n\n{content}\n\n"
prompt += "<|eot_id|><|start_header_id|>assistant<|end_header_id|>"

```

### 3.2.2 Generación de respuestas

Para la generación de respuestas, podemos utilizar un pipeline de tipo “text-generation” de la librería Transformers de Hugging Face. Este enfoque nos permite encapsular y simplificar el flujo de inferencia en un conjunto de métodos de alto nivel que serán ejecutados por un objeto de tipo pipeline, facilitando la configuración de los parámetros de generación y la posterior integración del modelo Llama en el flujo RAG. En el siguiente código podemos ver cómo crear este tipo de pipeline:

#### Código 3-7: Creación del pipeline generativo utilizando la librería Transformers

```
tokenizer.pad_token = tokenizer.eos_token

# Text generation pipeline
llm_pipe = pipeline(
    "text-generation",
    model=llm,
    tokenizer=tokenizer,
    temperature=sys.float_info.epsilon, # Use a very low temperature for
deterministic output
    max_new_tokens=128000,
    eos_token_id=tokenizer.eos_token_id,
    pad_token_id=tokenizer.pad_token_id,
    return_full_text=False
)

# Wrap the pipeline in HuggingFacePipeline (LangChain)
hf_pipeline = HuggingFacePipeline(pipeline=llm_pipe)
```

En primer lugar, en el pipeline debemos asignar como parámetros el modelo de lenguaje (en nuestro caso, el modelo Llama 3.1 cuantizado) y su tokenizador. Para este proyecto, en el que no deseamos que el modelo proporcione respuestas creativas sino más bien concretas y predecibles, nos interesa que nuestro modelo genere texto de forma prácticamente determinista. Para ello, debemos configurar el parámetro de temperatura del modelo a un valor lo más bajo posible para reducir la varianza en la generación de respuestas, como ya se explicó en el apartado 2.2.1.

Otro parámetro que podemos configurar es el número máximo de tokens que puede generar el modelo. Para tener una mayor versatilidad en la longitud de los textos que se generan, podemos configurar este parámetro a un valor alto, y posteriormente indicarle al modelo en el prompt la longitud máxima que deseamos en sus respuestas. Además, es necesario definir los tokens de fin de secuencia (EOS) y de relleno (PAD). El token EOS sirve para que el modelo sepa cuando debe parar la generación de texto, mientras que el token PAD se utiliza para rellenar secuencias al hacer procesamiento por batches. Dado que algunos modelos, como Llama, no incluyen un token de padding por defecto, en este proyecto se ha configurado el token PAD con el mismo identificador que el token EOS para evitar errores en la generación.

Finalmente, en la configuración del pipeline podemos especificar que no se incluya el texto del prompt original, quedándonos únicamente con el texto generado por el modelo.

Posteriormente, este pipeline podemos encapsularlo dentro de un objeto HuggingFacePipeline de LangChain para utilizar el modelo en flujos de trabajo más complejos, como los utilizados en el sistema RAG. Una vez que hemos creado el pipeline, podemos pasarle de forma sencilla nuestro prompt al modelo y comenzar la generación utilizando el método “invoke”:

#### Código 3-8: Invocación del pipeline generativo

```
# Generate the response
response = hf_pipeline.invoke(prompt)
```

### 3.2.3 Ejemplos de generación

A continuación, presentaremos algunos ejemplos de interacción reales con el modelo Llama 3.1. Para ello, se ha desarrollado un script en Python que permite mantener una conversación con el modelo de lenguaje directamente desde la línea de comandos, ofreciendo una experiencia similar a la de un chatbot interactivo. Además, el script admite dos modos opcionales de funcionamiento:

- **Modo RAG:** Permite recuperar información relevante desde una base de datos ChromaDB en función de la consulta del usuario. El funcionamiento del sistema RAG se explicará con detalle en los próximos apartados. Por simplicidad en la explicación, este modo se mantendrá desactivado en los ejemplos.
- **Modo con memoria:** Conserva el historial de conversación y lo incluye en cada nueva interacción con el modelo de lenguaje. Si se desactiva, cada interacción con el modelo será como una nueva conversación.

Este script ha sido una herramienta clave durante el desarrollo y pruebas del proyecto, ya que ha permitido evaluar rápidamente el comportamiento del modelo y ajustar parámetros antes de su integración definitiva en el sistema. Posteriormente, el código fue ampliado y adaptado para integrarlo en la API REST que forma parte de la solución final, la cual se describe en el apartado 3.6.

Tanto este script como el código completo de la API se encuentran disponibles en el siguiente repositorio GitHub:

[https://github.com/sergarlop2/chatbot\\_backend](https://github.com/sergarlop2/chatbot_backend)

#### 3.2.3.1 Ejemplo 1: Interacción básica

En este primer ejemplo, se presenta una interacción sencilla destinada a evaluar cómo responde el modelo a saludos y preguntas personales. Ante el saludo “Hola”, el modelo responde lo siguiente:

```
(llm) sergio@OMEN16:~/chatbot_backend$ python3 chat_cli.py
Loading checkpoint shards: 100%|██████████| 4/4 [00:21<00:00, 5.48s/it]
Device set to use cuda:0

Chat with Llama 3.1-8B-Instruct (type 'exit' to quit)

RAG: OFF | Memory: OFF

User: Hello

Assistant:

Hello. I'm here to assist you with any questions you have about digital
communications. What would you like to know?

User: |
```

Figura 3-5: Respuesta del modelo ante un saludo

Como podemos observar, el modelo responde de manera cordial y servicial, ofreciendo ayuda para resolver dudas sobre comunicaciones digitales. Este comportamiento se debe a las instrucciones incluidas en el system prompt, en las que se ha especificado que el modelo debe actuar como un asistente experto en sistemas de comunicaciones digitales. Si hubiéramos definido otro system prompt, como por ejemplo el de un asistente de atención al cliente, este comportamiento también se vería reflejado en las respuestas del modelo. Ante la pregunta: “¿Quién eres?”, el modelo responde de la siguiente manera:

```

User: Who are you?

Assistant:

I am a digital AI assistant specialized in digital communications. I can
provide information and answer questions related to digital communicati
on systems, including digital modulations, AWGN channels, OFDM, frequenc
y-selective channels, advanced channel models, MIMO systems, and diversi
ty in communication systems.

```

Figura 3-6: Respuesta del modelo ante una pregunta personal

### 3.2.3.2 Ejemplo 2: Explicación de conceptos

En este segundo ejemplo se evalúa la capacidad del modelo para dar una definición técnica clara y correcta. “¿Qué es la multiplexación?” es una pregunta breve y directa que no requiere un análisis profundo, pero permite verificar si el modelo es capaz de identificar el concepto en el contexto de las telecomunicaciones y dar una respuesta coherente:

```

User: What is multiplexing?

Assistant:

Multiplexing is a technique used in digital communication systems to transmit mul
tiple signals over a single communication channel. This is achieved by dividing t
he available bandwidth into multiple sub-channels, allowing multiple signals to b
e transmitted simultaneously.

There are several types of multiplexing techniques, including:

1. **Time Division Multiplexing (TDM)**: In TDM, the available bandwidth is divid
ed into time slots, and each signal is transmitted in a specific time slot.
2. **Frequency Division Multiplexing (FDM)**: In FDM, the available bandwidth is
divided into frequency bands, and each signal is transmitted in a specific freque
ncy band.
3. **Code Division Multiplexing (CDM)**: In CDM, each signal is assigned a unique
code, and the signals are transmitted simultaneously over the same bandwidth.
4. **Space Division Multiplexing (SDM)**: In SDM, multiple signals are transmitt
ed over multiple antennas or spatial channels.

Multiplexing allows for more efficient use of bandwidth and can increase the over
all data transmission rate. It is commonly used in various communication systems,
including telephone networks, cable television, and wireless communication syste
ms.

In the context of digital communication systems, multiplexing is often used in co
njunction with other techniques, such as modulation and demodulation, to enable r
eliable and efficient data transmission over various channels.

```

Figura 3-7: Explicación del modelo a conceptos técnicos

En este caso, el modelo explica correctamente el concepto de multiplexación, mencionando algunas técnicas como TDM, FDM, CDM y SDM, y ofreciendo una descripción sencilla de cada una. Cabe recordar que estamos usando el modelo Llama 3.1 8B sin habilitar el sistema RAG, por lo que este tipo de preguntas podrían responderse con mayor precisión. Por ejemplo, al inicio de su explicación, el modelo menciona que la multiplexación se consigue dividiendo el ancho de banda disponible en múltiples subcanales, cuando esto solo es cierto para FDM, lo que podría inducir a error. Probablemente, el modelo pretendía indicar que la multiplexación se logra dividiendo el canal en múltiples subcanales (en lugar de referirse al ancho de banda).

### 3.2.3.3 Ejemplo 3: Explicación de conceptos (español)

Como se ha podido intuir, el idioma utilizado en este proyecto para interactuar con el modelo es el inglés. No obstante, Llama 3.1 ha sido entrenado en múltiples idiomas y es capaz de entender y responder de forma competente en español:

```
User: ¿Qué es la multiplexación?

Assistant:

La multiplexación es un proceso en el que se combina múltiples señales de comunicación en una sola señal de comunicación que puede ser transmitida a través de un canal de comunicación compartido. Esto permite que varias señales se envíen simultáneamente a través de un solo canal, aumentando la eficiencia de la utilización del espectro de frecuencia y reduciendo la necesidad de múltiples canales de comunicación.

La multiplexación se utiliza comúnmente en sistemas de comunicación de datos, como en la transmisión de televisión, radio y redes de comunicación inalámbricas. Hay varios tipos de multiplexación, incluyendo:

1. Multiplexación en el tiempo (TDM, por sus siglas en inglés): se combina las señales en el tiempo, es decir, se envían las señales una detrás de la otra en el mismo canal.
2. Multiplexación en la frecuencia (FDM, por sus siglas en inglés): se combina las señales en la frecuencia, es decir, se envían las señales en diferentes frecuencias en el mismo canal.
3. Multiplexación en el código (CDM, por sus siglas en inglés): se combina las señales en el código, es decir, se envían las señales con diferentes códigos en el mismo canal.

La multiplexación es una técnica fundamental en la comunicación de datos y permite aumentar la capacidad de transmisión de información a través de un canal de comunicación compartido.
```

Figura 3-8: Respuesta del modelo en español

Como podemos ver, el mismo modelo con el mismo system prompt (en inglés) es capaz de identificar y responder en el idioma que utiliza el usuario. Además, aunque se haya formulado la misma pregunta en otro idioma, el texto generado no es una traducción literal de la respuesta del apartado anterior. Esto se debe al carácter generativo y probabilístico de estos modelos que, incluso ante la misma pregunta planteada de forma diferente, producen respuestas distintas. Aunque en este caso el modelo ha generado una respuesta adecuada, el idioma en el que mejor se desempeña Llama 3.1 es el inglés, ya que cerca del 53% de su dataset de fine-tuning está en este idioma, mientras que el dataset multilingüe representa apenas el 3% del total, como se explicó en el apartado 2.4.2.

### 3.2.3.4 Ejemplo 4: Comparación técnica

Este ejemplo se solicita al modelo una comparación breve entre dos modulaciones digitales: BPSK y QPSK, con el objetivo de evaluar su capacidad para comparar conceptos técnicos y resaltar diferencias clave sin recurrir a explicaciones demasiado extensas. Esta pregunta incrementa ligeramente el nivel de dificultad respecto a la anterior, ya que no solo es necesario conocer los conceptos teóricos, sino también aplicarlos en un análisis comparativo:

```

User: Explain briefly the main difference between BPSK and QPSK modulations
Assistant:
**BPSK (Binary Phase Shift Keying) and QPSK (Quadrature Phase Shift Keying)
Modulation Comparison**

BPSK and QPSK are both digital modulation techniques used to transmit infor
mation over a communication channel. The main difference between them lies
in the number of bits per symbol and the phase shift.

**BPSK:**

* Transmits 1 bit of information per symbol
* The phase of the carrier wave is shifted by 180° ( $\pi$  radians) to represent
a 1 or 0
* The constellation diagram is a circle with two points (I, Q) representing
the two possible phases

**QPSK:**

* Transmits 2 bits of information per symbol
* The phase of the carrier wave is shifted by 90° ( $\pi/2$  radians) to represen
t 00, 01, 10, or 11
* The constellation diagram is a square with four points (I, Q) representin
g the four possible phases

In summary, BPSK transmits 1 bit per symbol, while QPSK transmits 2 bits pe
r symbol, resulting in a higher data rate for QPSK. However, QPSK is more s
usceptible to phase noise and requires more complex receiver circuitry.

```

Figura 3-9: Comparación técnica realizada por el modelo

Observamos que el modelo ofrece una comparación correcta de ambas modulaciones, identificando adecuadamente los desplazamientos de fase de cada una y el número de bits transmitidos por símbolo.

### 3.2.3.5 Ejemplo 5: Resolución de problemas sencillos

A pesar de sus limitaciones, el modelo Llama 3.1 8B es capaz de resolver problemas matemáticos sencillos que no requieran un desarrollo complejo, sino la aplicación de fórmulas directas o sustituciones simples. En este caso, se plantea calcular el ancho de banda aproximado de un sistema OFDM con 180 subportadoras de 15 kHz:

```
User: Calculate the approximate bandwidth in an OFDM system with 180 subcarriers
of 15kHz

Assistant:

To calculate the approximate bandwidth in an OFDM system, we need to multiply th
e number of subcarriers by the subcarrier spacing.

The subcarrier spacing ( $\Delta f$ ) can be calculated as:

 $\Delta f = 1 / T$ , where T is the symbol duration.

However, we are given the subcarrier spacing ( $\Delta f = 15$  kHz) directly.

The bandwidth (B) of the OFDM system can be calculated as:

 $B = (\text{Number of subcarriers}) \times (\text{Subcarrier spacing})$ 
 $= 180 \times 15$  kHz
 $= 2700$  kHz

Therefore, the approximate bandwidth of the OFDM system is 2700 kHz or 2.7 MHz.
```

Figura 3-10: Resolución de problemas sencillos por parte del modelo

La principal dificultad de este problema es identificar que el ancho de banda aproximado de un sistema OFDM es:

$$B = N_{sc} \cdot W_{sc} \quad (3-1)$$

Donde  $B$  es el ancho de banda del sistema OFDM (en Hz),  $N_{sc}$  es el número de subportadoras y  $W_{sc}$  es el espaciado entre subportadoras (en Hz).

Aunque el modelo ha respondido correctamente a la pregunta, ha intentado incluir un cálculo innecesario para resolver el problema, incorporando el tiempo de símbolo como dato. Este tipo de situaciones evidencia el carácter generativo de los modelos de lenguaje, puesto que no son capaces de razonar, sino que generan la respuesta más probable ante un texto de entrada. Aunque estos modelos han sido entrenados para resolver algunos problemas matemáticos, es común que cometan errores de cálculo en operaciones aritméticas sencillas. Estos errores son más comunes en modelos pequeños, aunque pueden llegar a producirse en modelos más grandes. La forma de evitar estos errores es proporcionar al modelo herramientas para que realice estos cálculos (como si pudieran usar una calculadora), y generar la respuesta a partir de esos resultados. A pesar de que los modelos Llama tienen mecanismos para realizar llamadas a herramientas para realizar cálculos o ejecutar código [95], su implementación y uso quedan fuera del alcance de este proyecto.

### 3.2.3.6 Ejemplo 6: Conceptos avanzados

En este ejemplo se plantea una pregunta más específica para poner a prueba el conocimiento del modelo en conceptos avanzados de sistemas de comunicaciones. Ante la pregunta “¿Qué propiedad de las secuencias PN (Pseudorandom Noise) las hace adecuadas para la multiplexación en DS-SS?” obtenemos la siguiente respuesta:

```
User: What property of PN sequences makes them suitable for DS-SS multiplexing?
Assistant:
**Property of PN Sequences for DS-SS Multiplexing:**
PN (Pseudo-Noise) sequences are suitable for Direct Sequence Spread Spectrum (DS-SS) multiplexing due to their orthogonality property.
Orthogonality means that the cross-correlation between two different PN sequences is zero, i.e., the sequences are uncorrelated. This property allows multiple users to share the same frequency band without interference, as the signals can be easily separated at the receiver using correlation techniques.
Mathematically, this can be represented as:

$$R_{xy}(t) = \int x(t)y(t)dt = 0$$

where  $R_{xy}(t)$  is the cross-correlation between sequences  $x(t)$  and  $y(t)$ .
In DS-SS systems, each user is assigned a unique PN sequence, and the data signal is modulated onto this sequence. At the receiver, the received signal is correlated with the assigned PN sequence to extract the original data signal. The orthogonality property of PN sequences ensures that the signals from different users do not interfere with each other, allowing for simultaneous transmission and reception of multiple signals.
```

Figura 3-11: Conceptos avanzados explicados por el modelo

Vemos que el modelo ha identificado correctamente la ortogonalidad como la propiedad que hace que las secuencias PN sean adecuadas para los sistemas DS-SS. Además, ha incluido una definición precisa de ortogonalidad, explicando matemáticamente que dos secuencias son ortogonales cuando su correlación cruzada es cero. Gracias al system prompt utilizado, el modelo puede incluir fórmulas y demostraciones matemáticas sencillas para fundamentar su respuesta o explicar mejor algunos conceptos.

### 3.2.3.7 Ejemplo 7: Falta de contexto

En los ejemplos anteriores hemos comprobado que Llama 3.1 8B puede responder correctamente a preguntas básicas sobre sistemas de comunicaciones, e incluso a algunas más avanzadas que requieren un conocimiento especializado. Sin embargo, ciertas preguntas pueden resultar ambiguas, carecer del contexto necesario para ser respondidas correctamente, o simplemente el modelo no puede contestarlas a partir de la información aprendida durante la fase de preentrenamiento, llegando a provocar alucinaciones. Podemos ver esto reflejado en la siguiente pregunta, “Dado un proceso aleatorio estacionario  $X(t)$  con autocorrelación  $R_X(\tau)$ , ¿qué significa  $R_X(0)$ ?”:

```
User: Given a stationary random process  $X(t)$  with autocorrelation  $R_X(\tau)$ , what does  $R_X(0)$  mean?

Assistant:

**Related to digital communications**

 $R_X(\tau)$  is the autocorrelation function of the random process  $X(t)$ . The autocorrelation function is defined as:


$$R_X(\tau) = E[X(t) \cdot X(t + \tau)]$$


where  $E[\cdot]$  denotes the expected value.

 $R_X(0)$  is the autocorrelation of the process  $X(t)$  at  $\tau = 0$ .

To find  $R_X(0)$ , we substitute  $\tau = 0$  into the definition of the autocorrelation function:


$$R_X(0) = E[X(t) \cdot X(t + 0)]$$


$$= E[X(t) \cdot X(t)]$$


Since  $X(t)$  is a random process, the expected value of  $X(t)$  is the mean of the process, denoted by  $\mu_X$ . Therefore, we can write:


$$R_X(0) = E[X(t)] \cdot E[X(t)]$$


$$= \mu_X^2$$


So,  $R_X(0)$  is the square of the mean of the random process  $X(t)$ .
```

Figura 3-12: Respuesta incorrecta proporcionada por el modelo

En esta pregunta, aunque el modelo aplica inicialmente la definición general de autocorrelación y sustituye adecuadamente  $\tau = 0$ , comete un error al suponer que

$E[X(t) \cdot X(t)] = E[X(t)] \cdot E[X(t)] = E[X(t)]^2 = \mu_X^2$ , confundiendo la relación entre autocorrelación, varianza y media, puesto que la expresión correcta es  $E[X(t) \cdot X(t)] = \text{Var}[X(t)] + E[X(t)]^2$ . Además, el objetivo de la pregunta era llegar a la conclusión de que  $R_X(0)$  es la potencia media del proceso aleatorio, cosa que el modelo no logra hacer.

Por otro lado, en este otro caso se aprecia un claro ejemplo de alucinación en un problema que requiere aplicar una fórmula concreta. El modelo, al no conocer la fórmula o no saber identificarla, utiliza una fórmula inventada y la utiliza en todos los cálculos del problema, llegando a una solución incorrecta:

```
User: Consider a communication system with an input bit rate of  $R_b = 1$  Mbps ( $1 \cdot 10^6$  bits per second) and an excess bandwidth ( $\beta$ ) of 0.1. What is the required bandwidth when transmitting using BPSK? Also, calculate the bandwidth for QPSK, 16-QAM, and 64-QAM.

Assistant:

## Step 1: Determine the required bandwidth for BPSK
The required bandwidth for BPSK can be calculated using the formula:


$$B = 2R_b(1 + \beta)$$


where  $B$  is the required bandwidth,  $R_b$  is the input bit rate, and  $\beta$  is the excess bandwidth.

## Step 2: Plug in the values for BPSK
Given  $R_b = 1$  Mbps =  $1 \cdot 10^6$  bits per second and  $\beta = 0.1$ , we can calculate the required bandwidth for BPSK:


$$B_{\text{BPSK}} = 2 \cdot (1 \cdot 10^6) \cdot (1 + 0.1)$$


$$B_{\text{BPSK}} = 2 \cdot 10^6 \cdot 1.1$$


$$B_{\text{BPSK}} = 2.2 \cdot 10^6 \text{ Hz}$$


## Step 3: Determine the required bandwidth for QPSK
The required bandwidth for QPSK is twice that of BPSK, since QPSK transmits two bits per symbol.


$$B_{\text{QPSK}} = 2 \cdot B_{\text{BPSK}}$$


$$B_{\text{QPSK}} = 2 \cdot 2.2 \cdot 10^6$$


$$B_{\text{QPSK}} = 4.4 \cdot 10^6 \text{ Hz}$$


## Step 4: Determine the required bandwidth for 16-QAM
The required bandwidth for 16-QAM is approximately 4 times that of BPSK, since 16-QAM transmits 4 bits per symbol.


$$B_{\text{16QAM}} = 4 \cdot B_{\text{BPSK}}$$


$$B_{\text{16QAM}} = 4 \cdot 2.2 \cdot 10^6$$


$$B_{\text{16QAM}} = 8.8 \cdot 10^6 \text{ Hz}$$


## Step 5: Determine the required bandwidth for 64-QAM
The required bandwidth for 64-QAM is approximately 6 times that of BPSK, since 64-QAM transmits 6 bits per symbol.


$$B_{\text{64QAM}} = 6 \cdot B_{\text{BPSK}}$$


$$B_{\text{64QAM}} = 6 \cdot 2.2 \cdot 10^6$$


$$B_{\text{64QAM}} = 13.2 \cdot 10^6 \text{ Hz}$$


The final answer is:
- BPSK:  $2.2 \cdot 10^6$  Hz
- QPSK:  $4.4 \cdot 10^6$  Hz
- 16-QAM:  $8.8 \cdot 10^6$  Hz
- 64-QAM:  $13.2 \cdot 10^6$  Hz
```

Figura 3-13: Alucinaciones por parte del modelo

En este caso, el modelo debería haber utilizado la siguiente fórmula para calcular el ancho de banda de una modulación digital lineal:

$$B = \frac{(1 + \beta)R_b}{\log_2 M} \quad (3-2)$$

Donde  $B$  es el ancho de banda (en Hz),  $\beta$  es el factor de rolloff del filtro raíz de coseno alzado,  $M$  es el orden de la modulación y  $R_b$  es la tasa binaria (en bps).

Es interesante observar que el modelo intenta seguir una lógica parcialmente coherente pero errónea en su forma de resolver el problema. El modelo ha inferido una fórmula incorrecta para la modulación BPSK, y a partir de ella interpreta que el ancho de banda de cada modulación debe ir aumentando con respecto a la BPSK, puesto que el número de bits por símbolo va aumentando con el orden de la modulación. Según su lógica, el ancho de banda requerido aumenta con el orden de la modulación, lo cual es totalmente incorrecto, puesto que debe disminuir.

Una de las ventajas de utilizar un sistema RAG es que podemos proporcionar al modelo, de forma dinámica, las fórmulas necesarias para resolver estos problemas, evitando así que el modelo infiera información inventada e incorrecta que vaya arrastrando a lo largo de toda su respuesta.

### 3.3 Selección y preprocesamiento de documentos

La fase de selección y preprocesamiento de documentos constituye un paso crítico en el desarrollo del sistema RAG, ya que determina la calidad de la base de conocimiento sobre la que se construyen las respuestas del modelo de lenguaje. El objetivo es garantizar que los documentos integrados en la base de datos vectorial aporten información técnica relevante sobre sistemas de comunicaciones digitales, y que su estructura favorezca una recuperación eficiente y precisa del contexto.

#### 3.3.1 Selección de documentos

Para conformar la base de conocimiento, se han tomado los apuntes de clase de la asignatura “Sistemas de Comunicaciones” del Máster en Ingeniería de Telecomunicación de la Universidad de Sevilla, impartida por el profesor y catedrático Juan José Murillo Fuentes. Concretamente, se han utilizado cinco documentos en formato PDF:

- ChannelSCCI.pdf
- DigComSCCI.pdf
- SCCIModulation.pdf
- SCCIdiversity.pdf
- Unit3.SelectiveCh.pdf

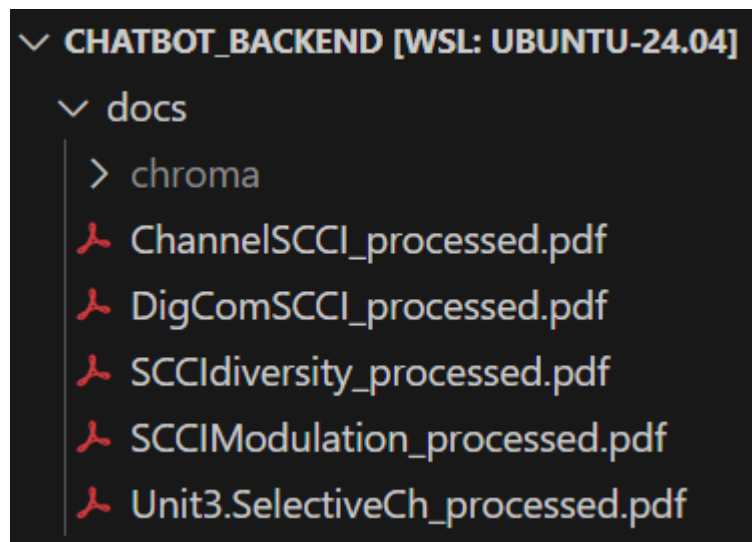


Figura 3-14: Documentos PDF utilizados en el sistema RAG

El contenido de estos documentos abarca desde los fundamentos básicos de las comunicaciones digitales, como codificación de fuente y de canal, hasta conceptos más avanzados, como canales selectivos en frecuencia, MIMO y técnicas de diversidad. De esta forma, se busca cubrir un amplio espectro de conceptos y técnicas utilizadas en el ámbito de las comunicaciones digitales, con el objetivo de que el modelo pueda proporcionar respuestas fundamentadas en dicho conocimiento.

#### 3.3.2 Preprocesamiento de documentos

Antes de vectorizar y almacenar los documentos en la base de datos, los documentos deben someterse a un proceso de depuración para eliminar información irrelevante que pueda degradar la calidad de las búsquedas semánticas. El filtrado realizado en este proyecto incluye:

- Portadas y páginas de presentación.
- Páginas delimitadoras de secciones
- Índices, tablas de contenidos y resúmenes que no aporten información técnica relevante o que ya se haya mencionado anteriormente.

- Páginas que contienen exclusivamente imágenes sin texto interpretable, dado que Llama 3.1 es incapaz de procesar imágenes al tratarse de un modelo de lenguaje no multimodal.

En este proyecto, la limpieza de documentos se realizó exclusivamente mediante la eliminación de páginas completas, sin llegar a modificar el contenido de aquellas que se conservaron. Una vez depurados, a estos documentos se les añadió el sufijo “processed” en el nombre para distinguirlos de los documentos originales, como se puede ver en la Figura 3-14: Documentos PDF utilizados en el sistema RAG.

Tras realizar la depuración de contenido, el siguiente paso en el preprocesamiento consiste en estructurar la información en unidades más pequeñas y manejables, conocidas como fragmentos o chunks. Este proceso de segmentación tiene como finalidad optimizar la búsqueda y recuperación de información en el flujo RAG, permitiendo que el modelo de lenguaje reciba únicamente los fragmentos de texto más relevantes, evitando la sobrecarga de información innecesaria. En caso contrario, una búsqueda semántica conllevaría recuperar documentos completos de la base de datos, lo cual añadiría ruido y empeoraría la calidad de las respuestas generadas por el modelo.

En este proyecto, se ha adoptado un esquema de segmentación en bloques de 5 páginas con un solapamiento del 20 % entre fragmentos consecutivos. Este solapamiento, equivalente a una página, se fundamenta en la necesidad de preservar la continuidad semántica en secciones del texto que puedan contener explicaciones distribuidas a lo largo de varias páginas. Sin este solapamiento, existiría el riesgo de que el corte entre dos fragmentos divida una definición, una deducción matemática o un razonamiento, lo cual reduciría la utilidad de los fragmentos recuperados en las consultas que se realicen.

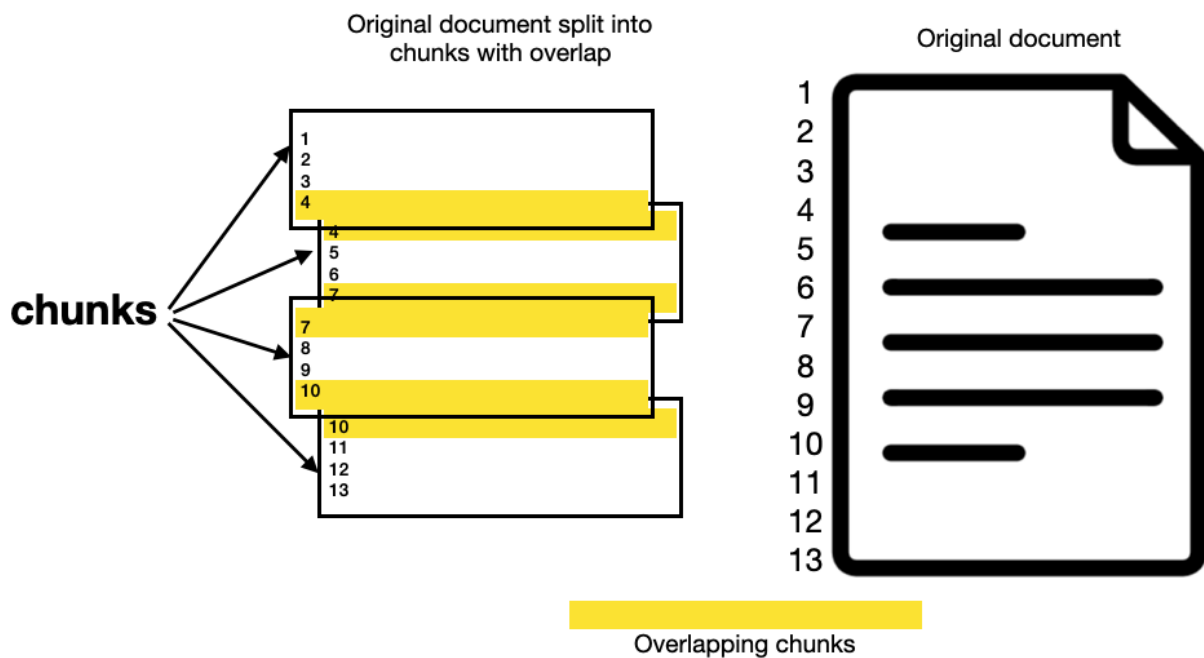


Figura 3-15: Segmentación de documentos en chunks con solapamiento [107]

Antes de realizar el proceso de segmentación, debemos cargar y agrupar las páginas de cada documento según su documento fuente, con el fin de evitar que los fragmentos combinen contenido procedente de distintos documentos. Esta agrupación se realiza empleando el campo “source” incluido en los metadatos de cada página, lo que permite vincular cada fragmento resultante con el PDF del que procede. Esto se debe a que la clase PyPDFLoader de LangChain carga todos los PDFs seleccionados y genera una lista de objetos que representan las páginas individuales junto con sus metadatos, sin hacer distinción entre documentos. Por lo tanto, debemos agrupar las páginas por documento fuente antes de aplicar el algoritmo de segmentación:

**Código 3-9: Agrupación de páginas por documento fuente**

```
# Load PDFs
docs = []
for pdf_file in pdf_files:
    loader = PyPDFLoader(pdf_file)
    docs.extend(loader.load())

# Group pages by source document
doc_groups = {}
for doc in docs:
    source = doc.metadata["source"]
    if source not in doc_groups:
        doc_groups[source] = []
    doc_groups[source].append(doc)
```

Una vez agrupadas las páginas por documento fuente, realizamos la segmentación de los documentos por chunks. Este proceso ordena las páginas por su número, toma bloques consecutivos de 5 páginas (CHUNK\_SIZE\_PAGES) y hace un solapamiento de 1 página (CHUNK\_OVERLAP\_PAGES) entre fragmentos. El texto de cada bloque de páginas se concatena, y se generan metadatos que incluyen el documento de origen, la página inicial, la página final y el número total de páginas que componen el fragmento:

**Código 3-10: Segmentación de documentos con solapamiento**

```
# Create chunks with overlap
split_docs = []
for source, pages in doc_groups.items():
    # Order pages and prepare for chunking
    sorted_pages = sorted(pages, key=lambda x: x.metadata["page"])
    total_pages = len(sorted_pages)
    start = 0

    while start < total_pages:
        end = start + CHUNK_SIZE_PAGES
        chunk_pages = sorted_pages[start:end]

        # Combine text and metadata
        combined_text = "\n\n".join([p.page_content for p in chunk_pages])
        metadata = {
            "source": source,
            "start_page": chunk_pages[0].metadata["page"],
            "end_page": chunk_pages[-1].metadata["page"],
            "total_chunk_pages": len(chunk_pages)
        }

        split_docs.append(Document(
            page_content=combined_text,
            metadata=metadata
        ))

        start += (CHUNK_SIZE_PAGES - CHUNK_OVERLAP_PAGES)
```

Finalmente, a cada fragmento generado se le añaden metadatos adicionales, como un identificador único (`chunk_id`) y la longitud del texto en número de caracteres (`chunk_length`):

### Código 3-11: Asignación de metadatos adicionales

```
# Add chunk-specific metadata
for idx, chunk in enumerate(split_docs):
    chunk.metadata.update({
        "chunk_id": idx,
        "chunk_length": len(chunk.page_content)
    })
```

Con esta estrategia de agrupación, segmentación y etiquetado mediante metadatos, nos aseguramos de que la información almacenada en la base de datos vectorial esté estructurada y referenciada de forma que no solo permita búsquedas semánticas precisas, sino que también proporcione trazabilidad sobre el origen de cada fragmento recuperado.

## 3.4 Vectorización y almacenamiento en ChromaDB

Una vez segmentados los documentos, el siguiente paso consiste en convertir el contenido textual de cada fragmento en representaciones numéricas llamadas *embeddings*, y almacenarlos en una base de datos vectorial para su posterior recuperación mediante búsquedas semánticas.

### 3.4.1 Generación de embeddings

Como ya se comentó previamente, para realizar la vectorización se ha empleado el modelo `sentence-transformers/all-mpnet-base-v2`, un modelo open source de Sentence Transformers ampliamente utilizado para tareas de búsqueda semántica. Este modelo transforma el texto de cada fragmento en un vector de 768 dimensiones, en el que la proximidad espacial entre vectores refleja la similitud semántica entre los textos originales [13].

El proceso de vectorización se lleva a cabo mediante la clase `HuggingFaceEmbeddings` de `LangChain`, especificando el nombre del modelo y activando la aceleración por GPU si está disponible. Además, se ha habilitado la normalización de embeddings para mejorar el rendimiento de las búsquedas basadas en la similitud coseno, asegurando que todos los vectores tengan norma unitaria y que la métrica dependa únicamente del ángulo entre los vectores:

### Código 3-12: Carga del modelo de embedding All-Mpnet-Base-V2

```
# Create embeddings and persist in Chroma
embedding = HuggingFaceEmbeddings(
    model_name=EMBEDDING_MODEL_NAME,
    model_kwargs={"device": str(torch.device("cuda" if
torch.cuda.is_available() else "cpu"))},
    encode_kwargs={"normalize_embeddings": True} # better for cosine
similarity
)
```

### 3.4.2 Almacenamiento en ChromaDB

Tras generar los vectores correspondientes a cada fragmento, estos se almacenan en la base de datos ChromaDB. En este proyecto, dicha base de datos se ha configurado para utilizar la similitud coseno como métrica de búsqueda (`"hnsw:space": "cosine"`), lo que implica que las búsquedas semánticas se realizan midiendo el ángulo entre los vectores en lugar de calcular la distancia euclídea. Esta elección resulta especialmente adecuada cuando

se trabaja con embeddings normalizados, ya que la similitud coseno proporciona una medida más robusta frente a variaciones de escala en los datos, lo que la hace más apropiada que la distancia euclídea.

### Código 3-13: Almacenamiento de embeddings en ChromaDB

```
vectordb = Chroma.from_documents(  
    documents=split_docs,  
    embedding=embedding,  
    persist_directory=PERSIST_DIR,  
    collection_metadata={"hsw:space": "cosine"}  
)
```

Gracias a este proceso de vectorización y almacenamiento, cuando el sistema recibe una consulta del usuario, esta se transforma en un vector en el mismo espacio de embeddings que los documentos. Luego, ChromaDB compara el vector de la consulta con los vectores almacenados y recupera aquellos con mayor similitud coseno, es decir, aquellos que están más próximos en el espacio semántico.

## 3.5 Pipeline RAG con LangChain

El pipeline RAG constituye el núcleo del sistema desarrollado para crear un asistente experto en sistemas de comunicaciones digitales. Su implementación se ha llevado a cabo utilizando el framework LangChain, que actúa como un orquestador de los distintos componentes implicados en el flujo generativo, desde la recuperación de información relevante de la base de datos hasta la generación de texto mediante el modelo Llama 3.1.

El primer paso dentro del pipeline RAG consiste en configurar el recuperador de fragmentos de la base de datos ChromaDB. En este contexto, se utiliza la estrategia top-k, que selecciona los k fragmentos más relevantes en función de su similitud semántica con la pregunta planteada por el usuario:

### Código 3-14: Configuración del retriever de ChromaDB

```
vectordb = Chroma(  
    persist_directory=PERSIST_DIR,  
    embedding_function=embedding  
)  
retriever = vectordb.as_retriever(search_kwargs={"k": 5})
```

En este fragmento de código, se carga la base de datos en memoria y se crea un objeto “retriever” capaz de realizar búsquedas semánticas sobre los vectores almacenados. El parámetro k=5 indica que, para cada consulta del usuario, se recuperarán los cinco fragmentos más relevantes, garantizando así que la información de contexto sea suficiente para enriquecer la generación de la respuesta. Para este proyecto, se ha tomado k=5 porque representa un equilibrio adecuado entre relevancia y coste computacional, ya que un valor inferior a cinco puede no aportar suficiente contexto para consultas complejas, mientras que un valor mayor aumentaría innecesariamente la latencia del sistema y el consumo de memoria al proporcionar demasiado contexto redundante.

Una vez configurado el recuperador, el siguiente paso en el pipeline es la incorporación del contexto al mensaje que se envía al modelo Llama 3.1. Para ello, cada vez que el usuario hace una consulta, el sistema invoca al recuperador con el texto de entrada, obtiene los fragmentos más relevantes, y organiza la información y la consulta para que sean fáciles de interpretar para el modelo:

**Código 3-15: Recuperación de contexto y construcción del mensaje del usuario**

```
# If RAG is enabled, retrieve context from the vector database
if args.rag:
    docs = retriever.invoke(user_input)
    context = "\n\n".join([d.page_content for d in docs])
    user_message = f"Context: ```{context}```\n\nQuestion:
```{user_input}```\n\nAnswer:"
else:
    user_message = user_input
```

En este bloque de código, la lógica condicional determina si el sistema debe operar en modo RAG o si el modelo únicamente recibe la consulta del usuario:

1. Si el RAG está habilitado, el recuperador (retriever) busca en la base de datos los fragmentos más relevantes en función de la consulta (user\_input).
2. A continuación, dichos fragmentos se concatenan en una variable llamada “context”, separándolos por saltos de línea dobles para mantener la legibilidad.
3. Finalmente, se construye un mensaje estructurado que combina tanto la pregunta del usuario como el contexto recuperado, de modo que el modelo disponga de la información necesaria para generar una respuesta fundamentada.

Este formato usado en el mensaje del usuario, inspirado en las recomendaciones de uso de Llama 3.1 Instruct, permite distinguir de manera clara entre:

- El contexto (Context), que incluye los fragmentos recuperados de la base de datos.
- La pregunta (Question), que contiene la consulta original del usuario.
- La respuesta (Answer), que indica al modelo el inicio de la generación de la respuesta.

Una vez generado el mensaje del usuario (user\_message), este se incluye en el prompt que se envía al modelo Llama 3.1, como ya se explicó en el apartado 3.2.1. El modelo, al recibir tanto la consulta como el contexto, produce una respuesta más precisa y ajustada al ámbito de las comunicaciones digitales. De esta manera, el pipeline RAG asegura que la generación de texto esté siempre respaldada por conocimiento técnico relevante, evitando respuestas vagas o fuera de contexto.

Podemos ver la mejora que supone el uso de RAG en el siguiente ejemplo, en el que pedimos al modelo que resuelva el mismo problema que se planteó en el apartado 3.2.3.7, pero esta vez con el sistema RAG habilitado:

```

RAG: ON | Memory: OFF

User: Consider a communication system with an input bit rate of  $R_b = 1 \text{ Mbps}$  ( $1 \cdot 10^6$  bits per second) and an excess bandwidth ( $\beta$ ) of 0.1. What is the required bandwidth when transmitting using BPSK? Also, calculate the bandwidth for QPSK, 16-QAM, and 64-QAM.

Assistant:

To calculate the required bandwidth for each modulation scheme, we can use the formula:


$$B = (1 + \beta)R_b / \log_2 M$$


where  $B$  is the bandwidth,  $\beta$  is the excess bandwidth,  $R_b$  is the input bit rate, and  $M$  is the number of constellation points.

For BPSK,  $M = 2$ , so:


$$\begin{aligned} B_{\text{BPSK}} &= (1 + 0.1) * 1e6 / \log_2 2 \\ &= 1.1 * 1e6 / 1 \\ &= 1.1 * 10^6 \text{ Hz} \end{aligned}$$


For QPSK,  $M = 4$ , so:


$$\begin{aligned} B_{\text{QPSK}} &= (1 + 0.1) * 1e6 / \log_2 4 \\ &= 1.1 * 1e6 / 2 \\ &= 0.55 * 10^6 \text{ Hz} \end{aligned}$$


For 16-QAM,  $M = 16$ , so:


$$\begin{aligned} B_{\text{16QAM}} &= (1 + 0.1) * 1e6 / \log_2 16 \\ &= 1.1 * 1e6 / 4 \\ &= 0.275 * 10^6 \text{ Hz} \end{aligned}$$


For 64-QAM,  $M = 64$ , so:


$$\begin{aligned} B_{\text{64QAM}} &= (1 + 0.1) * 1e6 / \log_2 64 \\ &= 1.1 * 1e6 / 6 \\ &= 0.1833 * 10^6 \text{ Hz} \end{aligned}$$


Therefore, the required bandwidths for each modulation scheme are:

* BPSK: 1.1 MHz
* QPSK: 0.55 MHz
* 16-QAM: 0.275 MHz
* 64-QAM: 0.1833 MHz

```

Figura 3-16: Respuesta correcta por parte del modelo gracias al uso de RAG

Gracias a la información recuperada de la base de datos, esta vez el modelo utiliza la fórmula correcta para resolver el problema, en vez de inferirla por su propia cuenta. Al enriquecer los prompts utilizando información procedente de fuentes actualizadas y verificadas, podemos aumentar la precisión de las respuestas del modelo y reducir el riesgo de sufrir alucinaciones.

### 3.6 Implementación de la API REST

La API REST es el componente que expone la funcionalidad del sistema desarrollado en este proyecto, permitiendo tanto la interacción con el modelo de lenguaje como la gestión de los documentos almacenados en el sistema RAG. Para su implementación se ha utilizado FastAPI [16], un framework ligero y eficiente que facilita la definición y desarrollo de endpoints de forma sencilla. En este contexto, un endpoint puede entenderse como la combinación de una URL, que identifica un recurso o funcionalidad específica de la API, y un método HTTP, que determina la operación a realizar sobre dicho recurso (consulta, inserción, actualización o eliminación). Además, FastAPI permite generar automáticamente documentación que sigue el estándar OpenAPI [108], utilizando para ello interfaces como Swagger o ReDoc.

Gracias a esta API, el frontend o cualquier otro sistema externo puede comunicarse con el backend enviando peticiones HTTP con mensajes en formato JSON. En dichos mensajes se ha empleado un esquema similar al que utiliza OpenAI en sus modelos GPT [109], con el objetivo de seguir un formato lo más estándar posible y permitir una compatibilidad futura con herramientas que consuman este tipo de APIs.

El código de la API REST se encuentra disponible en el siguiente repositorio GitHub:

[https://github.com/sergarlop2/chatbot\\_backend](https://github.com/sergarlop2/chatbot_backend)

### 3.6.1 Endpoints de la API

En la siguiente figura podemos ver parte de la documentación generada automáticamente con Swagger, en la que se muestran los endpoints expuestos por la API:

**FastAPI** 0.1.0 OAS 3.1  
/openapi.json

**Models** ^

- GET** /models List available models
- GET** /models/{model\_id} Get model details

**Chat** ^

- POST** /chat/completions Send messages to the language model

**Documents** ^

- GET** /docs List stored PDF documents
- PUT** /docs Upload and vectorize a PDF document
- GET** /docs/{filename} Get a PDF document
- DELETE** /docs/{filename} Delete a PDF document and its vectors

Figura 3-17: Documentación Swagger de la API REST

Podemos organizar los endpoints de la API en grupos distintos en función de las funcionalidades que ofrecen:

- **Models**
  - **GET /models:** Devuelve la lista de modelos disponibles en el sistema. Actualmente, solo se utiliza el modelo Llama 3.1 8B Instruct, pero se podría ampliar el sistema con más modelos Llama o de otras familias, como Mistral o Gemma.
  - **GET /models/{model\_id}:** Permite obtener información de un modelo concreto identificado por el parámetro “model\_id”. En el caso del modelo Llama utilizado en este proyecto, su identificador es “llama-3.1-8b-instruct”.
- **Chat**
  - **POST /chat/completions:** Permite enviar mensajes al modelo de lenguaje y recibir la respuesta generada. La solicitud se realiza mediante un JSON que contiene una lista de mensajes, donde cada mensaje especifica un rol (“system”, “user” o “assistant”) y su contenido textual. Además, la solicitud incluye una opción para habilitar el modo RAG del sistema, mejorando así la precisión y relevancia de la respuesta generada.
- **Documents**
  - **GET /docs:** Devuelve la lista de documentos PDF almacenados en el sistema RAG.

- **PUT /docs:** Permite subir un nuevo documento PDF al sistema RAG, que será procesado, dividido en fragmentos y vectorizado para futuras consultas.
- **GET /docs/{filename}:** Permite descargar un documento PDF almacenado en el sistema, identificado por el parámetro “filename”.
- **DELETE /docs/{filename}:** Elimina un documento PDF y sus vectores asociados de la base de datos vectorial. Para localizar el documento se utiliza el parámetro “filename”.

Estos endpoints permiten estructurar la funcionalidad del sistema de manera modular y accesible, actuando como servicios que pueden ser utilizados tanto por el frontend desarrollado en este proyecto como por cualquier otro sistema externo. Además, cada grupo de endpoints cumple un rol específico: los endpoints del grupo “Models” facilitan la obtención de información sobre los modelos disponibles, lo que permite seleccionar de manera dinámica el modelo a utilizar según las necesidades de la aplicación o del usuario; los endpoints del grupo “Chat” proporcionan un canal para interactuar con el modelo de lenguaje, combinando la generación de respuestas con la capacidad de recuperación de información del sistema RAG; y los endpoints del grupo “Documents” permiten gestionar la base de conocimiento, garantizando que la información utilizada por el modelo esté siempre actualizada.

### 3.6.2 Interacción con el modelo de lenguaje

La interacción con el modelo de lenguaje se articula a través de los endpoints agrupados en las categorías “Models” y “Chat”. El primer lugar, podemos descubrir qué modelos están disponibles y obtener cierta información sobre ellos. Para hacer las llamadas a la API de forma sencilla podemos utilizar Postman [110], una herramienta que permite enviar peticiones HTTP y visualizar sus respuestas de manera estructurada. En la siguiente figura se muestra un ejemplo de la respuesta obtenida al realizar una petición GET a /models:

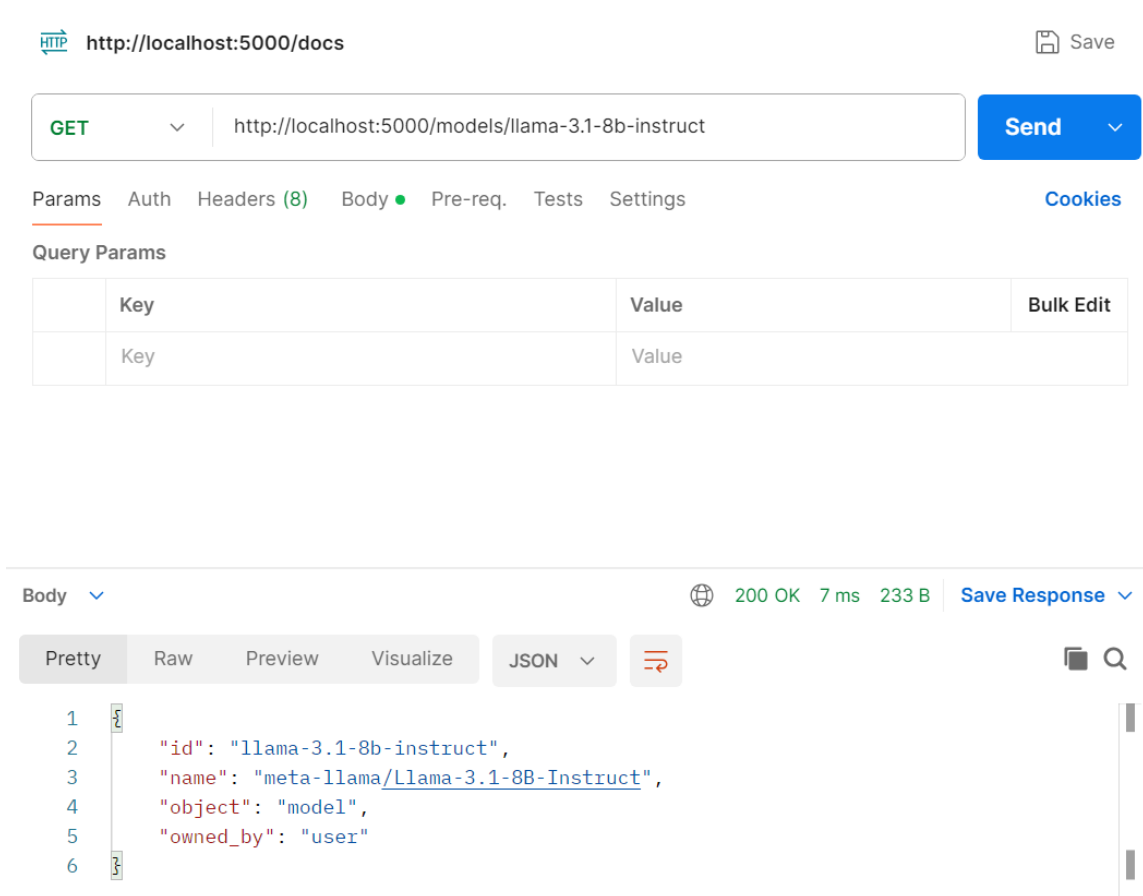
The screenshot shows the Postman interface for a GET request to `http://localhost:5000/models`. The response status is 200 OK, with a response time of 37 ms and a size of 260 B. The response body is displayed in JSON format:

```
1  {
2    "object": "list",
3    "data": [
4      {
5        "id": "llama-3.1-8b-instruct",
6        "name": "meta-llama/Llama-3.1-8B-Instruct",
7        "object": "model",
8        "owned_by": "user"
9      }
10   ]
11 }
```

Figura 3-18: Petición GET a /models

El resultado obtenido es un objeto JSON que contiene una lista con los modelos de lenguaje disponibles en el sistema, incluyendo información de cada uno de ellos. Como se comentó anteriormente, el único modelo disponible es Llama 3.1 8B Instruct, pero se podrían llegar a incluir más modelos en un futuro.

De forma similar, podemos obtener los detalles de un modelo en concreto con una petición GET a `/models/{model_id}`:



The screenshot shows a web browser's developer tools interface. At the top, the URL bar shows `http://localhost:5000/docs`. Below it, the network tab is active, showing a GET request to `http://localhost:5000/models/llama-3.1-8b-instruct`. The response is a JSON object with the following structure:

```

{
  "id": "llama-3.1-8b-instruct",
  "name": "meta-llama/Llama-3.1-8B-Instruct",
  "object": "model",
  "owned_by": "user"
}

```

Figura 3-19: Petición GET a `/models/{model_id}`

Una vez identificado el modelo, podemos enviarle mensajes haciendo una petición POST a `/chat/completions`. Para ello, el cuerpo de las peticiones HTTP que hagamos deben seguir el siguiente formato JSON:

### Código 3-16: Formato del JSON utilizado en las peticiones POST a `/chat/completions`

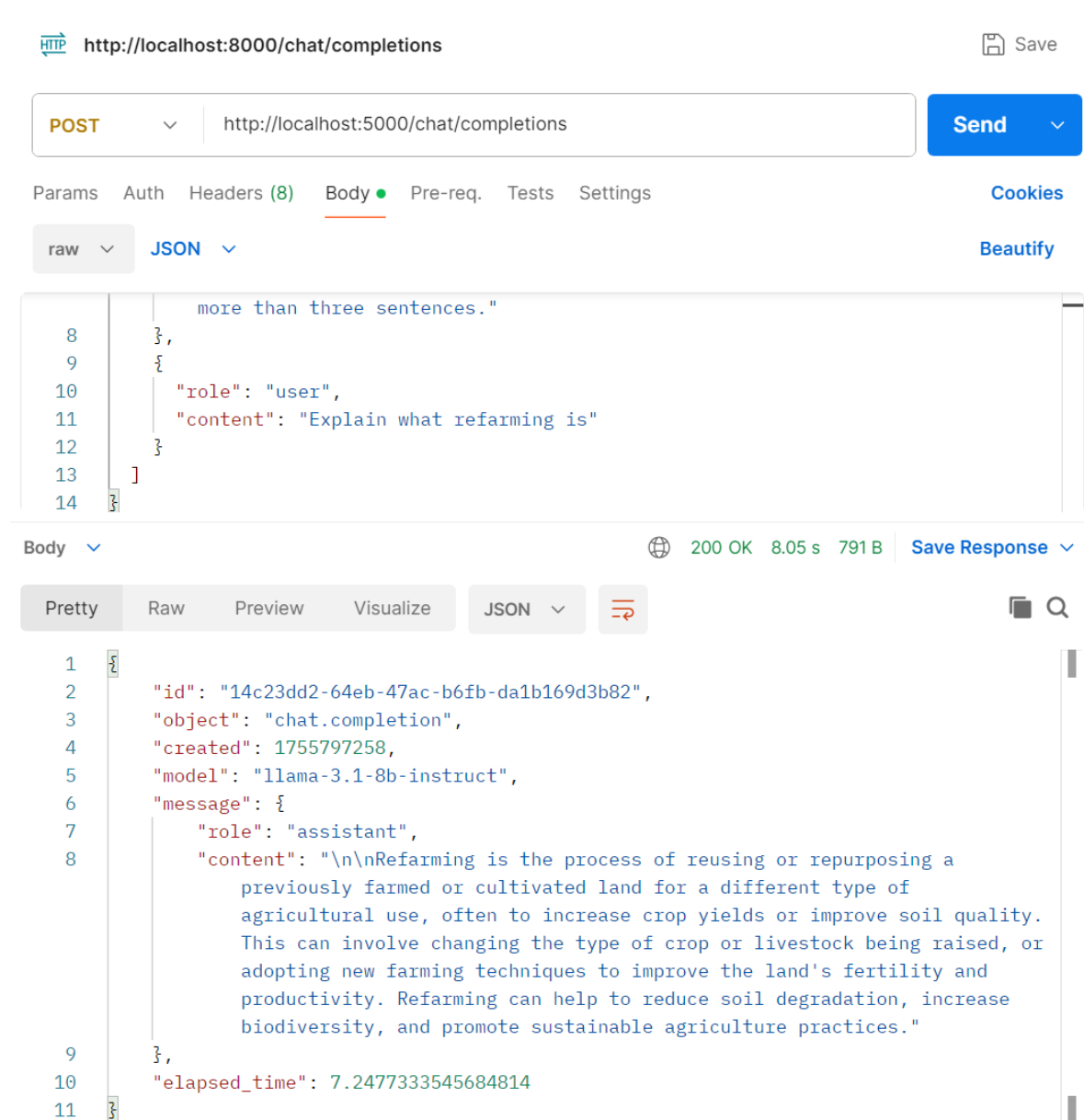
```

{
  "model": "llama-3.1-8b-instruct",
  "use_rag": false,
  "messages": [
    {
      "role": "system",
      "content": "You are an expert AI assistant. Briefly explain your answers in no more than three sentences."
    },
    {
      "role": "user",
      "content": "Explain what refarming is"
    }
  ]
}

```

En dicho objeto JSON debemos especificar el modelo de lenguaje que deseamos utilizar, si deseamos habilitar el sistema RAG, y finalmente, incluir la lista de mensajes que se le pasarán al modelo. Como ya se explicó en el apartado 3.2.1, los mensajes pueden tener tres roles diferentes: `system`, que nos permite especificar el comportamiento y las instrucciones generales que debe seguir el modelo; `user`, que representa las preguntas o solicitudes del usuario; y `assistant`, que corresponde a las respuestas generadas por el propio modelo de lenguaje.

En este caso, le pediremos al modelo que defina el concepto `refarming`, es decir, la redistribución de bandas de frecuencia previamente asignadas a determinados servicios de telecomunicaciones, con el fin de reasignarlas a tecnologías más modernas y eficientes. Además, deshabilitaremos el uso de RAG para que el modelo genere la respuesta sin utilizar la información almacenada en la base de datos vectorial. Al hacer la petición a la API siguiendo este formato JSON, obtenemos la siguiente respuesta:



The screenshot shows a REST client interface. At the top, the URL is `http://localhost:8000/chat/completions`. The request method is `POST` and the target URL is `http://localhost:5000/chat/completions`. The request body is in JSON format, containing an array with one object: `{ "role": "user", "content": "Explain what refarming is" }`. The response status is `200 OK`, with a response time of `8.05 s` and a size of `791 B`. The response body is in JSON format, containing an object with fields: `"id": "14c23dd2-64eb-47ac-b6fb-da1b169d3b82"`, `"object": "chat.completion"`, `"created": 1755797258`, `"model": "llama-3.1-8b-instruct"`, `"message": { "role": "assistant", "content": "\n\nRefarming is the process of reusing or repurposing a previously farmed or cultivated land for a different type of agricultural use, often to increase crop yields or improve soil quality. This can involve changing the type of crop or livestock being raised, or adopting new farming techniques to improve the land's fertility and productivity. Refarming can help to reduce soil degradation, increase biodiversity, and promote sustainable agriculture practices." }`, and `"elapsed_time": 7.2477333545684814`.

Figura 3-20: Petición POST a `/chat/completions` sin RAG habilitado

En la respuesta de la API podemos ver que se devuelve un objeto JSON con varios campos que aportan información sobre la generación realizada. En primer lugar, el campo `"id"` identifica de forma única la petición, permitiendo su trazabilidad dentro del sistema. A continuación, el campo `"object"` indica el tipo de objeto incluido en la respuesta de la API, en este caso `"chat.completion"`, que hace referencia a una interacción conversacional. Por otro lado, el campo `"created"` representa una marca temporal en formato Unix, que señala el instante de tiempo en el que se generó la respuesta. Seguidamente, el campo `"model"` especifica el modelo de lenguaje utilizado para generar la respuesta.

Dentro del objeto “message” es donde se encuentra la respuesta generada por el modelo de lenguaje, denotada por el rol “assistant”. En este caso, al no habilitar el sistema RAG, el modelo ofrece una definición del concepto refarming relacionada con la agricultura, en lugar de referirse a su significado en el ámbito de las telecomunicaciones. No obstante, ha sido capaz de seguir las instrucciones incluidas en el mensaje “system”, proporcionando una breve respuesta sin sobrepasar los límites de extensión especificados.

Por último, el campo “elapsed\_time” refleja el tiempo empleado en segundos para producir la respuesta, lo que nos permite medir el rendimiento y eficiencia del sistema en cada interacción.

Para evaluar el impacto que supone la inclusión de información de contexto en los prompts del modelo, podemos realizar la misma consulta sobre el refarming, pero esta vez habilitando el sistema RAG:

```

1  {}
2  "id": "8dd6f4cb-2007-4553-ad1b-af5fd3aae295",
3  "object": "chat.completion",
4  "created": 1755797322,
5  "model": "llama-3.1-8b-instruct",
6  "message": {
7    "role": "assistant",
8    "content": "\n\nRefarming is a process in wireless communication systems
           where a previously allocated frequency band is re-allocated to a
           different type of service or technology, often to improve spectrum
           efficiency and utilization. This can involve reconfiguring the existing
           infrastructure to support a new type of modulation, coding scheme, or
           multiple access technique, allowing for more efficient use of the
           available spectrum."
9  },
10 "elapsed_time": 9.024468183517456,
11 "sources": [
12   {
13     "chunk_id": 50,
14     "chunk_length": 3830,
15     "end_page": 16,
16     "source": "docs/DigComSCCI_processed.pdf",
17     "start_page": 12,
18     "total_chunk_pages": 5
19   },
20   {
21     "chunk_id": 48,
22     "chunk_length": 3383,

```

Figura 3-21: Petición POST a /chat/completions con RAG habilitado

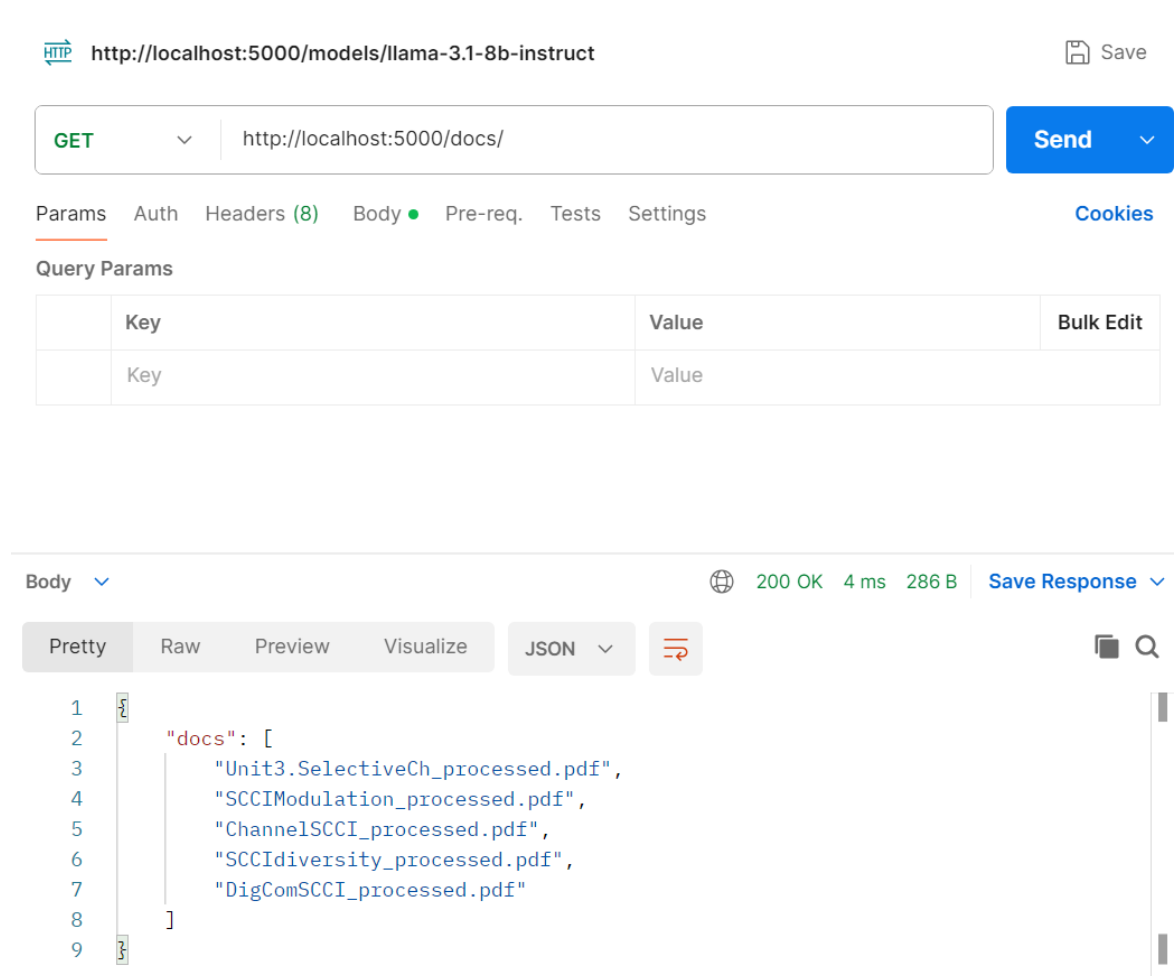
Como se aprecia en la figura Figura 3-21: Petición POST a /chat/completions con RAG habilitado, esta vez la respuesta sí está enmarcada en el dominio de las telecomunicaciones. Además, el JSON incluye una lista con las fuentes (sources) que el modelo ha utilizado para generar su respuesta. En cada de ellas se incluye una serie de metadatos, como el identificador del fragmento (chunk\_id) almacenado en la base de datos, la longitud de dicho fragmento (chunk\_length) en número de caracteres, las páginas de inicio (start\_page) y fin (end\_page) del fragmento en el documento original, el número total de páginas del fragmento (total\_chunk\_pages), y el nombre del documento del que se ha obtenido el fragmento (source).

Por otro lado, podemos ver que el tiempo de generación de la respuesta ha sido ligeramente superior en esta petición, debido a que el modelo tiene que procesar los fragmentos de texto recuperados de la base de datos para elaborar una respuesta coherente y contextualizada.

### 3.6.3 Gestión de documentos del sistema RAG

Los endpoints agrupados en la categoría “Documents” permiten realizar operaciones básicas de listado, descarga, subida y eliminación de documentos en el sistema RAG. Gracias a estas funcionalidades, se facilita la administración de la información almacenada en la base de datos vectorial, asegurando que los datos disponibles estén siempre alineados con las necesidades del usuario o de la aplicación.

En primer lugar, con la petición GET /docs se pueden listar los documentos almacenados en el sistema RAG. Esta operación devuelve un objeto JSON con los nombres de los archivos disponibles, lo que permite al usuario obtener una visión general del estado de la base de datos vectorial:



The screenshot shows a REST client interface with the following details:

- URL: `http://localhost:5000/models/llama-3.1-8b-instruct`
- Method: `GET`
- Path: `http://localhost:5000/docs/`
- Response Status: `200 OK`, `4 ms`, `286 B`
- Response Body (JSON):

```
1 {  
2   "docs": [  
3     "Unit3.SelectiveCh_processed.pdf",  
4     "SCCIModulation_processed.pdf",  
5     "ChannelSCCI_processed.pdf",  
6     "SCCIdiversity_processed.pdf",  
7     "DigComSCCI_processed.pdf"  
8   ]  
9 }
```

Figura 3-22: Petición GET a /docs

Como podemos ver en la Figura 3-22: Petición GET a /docs, en el sistema RAG están almacenados los documentos detallados en el apartado 3.3.1.

En caso de que necesitemos consultar un documento en específico, podemos hacer una petición GET a `/docs/{filename}`, indicando en el parámetro “filename” el nombre del documento que queremos descargar:

The screenshot shows a web browser's developer tools interface. At the top, the URL bar shows `http://localhost:5000/models/llama-3.1-8b-instruct`. Below it, the request method is set to `GET` and the request URL is `http://localhost:5000/docs/Unit3.SelectiveCh_processed.pdf`. The response status is `200 OK` with a response time of `28 ms` and a size of `2.21 MB`. The response body is displayed as a slide from a presentation, titled `Inter-Symbol Interference (ISI)`. The slide content includes:

- If the channel impulse response is not  $\delta(t)$  but we have some memory

$$h(t) = \sum_{l=1}^L a_l \delta(t - \tau_l)$$

at reception we have, apart from noise, several delayed copies of the transmitted signal superimposed, each one multiplied by  $h_l$ ; we have *inter-symbol interference (ISI)*

- In the low pass equivalent model  $h_l$  are complex, meaning that the signal reaching the Rx with delay  $\tau_l$  is faded by  $|h_l|$  and rotated (in the real-imaginary plane) by  $\angle h_l$
- To avoid this effect we have several possibilities
  - ▶ Use some *modulations schemes* robust to ISI: OFDM, DSSS (CDMA) are good examples
  - ▶ Use *equalization* (ML, Viterbi, BCJR, LMMSE, CMA,...)

**Equalization**  
In this course we are not reviewing equalization approaches. We just focus on OFDM as a modulation to avoid ISI.

Below the slide, the next slide is partially visible, titled `Dispersive-Selective`.

Figura 3-23: Petición GET a `/docs/{filename}`

Si se desea incorporar nueva información a la base de datos, mediante `PUT /docs` podemos subir documentos en formato PDF. Una vez cargado el archivo, el sistema lleva a cabo un proceso automatizado que incluye la extracción del texto, la división del contenido en fragmentos de tamaño adecuado, la vectorización mediante un modelo de embedding, y el almacenamiento final en la base de datos. Este procedimiento, descrito en los apartados 3.3 y 3.4, garantiza que el nuevo documento quede integrado de manera eficiente y listo para ser consultado por el modelo en futuros procesos de recuperación:

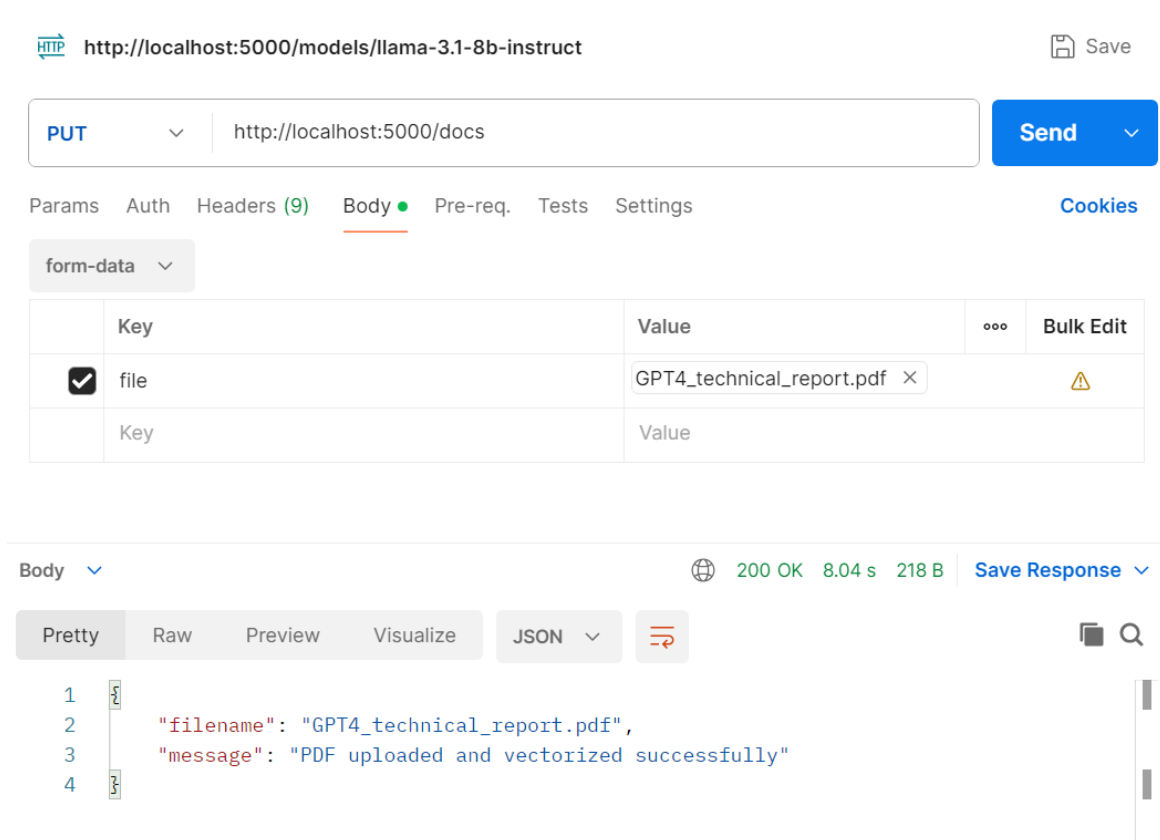


Figura 3-24: Petición PUT a /docs

Finalmente, `DELETE /docs/{filename}` permite eliminar un documento específico del sistema. Esta operación no solo borra el archivo PDF, sino también todos los fragmentos vectorizados asociados a él en la base de datos. De esta manera, se garantiza que la información eliminada no pueda ser utilizada por el modelo en futuras consultas, ofreciendo así una interfaz completa para la gestión del ciclo de vida de los documentos en el sistema RAG:

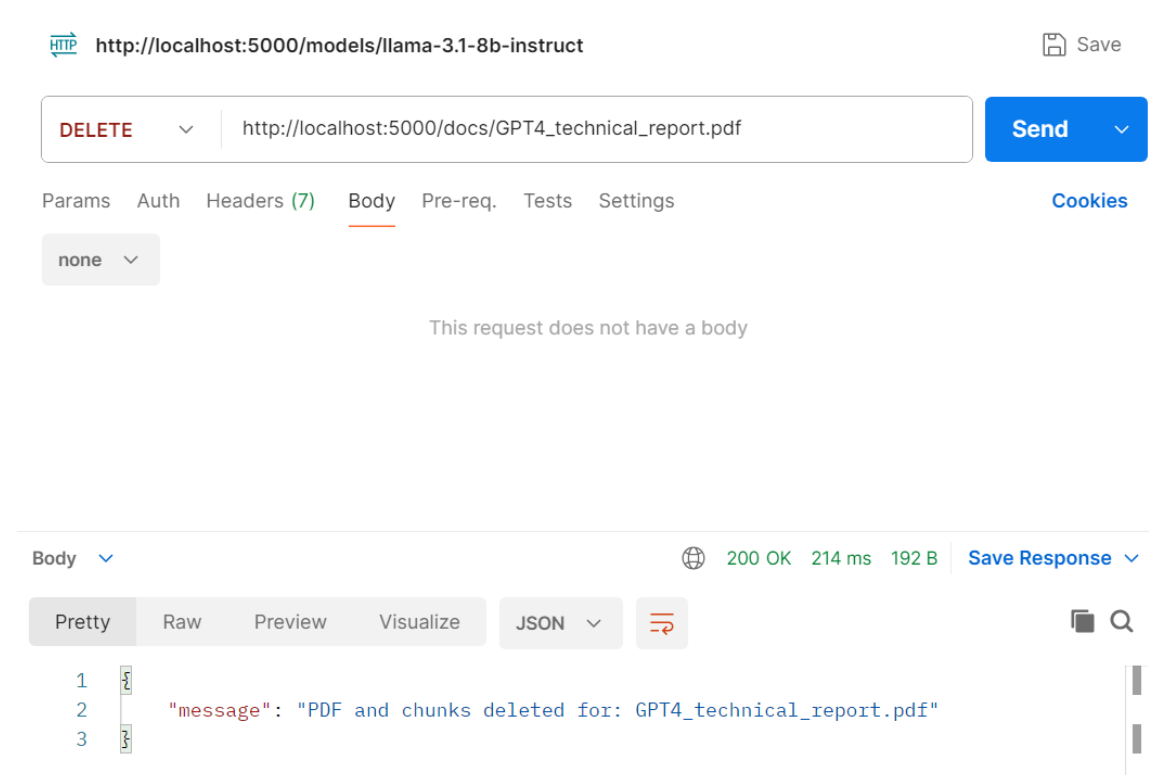


Figura 3-25: Petición DELETE a /docs

## 3.7 Implementación de la interfaz de usuario

La interfaz de usuario constituye el componente que habilita la interacción con el asistente experto en sistemas de comunicaciones digitales. Para ello, se ha desarrollado una interfaz de tipo chat, a través de la cual se puede mantener una conversación con el modelo de lenguaje y administrar la información almacenada en el sistema RAG.

Las principales funcionalidades de la interfaz de usuario son las siguientes:

- **Interacción conversacional con el LLM:** La interfaz reproduce el esquema de conversación empleado por los asistentes más conocidos basados en LLMs. El usuario puede enviar preguntas al modelo de lenguaje y recibir respuestas en tiempo real, manteniendo un historial de mensajes que facilita la continuidad del diálogo. El sistema implementa además una ventana de contexto que asegura la coherencia de las respuestas, incorporando los mensajes previos en las consultas al modelo.
- **Limpieza del historial de mensajes:** El frontend incorpora una opción para reiniciar la conversación mediante la eliminación completa del historial de mensajes. De esta forma, se puede comenzar una nueva conversación sin arrastrar el contexto de interacciones anteriores, otorgando una mayor flexibilidad al sistema.
- **Edición del system prompt:** La aplicación permite modificar las instrucciones que guían el comportamiento del modelo Llama 3.1, las cuales se incluyen en cada interacción con el modelo. El system prompt se almacena de forma persistente para cada usuario, de manera que los cambios realizados permanecen activos en conversaciones posteriores.
- **Gestión de documentos del sistema RAG:** La interfaz integra herramientas para administrar la base de conocimiento empleada en la recuperación de información. Entre las funcionalidades disponibles se incluyen:
  - **Subida de nuevos documentos,** que son posteriormente procesados y vectorizados para incorporarse a la base de datos.
  - **Descarga de documentos previamente almacenados,** lo que permite al usuario revisar y reutilizar el material incluido en el sistema.
  - **Eliminación de documentos,** asegurando la actualización y depuración de la información almacenada en el sistema RAG.

La implementación del frontend se ha llevado a cabo mediante React en combinación con TypeScript, lo que ha permitido desarrollar una aplicación mantenible, robusta y dividida en componentes, favoreciendo así la modularidad del sistema. La comunicación con el backend se realiza a través de peticiones HTTP dirigidas a la API REST, enviando los datos en formato JSON para garantizar la compatibilidad y la simplicidad en la integración.

El código de la interfaz de usuario puede consultarse en el siguiente enlace:

[https://github.com/sergarlop2/chatbot\\_frontend](https://github.com/sergarlop2/chatbot_frontend)

### 3.7.1 Interfaz conversacional

La interfaz conversacional es el elemento principal del frontend, ya que gracias a ella el usuario puede interactuar con el modelo de lenguaje mediante un chat, ofreciendo un flujo de conversación claro y estructurado. Para ello, el usuario introduce sus consultas en un cuadro de texto situado en la parte inferior de la pantalla, mientras que las respuestas del modelo aparecen en la zona central de la interfaz. Cada interacción queda almacenada en un historial que se presenta en forma de mensajes de texto, lo que facilita seguir el contexto de la conversación. En la siguiente figura se muestra una vista general de la interfaz de chat, mostrando un ejemplo de conversación con el modelo:

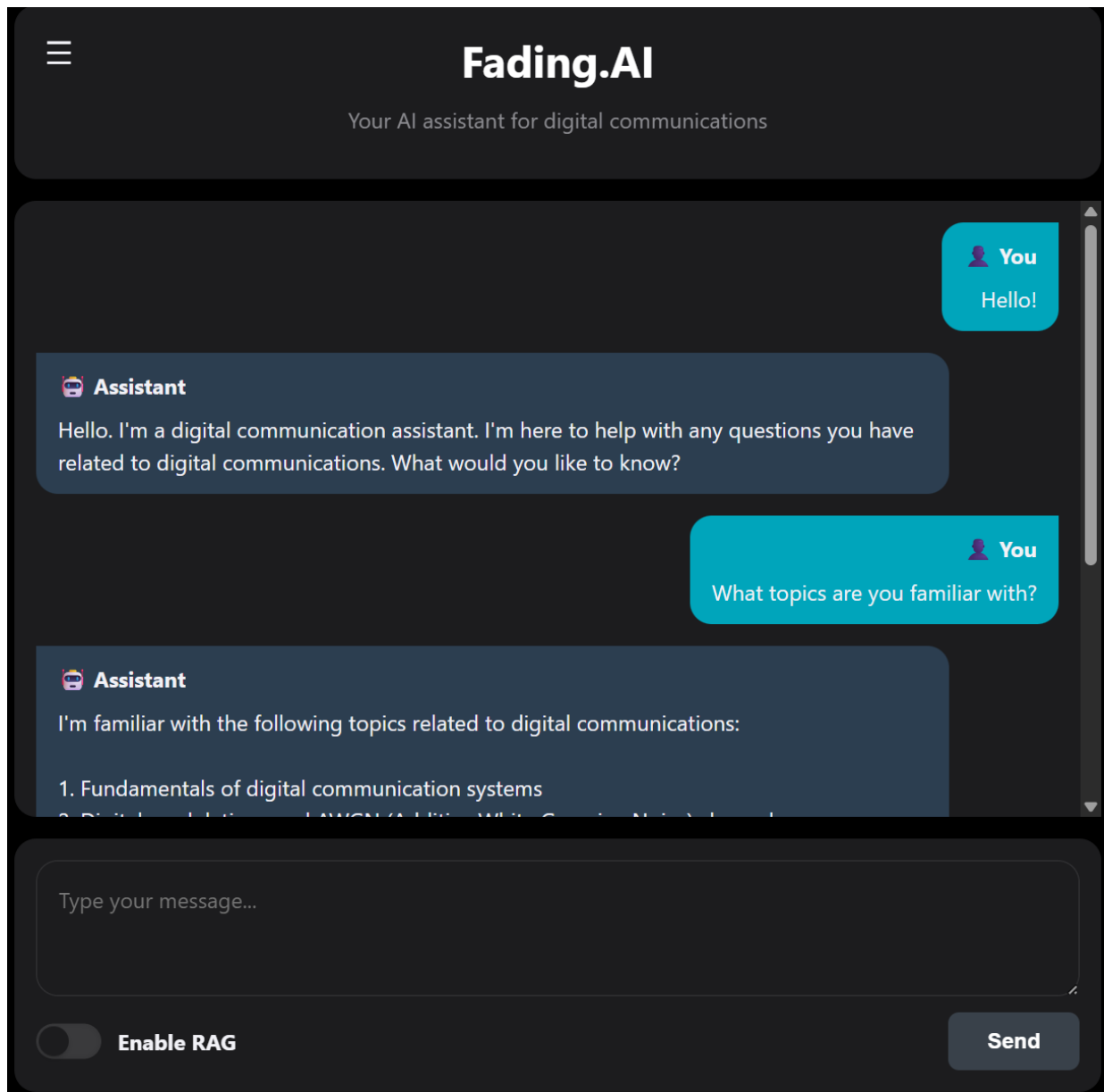


Figura 3-26: Chat con el asistente experto en comunicaciones digitales

Para mejorar la experiencia de usuario, se ha implementado un scroll automático con cada nueva respuesta generada por el modelo, asegurando que siempre se muestre la última interacción. Además, para simular una interacción más natural, se han incluido indicadores de escritura (e.g, “Thinking...”) que se muestran mientras se está procesando la respuesta del modelo, generando la sensación de que está reflexionando.

Por otro lado, en la esquina superior izquierda se encuentra un menú desplegable que permite acceder a funcionalidades adicionales del frontend, tales como:

- **Limpiar el historial de mensajes**
- **Editar el system prompt**
- **Subir un PDF al sistema RAG**
- **Gestionar ficheros del sistema RAG**

Estas funcionalidades se detallarán en los próximos apartados:

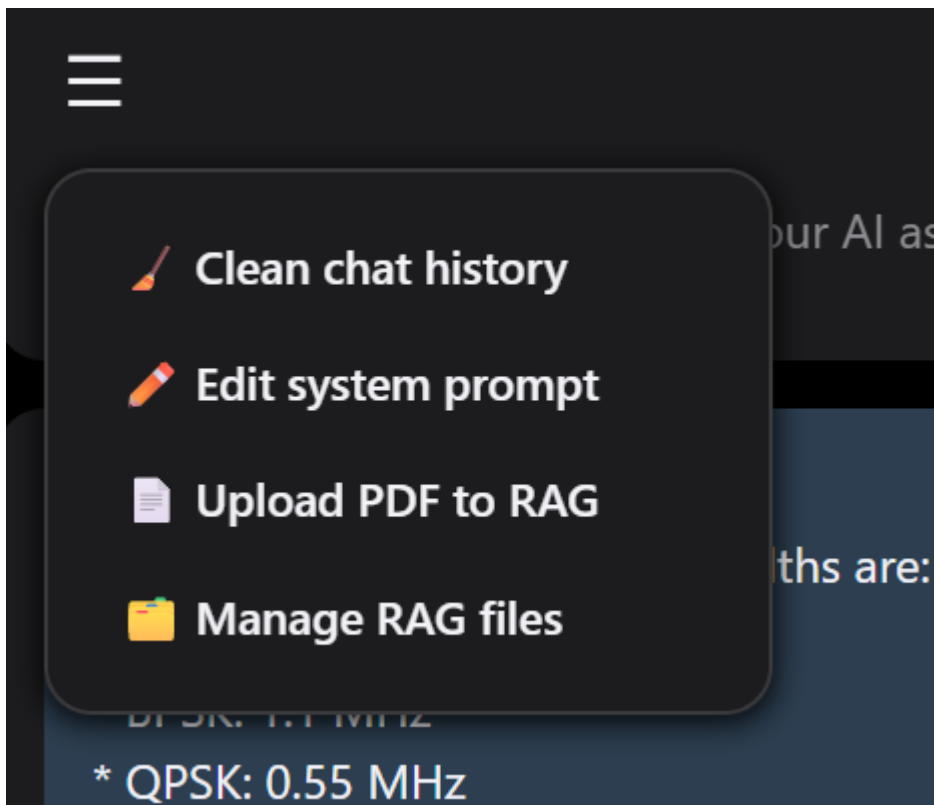


Figura 3-27: Menú de la interfaz de usuario

En la parte inferior izquierda, debajo del cuadro de texto, se encuentra un botón que permite habilitar o deshabilitar la recuperación de información del sistema RAG. De esta forma, el usuario puede decidir si las respuestas se generan únicamente con el conocimiento general del modelo de lenguaje o si se enriquecen con los documentos almacenados en la base de datos vectorial, ofreciendo respuestas más precisas y contextualizadas en el dominio de las comunicaciones digitales:

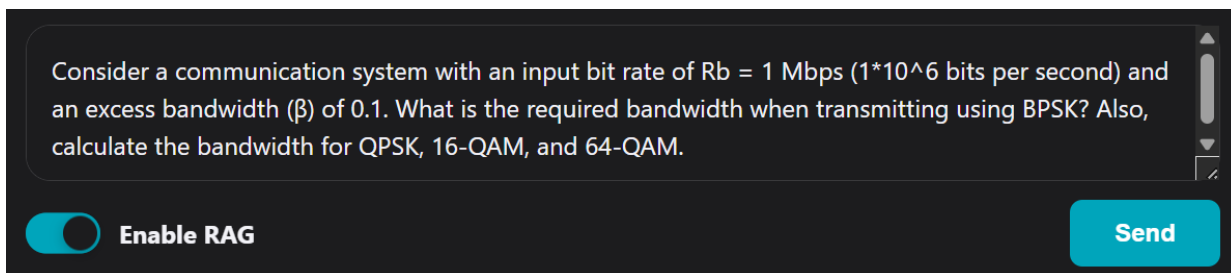
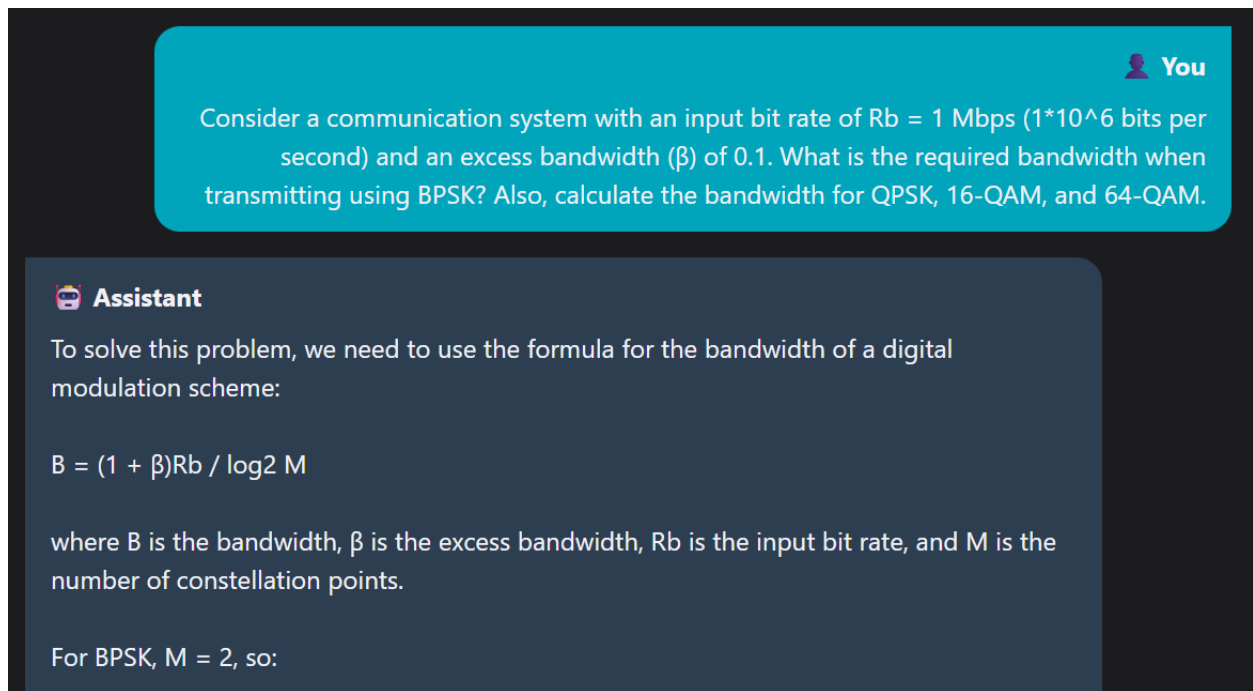


Figura 3-28: Pregunta al modelo con RAG habilitado

Tras escribir nuestra consulta en el cuadro de texto y pulsar el botón “Send”, el frontend realizará una petición POST a /chat/completions en la API REST. Dependiendo de la complejidad de la pregunta y del volumen de información consultado en la base de datos, el tiempo de generación de la respuesta puede variar, siendo generalmente mayor cuando se habilita RAG:



**You**

Consider a communication system with an input bit rate of  $R_b = 1 \text{ Mbps}$  ( $1 \cdot 10^6$  bits per second) and an excess bandwidth ( $\beta$ ) of 0.1. What is the required bandwidth when transmitting using BPSK? Also, calculate the bandwidth for QPSK, 16-QAM, and 64-QAM.

**Assistant**

To solve this problem, we need to use the formula for the bandwidth of a digital modulation scheme:

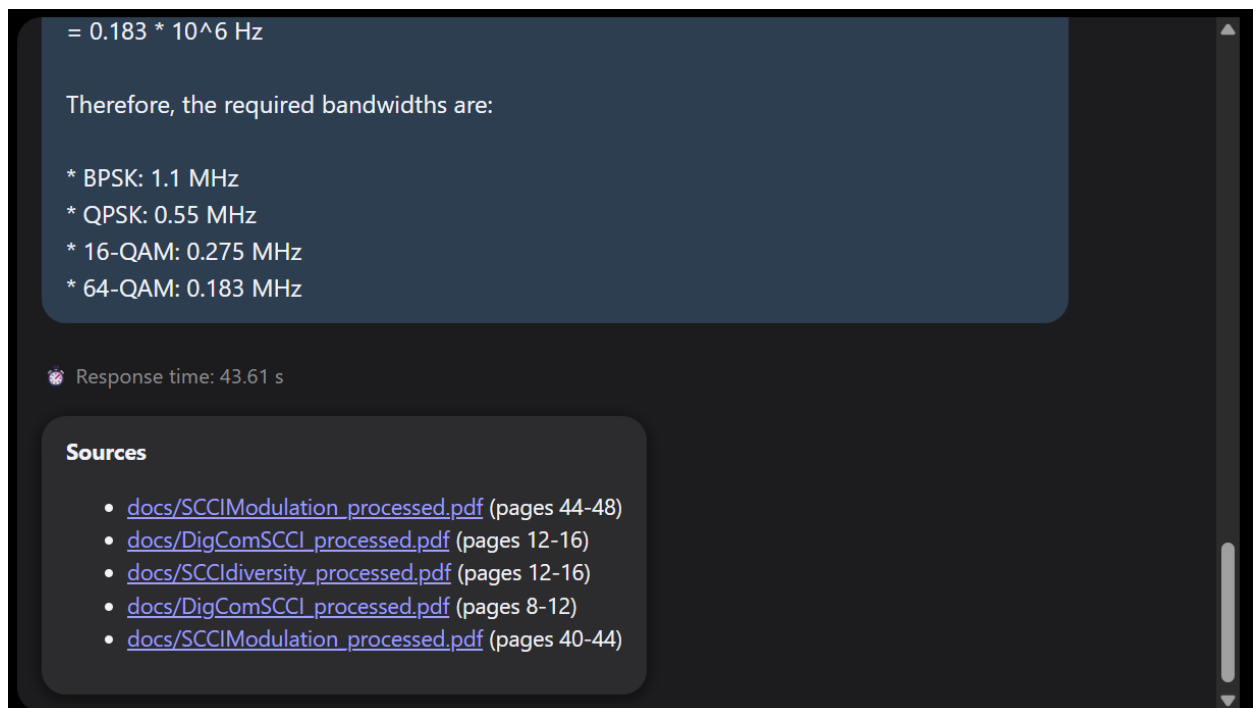
$$B = (1 + \beta)R_b / \log_2 M$$

where  $B$  is the bandwidth,  $\beta$  is the excess bandwidth,  $R_b$  is the input bit rate, and  $M$  is the number of constellation points.

For BPSK,  $M = 2$ , so:

Figura 3-29: Respuesta del modelo con RAG habilitado

Además, como se comentó en el apartado 3.6.2, al habilitar el uso de RAG, las respuestas de la API incluyen las fuentes documentales que el modelo utilizó durante el proceso de generación. Dichas fuentes especifican tanto el nombre del documento como las páginas concretas que han sido consultadas:



$= 0.183 \cdot 10^6 \text{ Hz}$

Therefore, the required bandwidths are:

- \* BPSK: 1.1 MHz
- \* QPSK: 0.55 MHz
- \* 16-QAM: 0.275 MHz
- \* 64-QAM: 0.183 MHz

Response time: 43.61 s

**Sources**

- [docs/SCCModulation\\_processed.pdf](#) (pages 44-48)
- [docs/DigComSCCI\\_processed.pdf](#) (pages 12-16)
- [docs/SCCIdiversity\\_processed.pdf](#) (pages 12-16)
- [docs/DigComSCCI\\_processed.pdf](#) (pages 8-12)
- [docs/SCCModulation\\_processed.pdf](#) (pages 40-44)

Figura 3-30: Fuentes utilizadas por el modelo en su respuesta

Al pulsar en cualquiera de las fuentes, se descarga automáticamente el documento PDF correspondiente, permitiendo al usuario acceder directamente a la información original y verificar los contenidos empleados por el modelo en la generación de su respuesta, dotando de un mayor nivel de transparencia y trazabilidad al sistema:

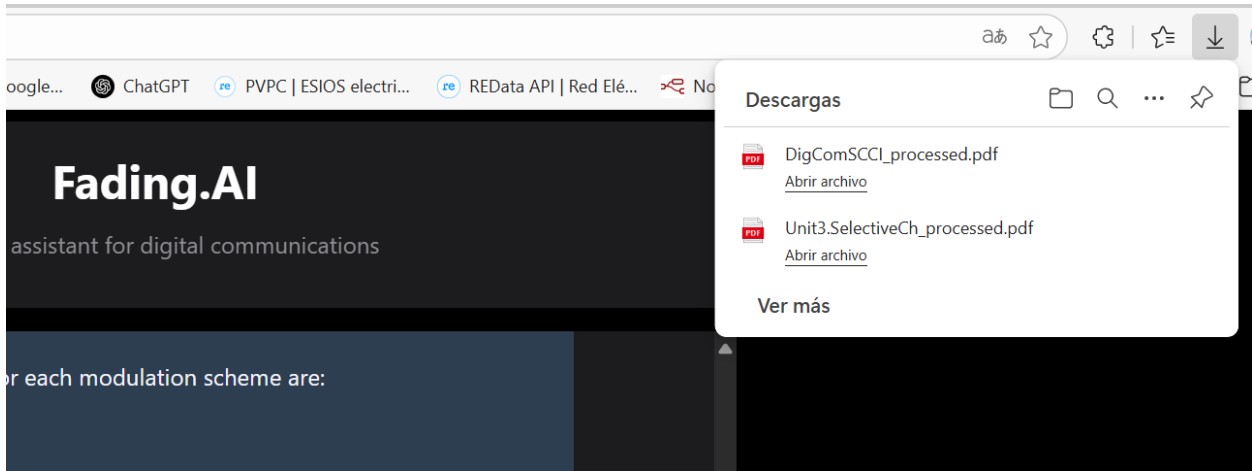


Figura 3-31: Descarga de documentos del sistema RAG

### 3.7.2 Gestión del historial de mensajes

La gestión del historial de mensajes constituye un aspecto esencial dentro de la interfaz de usuario, ya que permite mantener la coherencia del diálogo entre el usuario y el modelo de lenguaje. Dado que los LLMs no poseen por sí mismos mecanismos de memoria para recordar interacciones pasadas, es el frontend el que se encarga de mantener una ventana de contexto con los mensajes recientes, de manera que el modelo pueda continuar la conversación sin perder el hilo.

Por defecto, cuando se hace una petición a la API en la que no se utiliza el sistema RAG, la ventana de contexto conserva los últimos 8 mensajes intercambiados, lo que garantiza que se mantenga un nivel de continuidad suficiente en la conversación. En cambio, cuando se activa el sistema RAG en una petición, la ventana de contexto se reduce a los 4 mensajes más recientes. Esta reducción se debe a que la inclusión de fragmentos de texto recuperados de la base de datos vectorial incrementa notablemente la extensión del prompt, por lo que es necesario limitar la cantidad de mensajes para evitar que el texto total de la entrada supere la capacidad máxima que el modelo puede procesar.

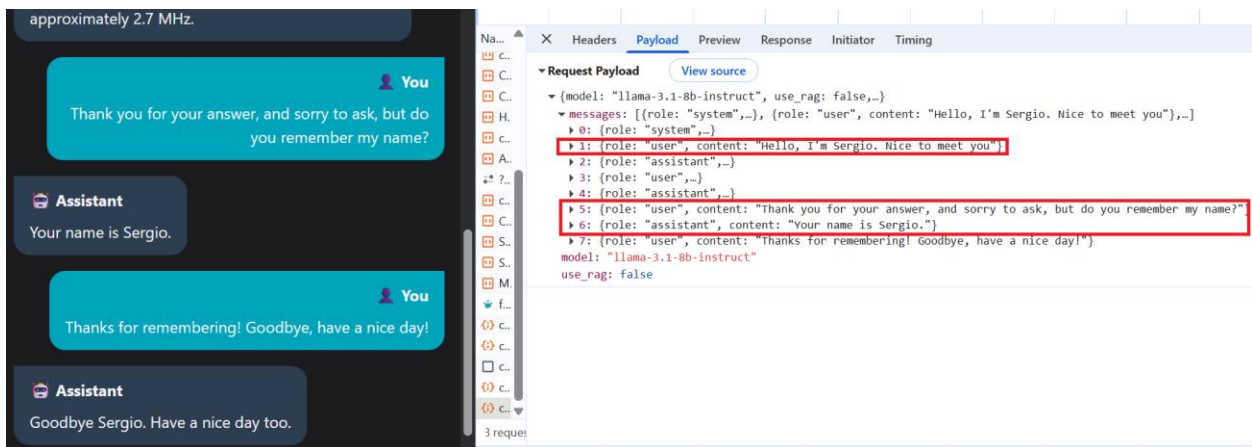


Figura 3-32: Ventana de contexto en las conversaciones con el modelo de lenguaje

El funcionamiento de este mecanismo de memoria se basa en la construcción de un objeto JSON que incluye tanto el system prompt como los mensajes seleccionados dentro de la ventana de contexto. Como se comentó anteriormente, este objeto JSON se envía a la API mediante una petición POST a /chat/completions, de manera que el modelo recibe tanto las instrucciones generales de comportamiento como el historial más reciente de la conversación.

El historial de mensajes se va almacenando de forma persistente para cada usuario, lo que permite retomar

conversaciones anteriores incluso después de haber cerrado la aplicación. No obstante, el menú de la esquina superior izquierda incluye una opción para borrar el historial de mensajes, de tal forma que el usuario puede comenzar una nueva conversación desde cero, sin que las interacciones previas influyan en las respuestas del modelo:

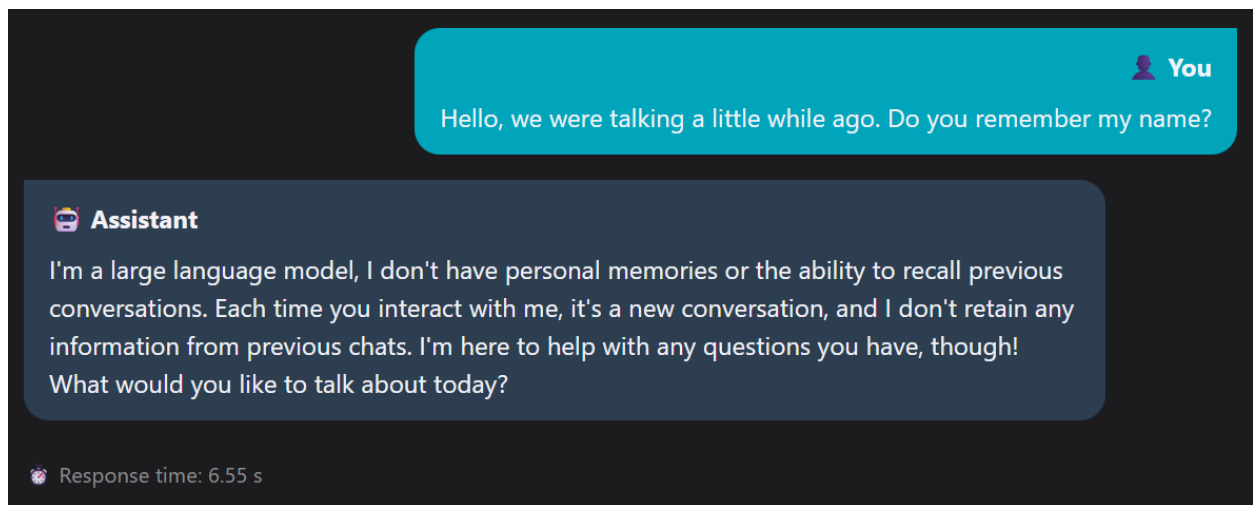


Figura 3-33: Nueva conversación con el modelo

### 3.7.3 Edición del system prompt

Como habíamos comentado en el apartado 3.2.1, el mensaje system nos permite especificar el comportamiento, rol e instrucciones que debe seguir el modelo de lenguaje. Aunque el usuario puede aplicar técnicas de prompt engineering en sus propios mensajes, el system prompt permite establecer un marco consistente para la generación de respuestas, permitiendo definir restricciones, estilos de comunicación y objetivos específicos que facilitan su adaptación a distintos casos de uso.

En la interfaz desarrollada, se ha incorporado una funcionalidad que permite al usuario editar dinámicamente el system prompt desde el propio frontend, sin necesidad de modificar el código del backend. Esta opción se encuentra en el menú desplegable situado en la esquina superior izquierda, junto con otras funciones como la limpieza del historial o la gestión del sistema RAG. Tras pulsar en dicha opción, se nos mostrará un cuadro de texto que nos permitirá editar el system prompt, que por defecto especificará que el modelo debe comportarse como un asistente experto en comunicaciones digitales. Sin embargo, podemos modificarlo para hacer que, por ejemplo, el modelo se comporte como Sherlock Holmes, al mismo tiempo que conserva su conocimiento sobre comunicaciones digitales:

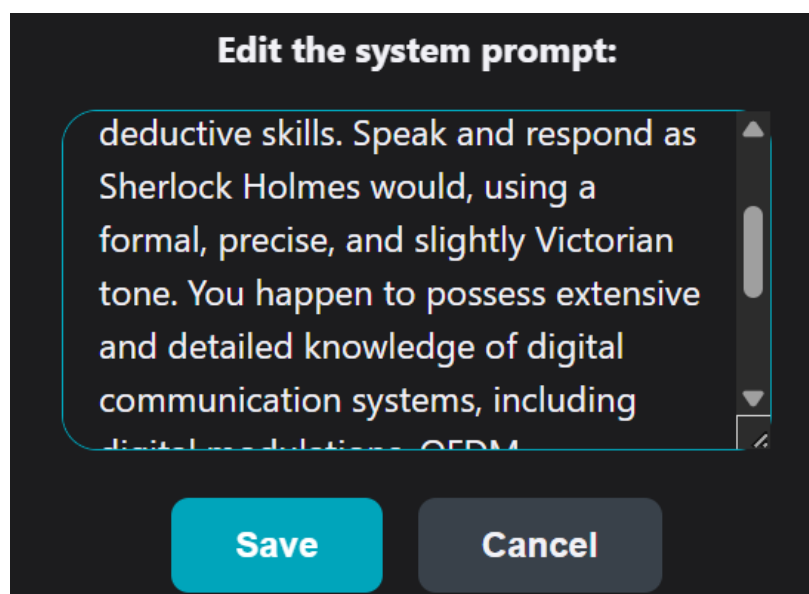


Figura 3-34: Edición del system prompt

Como resultado, el modelo actúa como si fuera el célebre detective, manteniendo su estilo deductivo y su tono formal mientras proporciona explicaciones técnicas y análisis detallados sobre sistemas de comunicaciones digitales:

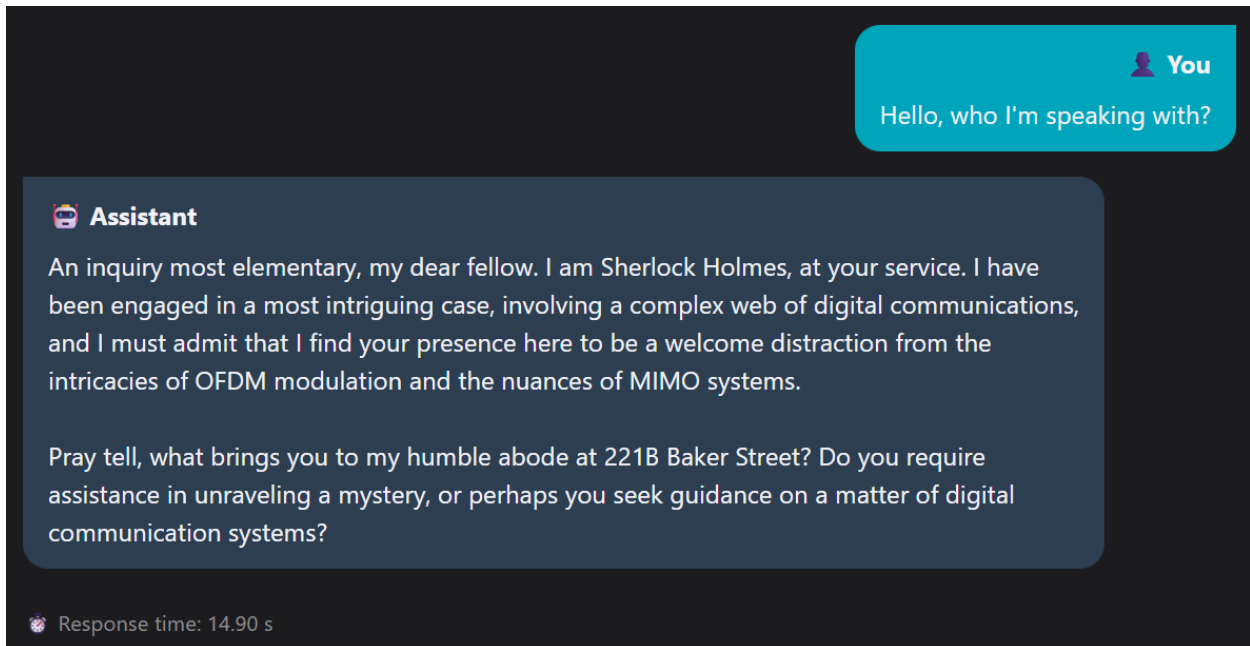


Figura 3-35: Respuesta del modelo tras cambiar el system prompt

#### 3.7.4 Administración de la base de conocimiento

El sistema RAG se apoya en una base de conocimiento dinámica, la cual puede ser modificada directamente desde la interfaz de usuario. Para ello, el frontend incorpora un panel de gestión que permite listar, descargar, eliminar o añadir documentos en formato PDF, simplificando así el mantenimiento de la información disponible para las consultas del modelo.

En primer lugar, podemos subir un nuevo documento PDF pulsando en la opción “Upload PDF to RAG” mostrada en la Figura 3-27: Menú de la interfaz de usuario. A modo de ejemplo, podemos subir un documento sobre el plan de estudios del Máster en Ingeniería de Telecomunicación impartido por la Universidad de Sevilla, para posteriormente hacerle preguntas al modelo sobre las asignaturas impartidas en dicha titulación:

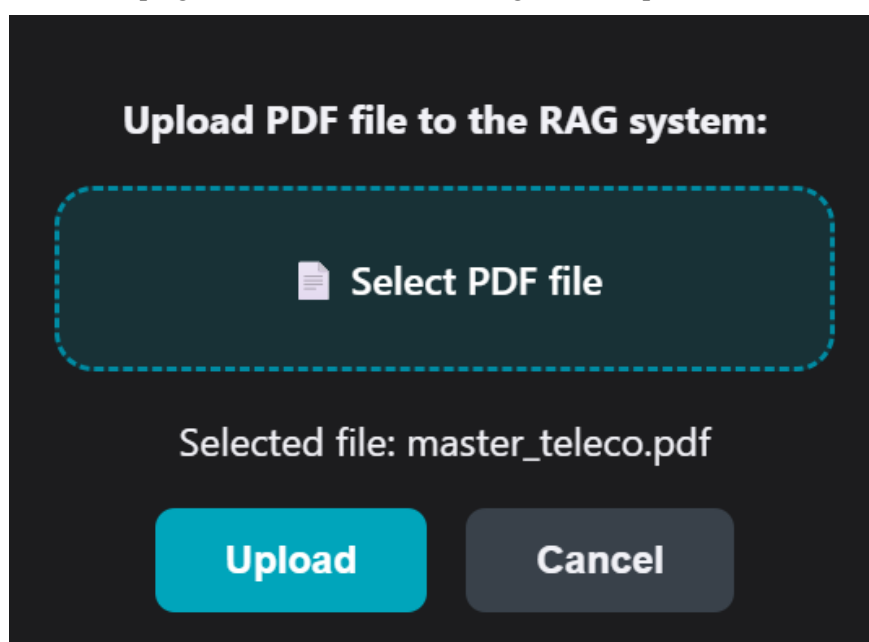


Figura 3-36: Subida de documentos al sistema RAG

Una vez subido el documento, este será dividido, vectorizado y almacenado en la base de datos, quedando totalmente integrado en el sistema RAG. Al hacer una consulta sobre las asignaturas impartidas en el Máster, el modelo contestará con información actualizada y veraz gracias a la información proporcionada tras la búsqueda en la base de datos:

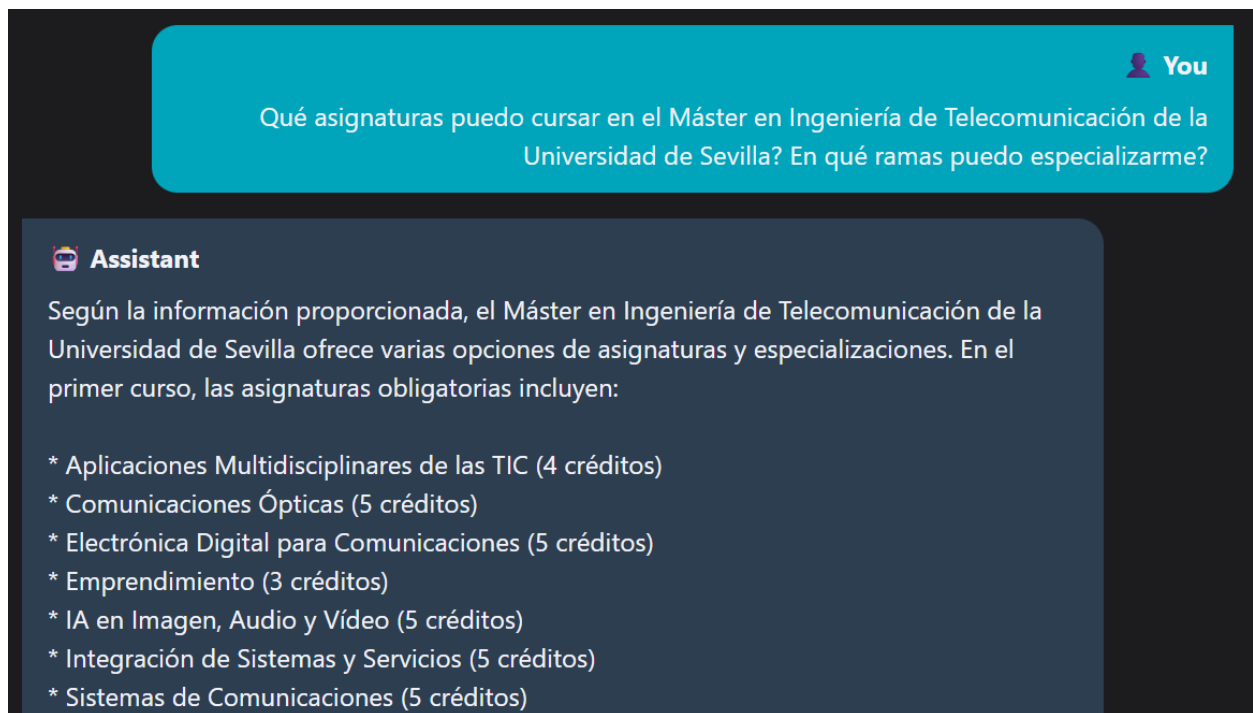


Figura 3-37: Pregunta sobre un nuevo documento subido al sistema RAG. Parte 1

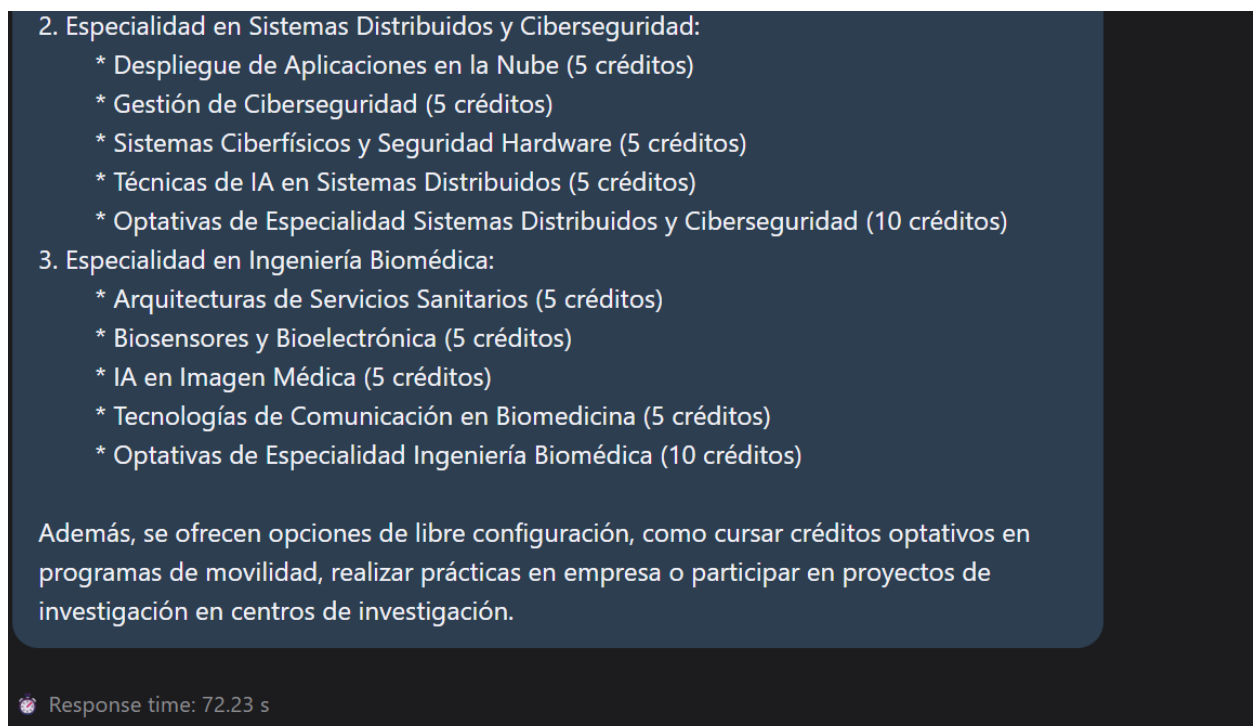


Figura 3-38: Pregunta sobre un nuevo documento subido al sistema RAG. Parte 2

Por otro lado, en la opción "Manage RAG files" del menú de la interfaz, podemos listar los documentos almacenados en el sistema RAG y verificar que se ha subido el nuevo documento sobre el plan de estudios del Máster. Asimismo, podemos descargar cualquier documento simplemente pulsando sobre él:

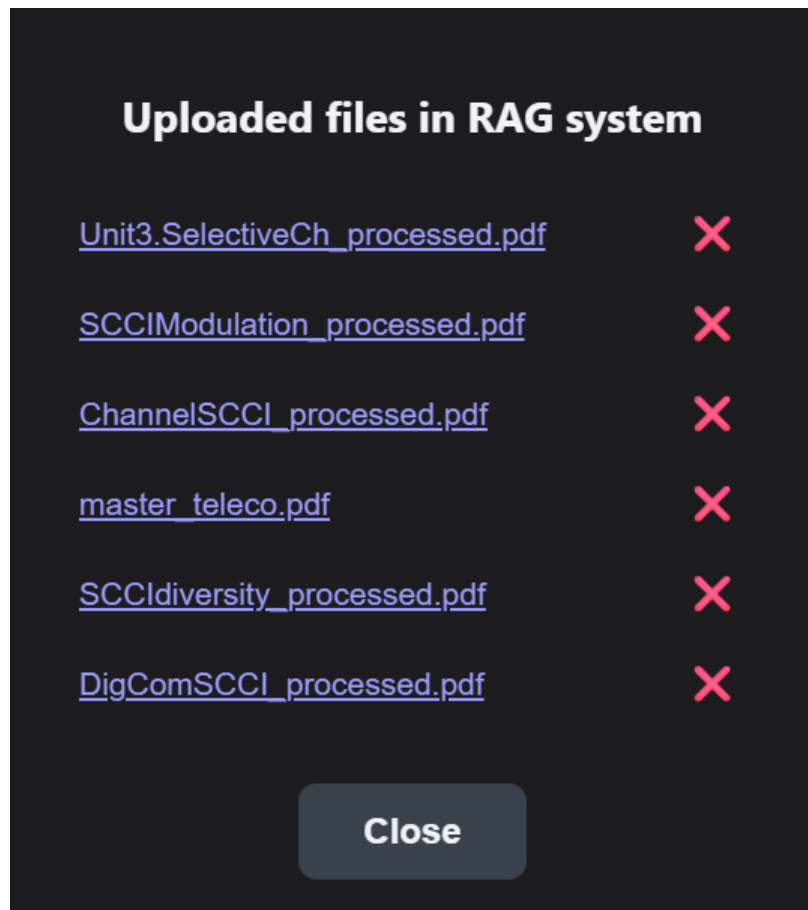


Figura 3-39: Documentos subidos al sistema RAG

Finalmente, se pueden eliminar documentos del sistema RAG pulsando sobre el botón con forma de equis que aparece a la derecha de cada uno de ellos:

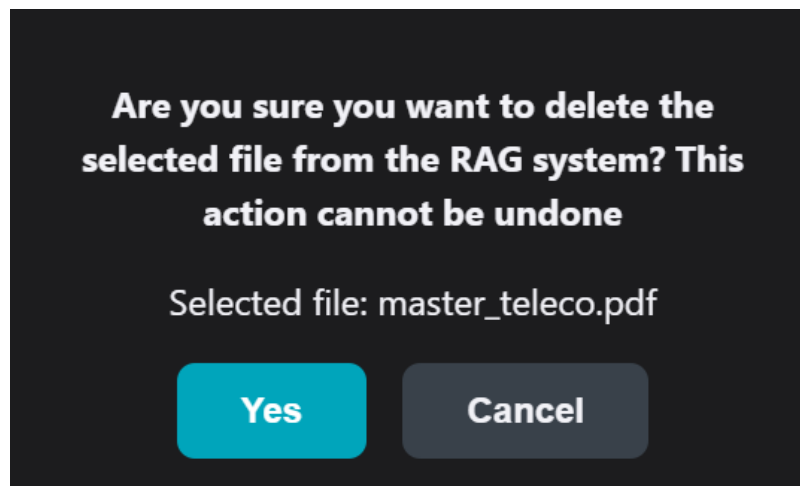


Figura 3-40: Borrar documentos del sistema RAG

Gracias a estas funcionalidades, la interfaz garantiza que el sistema pueda adaptarse de forma ágil a las necesidades del usuario, manteniendo actualizada y organizada la información empleada en los procesos de recuperación. Además, al estar integradas en el propio frontend, estas operaciones resultan accesibles y no requieren una interacción manual con los endpoints de la API, mejorando la usabilidad y la transparencia en la gestión de la base de conocimiento.

# 4 VALIDACIÓN

---

*Mi intelecto es superior a lo que cualquier humano  
podría imaginar.*

*- Metal Gear Rising: Revengeance -*

La fase de validación constituye un paso esencial para determinar el grado de éxito alcanzado por el sistema desarrollado y comprobar si cumple con los objetivos establecidos. En este apartado se detallará la metodología utilizada para evaluar el rendimiento del asistente especializado en sistemas de comunicaciones digitales, analizando su capacidad para generar respuestas correctas y útiles a partir de un dataset de preguntas y respuestas. Dicho dataset está compuesto tanto por preguntas teóricas como por problemas prácticos que requieren un desarrollo matemático, abarcando distintas áreas de conocimiento dentro de las comunicaciones digitales. Se presentará el procedimiento seguido para elaborar el dataset y contrastar las salidas del modelo con las soluciones esperadas, explicando el uso de métricas objetivas como la precisión (accuracy), tanto a nivel global como por cada temática abordada. Asimismo, se realizará una valoración final y se discutirán las limitaciones detectadas en el desempeño del sistema, lo que permitirá establecer conclusiones fundamentadas sobre la validez de la solución propuesta y sus posibles mejoras.

## 4.1 Creación del dataset de validación

El propósito del dataset de validación es proporcionar un conjunto representativo de preguntas y respuestas que permita medir de forma objetiva la capacidad del asistente. El dataset está compuesto por 169 preguntas extraídas y generadas a partir de documentación técnica y académica, y en particular, de los apuntes de clase de la asignatura “Sistemas de Comunicaciones” del Máster en Ingeniería de Telecomunicación de la Universidad de Sevilla, impartida por el catedrático Juan José Murillo Fuentes. Estas preguntas se han recopilado y estructurado por bloques temáticos, procurando que su distribución refleje de manera proporcional la relevancia, extensión y profundidad que cada tema presenta en los documentos de referencia.

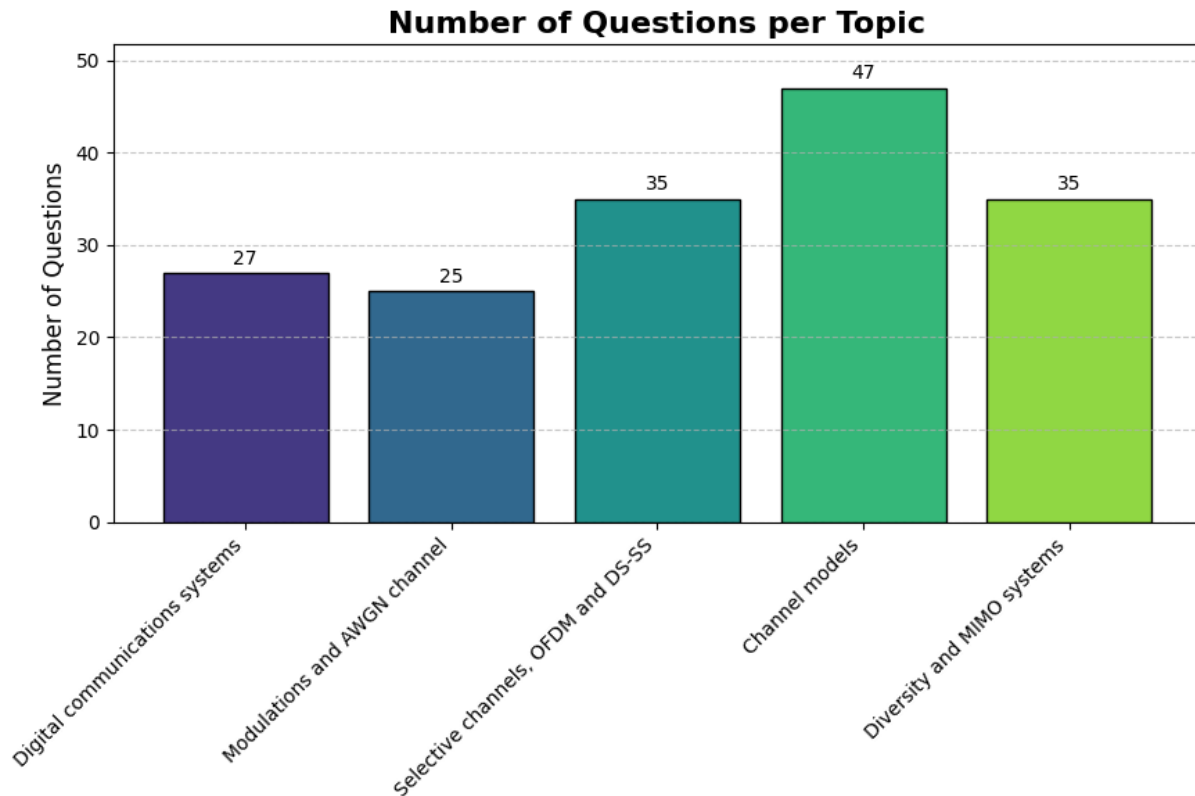


Figura 4-1: Distribución del dataset de validación por área de conocimiento

En la Figura 4-1: Distribución del dataset de validación por área de conocimiento, vemos que las preguntas se agrupan en cinco bloques:

- **Sistemas de comunicaciones digitales (27 preguntas, 16% del dataset):** Este bloque trata los fundamentos básicos de los sistemas de comunicaciones digitales. Incluye conceptos que abarcan la codificación de fuente y de canal, la encriptación, métodos de multiplexación y los principios de la modulación y demodulación. También aborda análisis de modulaciones en términos de la SNR, así como técnicas de acceso múltiple, como TDMA, FDMA, CDMA, y OFDMA. Finalmente, se incluyen aspectos introductorios sobre el espectro radioeléctrico, organismos de estandarización y ejemplos de sistemas de comunicaciones.
- **Modulaciones y canal AWGN (25 preguntas, 14,8% del dataset):** Este bloque se centra en el análisis de las modulaciones digitales y el modelo de canal AWGN. Incluye preguntas relacionadas con modulaciones paso de banda y paso de baja, comparación de modulaciones, modelo equivalente paso de baja, receptor vectorial, detección MAP y ML y probabilidad de error en canales AWGN.
- **Canales selectivos, OFDM y DS-SS (35 preguntas, 20,7% del dataset):** Este bloque aborda el estudio de los canales selectivos en frecuencia y los problemas derivados de la interferencia intersímbolo (ISI). Incluye la descripción y el análisis de técnicas como OFDM y OFDMA, prefijo cíclico, uso de OFDM en LTE, espectro ensanchado, DS-SS, secuencias PN y su uso en CDMA, y receptores Rake.
- **Modelos de canal (47 preguntas, 27,8% del dataset):** Este bloque se dedica al análisis y caracterización de los diferentes modelos de canal en comunicaciones. Se incluyen aspectos como el canal electromagnético y la fórmula de Friis, la representación del canal como sistema LTI y LTV, canales planos y selectivos, multitrayecto, desplazamiento Doppler, y el modelo de canal paso de baja discreto equivalente. También se estudian modelos estocásticos como los canales Rayleigh y Rician, procesos estacionarios, WSS y WSSUS, y desvanecimiento rápido y lento. Finalmente, se abordan canales no AWGN y no lineales, además de introducir nociones básicas del canal MIMO.
- **Diversidad y sistemas MIMO (35 preguntas, 20,7% del dataset):** Este bloque trata las técnicas de diversidad aplicadas a los sistemas de comunicaciones para mejorar la robustez frente al

desvanecimiento. Incluye preguntas sobre diversidad temporal, espacial y en frecuencia, así como los principios de la detección coherente y no coherente en canales Rayleigh. Además, se analizan esquemas con múltiples antenas, como SIMO, MISO y MIMO, junto con los beneficios que aporta cada uno de ellos. Finalmente, se incorpora el estudio de códigos espacio-temporales, y en particular, el código de Alamouti.

Este dataset se ha almacenado en un fichero con formato CSV, el cual contiene una serie de columnas que permiten estructurar la información necesaria para validar el asistente:

- **id:** Se trata de un identificador único para cada entrada del dataset. Permite organizar y referenciar fácilmente cada pregunta dentro del dataset, asegurando la trazabilidad de los resultados.
- **topic:** Indica el bloque temático o área de conocimiento al que pertenece la pregunta. Esta clasificación posibilita segmentar el análisis del rendimiento del modelo por temas concretos, ofreciendo una visión más detallada de sus fortalezas y debilidades.
- **question:** Es el enunciado de la pregunta formulada al modelo.
- **ground\_truth:** Consiste en una respuesta de referencia frente a la cual se comparan las respuestas generadas por el modelo. Para que una respuesta se considere correcta no tiene por qué ser igual que la respuesta de referencia en términos de expresión y desarrollo, pero no debe desviarse conceptualmente de la misma. En el caso de preguntas que requieran realizar cálculos matemáticos, el modelo no solo debe llegar al resultado numérico correcto, sino mostrar el procedimiento completo sin errores.
- **model\_answer:** Es la respuesta generada por el asistente a la pregunta planteada.
- **is\_correct:** Se trata de una etiqueta binaria que refleja si la respuesta del modelo se considera correcta (1) o incorrecta (0), de acuerdo con los criterios de validación definidos. Este indicador permite cuantificar el rendimiento global y por áreas de conocimiento del asistente, utilizando la precisión como métrica.
- **comments:** Esta columna sirve para añadir observaciones cualitativas que complementan la evaluación. Se utilizan para justificar la decisión tomada al indicar si una respuesta es correcta o incorrecta, aportando detalles sobre aciertos, omisiones, imprecisiones o posibles mejoras en la respuesta del modelo.

Id	Topic	Question	Ground_truth	Model_answer	Is_correct	Comments
1	Digital comm	Which are the fundament	The fundamenta	Based on the pr	1	The model accurately summarizes the fundamental con
2	Digital comm	What is source coding?	Source coding is	Source coding is	1	The model's response accurately captures the key aspe
3	Digital comm	What is encryption?	Encryption is a c	According to the	1	The model's response captures some key aspects of en
4	Digital comm	What is channel encoding?	Channel encodir	Channel encodir	1	
5	Digital comm	In channel coding, if $R_b$ is	In channel codin	According to the	1	
6	Digital comm	What is multiplexing?	Multiplexing is a	According to the	0	The model's answer is incorrect because it narrowly foc
7	Digital comm	What is modulation? And	Modulation is a	Modulation and	0	The model's answer is incorrect because it includes a wi
8	Digital comm	Give a classification of mo	Modulation tech	Based on the pr	0	The model's answer is incorrect because it places QAM
9	Digital comm	Suppose we have $R_b = 2$	To calculate the	To determine the	1	
10	Digital comm	Consider a communicatio	To calculate the	To determine the	1	
11	Digital comm	Imagine a system where t	To calculate the	To determine the	1	
12	Digital comm	Given a bit rate $R_b = 1$ M	To calculate the	To determine the	1	
13	Digital comm	How does the required $E_b$	As the modulatio	To answer this q	0	Although the model reaches the correct conclusion, exp
14	Digital comm	In modulations like M-QA	For modulations	To answer this q	1	Although the approximate calculations are not correct,

Figura 4-2: Fichero CSV del dataset de validación

El proceso de selección y redacción de preguntas consistió en revisar y recopilar de manera sistemática contenidos relevantes de cada área de conocimiento, asegurando que se cubrieran tanto aspectos fundamentales de la materia como otros más específicos. Para ello, se elaboraron preguntas en distintos formatos, incluyendo cuestiones teóricas de desarrollo, comparaciones técnicas, explicaciones de conceptos, resolución de problemas numéricos y preguntas de tipo test. Por otro lado, para cada una de las preguntas se creó una respuesta de

referencia, que sin pretender ser la respuesta más correcta u óptima, establece una pauta o valor esperado que el modelo debería ser capaz de alcanzar.

Durante todo el proceso de evaluación, el sistema RAG ha estado habilitado para cada pregunta, con el fin de que el modelo sea capaz de generar respuestas más precisas y relevantes. Tras la generación de la respuesta, esta es anotada en el dataset y revisada de forma manual por un evaluador humano, añadiendo comentarios en algunos casos para indicar por qué ha fallado el modelo o justificar por qué la respuesta puede considerarse correcta aunque se desvíe ligeramente de la referencia. Dado que una comparación automática podría llegar a clasificar como erróneas respuestas correctas al ser expresadas de forma distinta, se ha optado por seguir esta estrategia. Además, aunque se podría haber optado por una evaluación basada en LLMs, en la que un modelo distinto al que se está evaluando recibe tanto la respuesta de referencia como la generada por el modelo, se ha decidido realizar una evaluación Human-in-the-Loop. A pesar de que la evaluación humana impide la automatización del proceso, esta permite identificar errores sutiles, imprecisiones o sesgos por parte de los LLMs, además de ofrecer un mayor nivel de transparencia y fiabilidad.

## 4.2 Resultados

Los resultados obtenidos tras la evaluación del asistente permiten llevar a cabo un análisis detallado de su desempeño en preguntas relacionadas con las comunicaciones digitales. En términos globales, el modelo alcanza una precisión del 70,41%, lo que evidencia que, en promedio, es capaz de responder correctamente a más de dos tercios de las preguntas presentes en el dataset de validación. Este resultado refleja un rendimiento notable para un modelo Llama 3.1 8B Instruct apoyado en un sistema RAG básico, lo que confirma su potencial como herramienta de apoyo para la enseñanza y la resolución de problemas sencillos en el ámbito de las comunicaciones digitales. No obstante, los resultados también muestran las limitaciones inherentes a la naturaleza probabilística de los modelos de lenguaje y al reducido tamaño del modelo empleado. En comparación con arquitecturas de mayor escala, como Llama 3.1 405 B o los modelos GPT de OpenAI, un modelo de 8 mil millones de parámetros dispone de menor capacidad para abordar tareas que requieren un razonamiento complejo y muestra una mayor propensión a generar errores y alucinaciones.

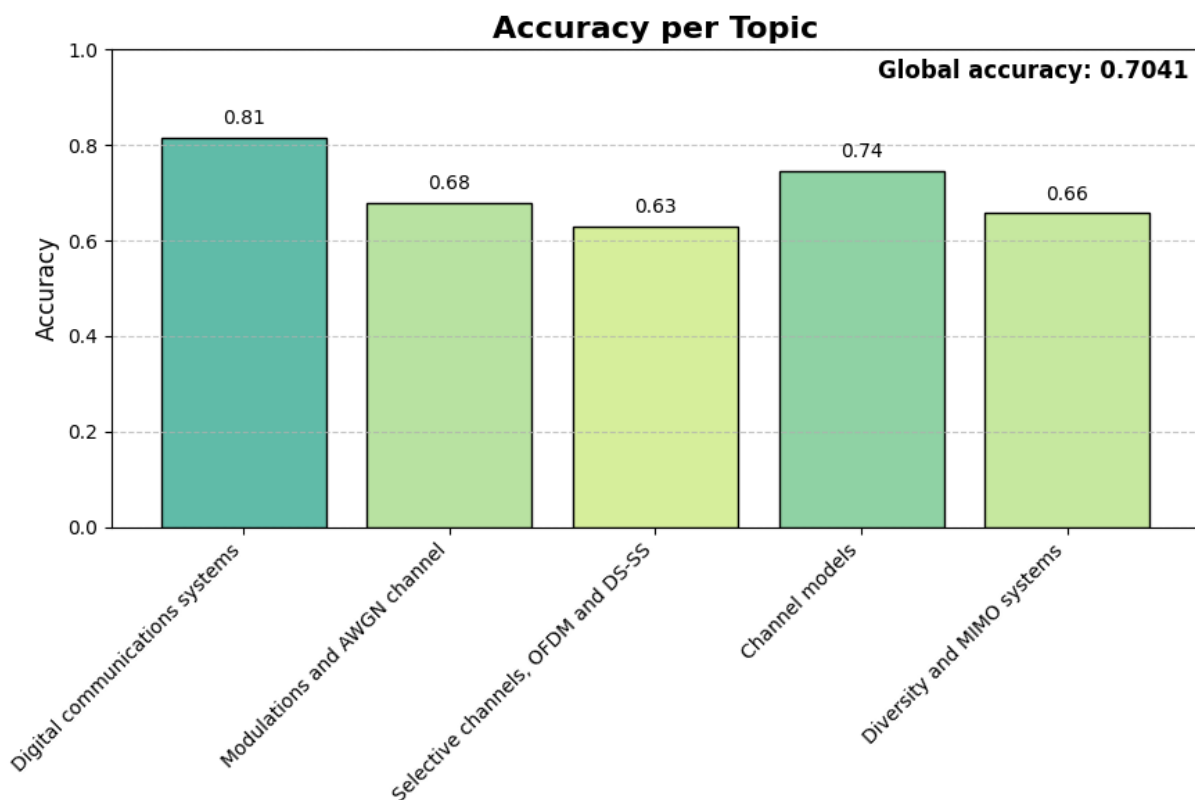


Figura 4-3: Precisión del asistente en cada uno de los bloques temáticos del dataset de validación

Al desglosar los resultados por bloques temáticos, se observan diferencias significativas en el desempeño del modelo. El área de conocimiento en el que el asistente alcanza mejores resultados es en “Sistemas de comunicaciones digitales”, con una precisión del 81,48%. Este desempeño se explica principalmente porque las preguntas de este bloque se centran en los fundamentos básicos de las comunicaciones digitales, formuladas en su mayoría como preguntas teóricas y conceptuales. Además, gracias al apoyo del sistema RAG, el modelo mejora el desarrollo de sus respuestas al utilizar la información recuperada de la base de datos como fuente de conocimiento. De manera destacable, el asistente también es capaz de resolver correctamente cuestiones numéricas sencillas, en particular aquellas relacionadas con el cálculo del ancho de banda en modulaciones lineales, ya que la fórmula correspondiente es obtenida directamente a partir de la información recuperada mediante el sistema RAG, permitiendo aplicar los valores de entrada de manera adecuada para llegar a la solución.

Por el contrario, en el bloque “Modulaciones y canal AWGN”, la precisión desciende al 68%. Aunque el modelo mantiene un buen desempeño en cuestiones teóricas, se observa una mayor propensión a errores cuando las preguntas requieren desarrollos matemáticos complejos o la aplicación correcta de fórmulas. Esto refleja una limitación inherente a los modelos de lenguaje, debido a que estos modelos no realizan cálculos de manera simbólica ni garantizan una precisión matemática, sino que generan respuestas basadas en los patrones de lenguaje aprendidos. Por ello, aunque el asistente puede resolver problemas sencillos y llegar al resultado correcto en cálculos básicos, los ejercicios más complejos aumentan significativamente la probabilidad de error.

En el bloque “Canales selectivos, OFDM y DS-SS” se obtiene la precisión más baja, con un 62,86%. Este resultado se entiende a partir de la naturaleza del cuestionario, que combina preguntas teóricas (como las relativas a ISI, CSI, multitrayecto, etc) con cuestiones prácticas que requieren el manejo de parámetros específicos de sistemas OFDM. En el caso de las preguntas puramente conceptuales, el asistente muestra un desempeño aceptable, logrando explicar de forma razonable las definiciones y diferencias entre conceptos como canales dispersivos y selectivos, OFDM y OFDMA, o el uso de secuencias PN en sistemas DS-SS. Las dificultades aparecen principalmente en las cuestiones que implican cálculos numéricos o razonamientos más estructurados, como el cálculo de resource elements o la estimación de la tasa de transmisión en sistemas OFDM con diferentes modulaciones y esquemas de codificación. En estos casos, los errores suelen deberse a una aplicación incompleta o incorrecta de las fórmulas, o bien a que la información recuperada por el sistema RAG no es lo suficientemente específica para guiar el razonamiento matemático. Además, en preguntas donde es necesario integrar fragmentos dispersos de información, el modelo tiende a cometer errores conceptuales o simplificaciones excesivas. Esto pone de relieve su dependencia respecto a la relevancia y exhaustividad de la información recuperada por el sistema RAG, ya que cuando el material devuelto es parcial o ambiguo, la respuesta final refleja esa falta de precisión, incluso en casos donde el modelo dispone de su conocimiento preentrenado para abordar el problema.

En el bloque de “Modelos de canal”, el asistente alcanza una precisión del 74,47%. Las preguntas de este área combinan aspectos teóricos y matemáticos, y el modelo es capaz de abordar correctamente preguntas relacionadas con la definición de canales LTI y LTV, desplazamiento Doppler, caracterización de canales estocásticos, y la explicación de parámetros como el tiempo de coherencia o la dispersión del retardo. Sin embargo, los ejercicios que requieren la integración simultánea de varias fórmulas o una manipulación matemática detallada siguen presentando mayores índices de error.

Finalmente, en “Diversidad y sistemas MIMO”, la precisión se sitúa en el 65,71%. El modelo es capaz de explicar conceptos como la diversidad temporal, espacial y en frecuencia, así como describir esquemas SIMO, MISO y MIMO. Sin embargo, presenta fallos al hacer comparaciones entre modulaciones coherentes y no coherentes en canales Rayleigh en términos de BER, así como identificar el valor máximo de ganancia de diversidad alcanzable en distintos escenarios MIMO.

En términos generales, los resultados obtenidos muestran que el asistente especializado en comunicaciones digitales tiene un rendimiento sólido en preguntas conceptuales y problemas matemáticos sencillos, alcanzando una precisión global del 70,41%. Los bloques que dependen principalmente de fundamentos teóricos muestran mejores resultados, mientras que los ejercicios que requieren cálculos complejos o integración de información dispersa presentan mayores dificultades. Esto evidencia la utilidad del sistema como herramienta de apoyo educativo y de referencia, aunque también resalta las limitaciones propias del modelo de lenguaje elegido frente a tareas de razonamiento matemático avanzado. En conjunto, los resultados avalan la viabilidad del enfoque propuesto, indicando que mejoras futuras en la precisión podrían obtenerse mediante la optimización del sistema RAG en las búsquedas por similitud, ampliación y limpieza de la base de conocimiento, mejora del dataset de

validación, o incorporación de herramientas de cálculo para modelos de lenguaje, siguiendo las últimas tendencias en Agentic AI.

# 5 DESPLIEGUE DEL SISTEMA

---

*Admiro su pureza. Es un superviviente al que no afectan la conciencia, los remordimientos ni las fantasías de moralidad.*

*- Alien: El octavo pasajero -*

En este capítulo, explicaremos el proceso de despliegue del sistema desarrollado en este proyecto, poniendo especial atención en las decisiones adoptadas para garantizar su portabilidad y compatibilidad en distintos entornos de ejecución. Para ello, el backend del sistema se ha encapsulado en una imagen Docker, con el fin de ofrecer un entorno de ejecución homogéneo e independiente de las particularidades del sistema anfitrión. Como base, se ha partido de una imagen optimizada para la ejecución de modelos de lenguaje con soporte CUDA, lo que permite aprovechar la aceleración por GPU. Dicha imagen ha sido modificada para incluir las dependencias necesarias del sistema, descargar los modelos y cargar los documentos que conforman la base de conocimiento, garantizando así que el contenedor quede listo para su ejecución. Asimismo, para facilitar la gestión del despliegue, se ha empleado Docker Compose, una herramienta de orquestación de contenedores que permite definir de forma declarativa la configuración del contenedor, usar variables de entorno e incluso crear volúmenes para almacenar información de forma persistente. Por otro lado, el frontend ha sido desplegado de forma independiente sin recurrir a contenedores Docker, debido a la simplicidad que ofrece React junto a Vite en la distribución y despliegue de aplicaciones web.

## 5.1 Creación de la imagen Docker del backend

La creación de la imagen Docker del backend constituye un aspecto clave para garantizar la portabilidad y reproducibilidad del sistema. Para ello se ha utilizado un Dockerfile, un fichero que contiene las instrucciones necesarias para construir imágenes personalizadas. La imagen se ha construido a partir de la distribución Docker oficial de PyTorch [111], en su versión optimizada para ejecución con CUDA, con el fin de asegurar la compatibilidad con entornos que dispongan de aceleración por GPU:

### Código 5-1: Imagen base usada en el Dockerfile

```
FROM pytorch/pytorch:2.5.1-cuda12.4-cudnn9-runtime
```

A partir de esta imagen base, se han añadido las dependencias específicas del sistema mediante un fichero requirements.txt, minimizando el almacenamiento en disco gracias a la configuración de instalación sin caché y la eliminación de ficheros temporales:

**Código 5-2: Instalación de las dependencias del sistema**

```

WORKDIR /app

COPY requirements.txt .

RUN apt update && apt install -y --no-install-recommends git \
    && apt autoremove -y \
    && apt clean \
    && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/* \
    && pip install --no-cache-dir -r requirements.txt \
    && rm -rf /root/.cache/pip

```

La imagen creada en este proyecto incorpora un conjunto de scripts orientados a automatizar las tareas críticas para la preparación del contenedor que se pondrá en ejecución. En particular, el script `download_models.py` se encarga de descargar el modelo Llama y el modelo de embedding requeridos desde Hugging Face, mientras que `load_docs.py` permite cargar y procesar la documentación que conforma la base de conocimiento del sistema RAG. Estos ficheros se copian en la imagen junto con la aplicación principal del backend y los documentos PDF que serán procesados y almacenados en la base de datos vectorial:

**Código 5-3: Instalación del código del backend**

```

COPY app.py .
COPY download_models.py .
COPY load_docs.py .
COPY docs ./docs

```

Una vez incorporados, estos scripts de preparación se ejecutan durante la construcción de la imagen, de manera que el contenedor quede listo para su ejecución sin necesidad de realizar procesos adicionales tras el despliegue. Además, nos aseguramos de borrar la caché de Hugging Face tras descargar los modelos, con el objetivo de reducir al máximo el tamaño de la imagen resultante:

**Código 5-4: Ejecución de scripts para preparar el backend**

```

RUN python3 download_models.py \
    && python3 load_docs.py \
    && rm -rf /root/.cache/huggingface/xet

```

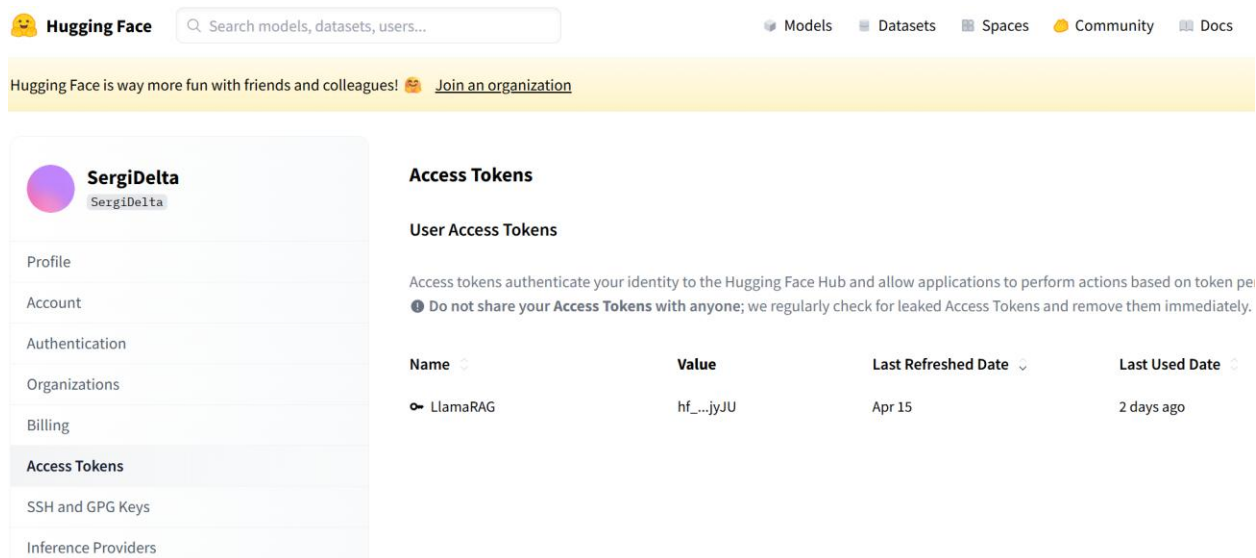
Asimismo, se ha contemplado la gestión segura de credenciales mediante el uso de variables de entorno, debido a que, para descargar el modelo Llama desde Hugging Face, es necesario disponer de un token de acceso al repositorio distribuido por Meta [112]. Para ello, basta con aceptar las condiciones de uso especificadas por Meta y generar un token de acceso desde nuestra cuenta de Hugging Face [113]. Dicho token se define en el Dockerfile como un argumento que será proporcionado en el despliegue del contenedor, y posteriormente se configurará como variable de entorno:

**Código 5-5: Variable de entorno para el token de Hugging Face**

```

ARG HUGGINGFACEHUB_API_TOKEN
ENV HUGGINGFACEHUB_API_TOKEN=$HUGGINGFACEHUB_API_TOKEN

```



Name	Value	Last Refreshed Date	Last Used Date
LlamaRAG	hf_...jyJU	Apr 15	2 days ago

Figura 5-1: Token de acceso para descargar modelos de Hugging Face

Una vez instaladas las dependencias, la imagen resultante tiene como comando principal la ejecución de un servidor Uvicorn [114] encargado de gestionar las peticiones al backend vía API REST. De esta forma, los contenedores que despluguemos a partir de esta imagen ejecutarán el backend y comenzarán a servir peticiones de forma automática, sin necesidad de ejecutar ningún comando adicional. Por defecto, el servidor atiende peticiones en el puerto 5000 y en todas las interfaces de red, pero estos parámetros son configurables mediante las variables de entorno CHATBOT\_API\_PORT y CHATBOT\_API\_HOST:

#### Código 5-6: Ejecución del backend

```
EXPOSE 5000

CMD ["sh", "-c", "uvicorn app:app --host ${CHATBOT_API_HOST:-0.0.0.0} --port ${CHATBOT_API_PORT:-5000}"]
```

Una vez construida la imagen, esta ocupa unos 40 GB en disco, debido a que contiene tanto el modelo Llama como las dependencias necesarias para su ejecución, incluyendo la librería PyTorch y las herramientas requeridas por CUDA, lo que incrementa de forma significativa el tamaño final de la imagen.

```
sergio@OMEN16:~$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
sergarlop2/chatbot_backend  1.0         f637c1860efe     6 days ago     40.4GB
sergio@OMEN16:~$ |
```

Figura 5-2: Tamaño en disco de la imagen Docker desarrollada

Para distribuir la imagen Docker de forma eficiente y evitar tener que volver a construirla a partir del Dockerfile, la imagen ha sido subida a Docker Hub bajo el nombre chatbot\_backend. Dicha imagen ocupa unos 15 GB cuando se comprime, y está disponible para su descarga en el siguiente repositorio:

[https://hub.docker.com/repository/docker/sergarlop2/chatbot\\_backend](https://hub.docker.com/repository/docker/sergarlop2/chatbot_backend)

The screenshot displays the Docker Hub interface for a repository named 'sergarlop2/chatbot\_backend:1.0'. The top navigation bar includes 'Explore' and 'My Hub' tabs, along with a search bar. The left sidebar shows the user's profile and navigation options like 'Repositories', 'Collaborations', 'Settings', 'Billing', and 'Usage'. The main content area shows the repository details, including the index digest, OS/ARCH (linux/amd64), compressed size (15.4 GB), last pushed (6 days by sergarlop2), type (Image), and manifest digest. Below this, the 'Image Layers' section is expanded, showing a list of layers with their names and sizes. The layers include ARG RELEASE, ARG LAUNCHPAD\_BUILD\_ARCH, LABEL org.opencontainers.image.ref.name=ubuntu, LABEL org.opencontainers.image.version=22.04, ADD file ... in / (29.03 MB), CMD ["/bin/bash"], and ARG PYTORCH\_VERSION=2.5.1. The 'Command' section shows 'ARG RELEASE'.

Figura 5-3: Repositorio Docker Hub de la imagen desarrollada

En conclusión, la creación de esta imagen personalizada asegura que cualquier usuario o sistema que despliegue contenedores a partir de ella disponga de un entorno idéntico e independiente del sistema anfitrión, garantizando que el backend esté listo para su ejecución con un esfuerzo de configuración mínimo.

## 5.2 Despliegue del backend con Docker Compose

El despliegue del backend desarrollado se ha llevado a cabo mediante Docker Compose, con el objetivo de simplificar la creación del contenedor y facilitar la configuración de los parámetros necesarios para su ejecución. Docker Compose permite definir, en un único fichero declarativo, todos los aspectos relacionados con la puesta en marcha del contenedor, incluyendo la construcción de la imagen, las variables de entorno, la exposición de puertos, el uso de volúmenes persistentes, y la configuración del hardware.

En este proyecto, el fichero docker-compose.yml especifica la definición del servicio chatbot\_backend, que representa el backend del sistema. Este servicio está configurado para construir la imagen a partir del Dockerfile descrito en el apartado 5.1, permitiendo además inyectar el token de acceso de Hugging Face como argumento de compilación. De esta manera, la descarga de los modelos se realiza durante la construcción de la imagen, sin necesidad de descargarlos tras desplegar el contenedor:

### Código 5-7: Construcción de la imagen con argumentos de compilación

```
version: "3.8"

services:
  chatbot_backend:
    build:
      context: .
      dockerfile: Dockerfile
    args:
      HUGGINGFACEHUB_API_TOKEN: ${HUGGINGFACEHUB_API_TOKEN}
```

Una vez construida la imagen, el contenedor se crea a partir de la misma en su versión etiquetada como `sergarlop2/chatbot_backend:1.0`, lo que garantiza la consistencia en el despliegue, facilita la distribución de la aplicación en distintos entornos, y ayuda a mantener un control de versiones. Adicionalmente, se le asigna un nombre al contenedor para simplificar las tareas de gestión una vez desplegado:

#### Código 5-8: Definición del contenedor y la imagen

```
container_name: chatbot_backend
image: sergarlop2/chatbot_backend:1.0
```

Para habilitar la comunicación con el exterior, se ha configurado la exposición del puerto definido en la variable de entorno `CHATBOT_API_PORT`, que se encuentra especificada en un fichero de variables de entorno. Esto permite modificar fácilmente el puerto sin necesidad de alterar el código fuente o el Dockerfile de la imagen, garantizando una mayor flexibilidad en el despliegue:

#### Código 5-9: Configuración de puertos y de variables de entorno

```
ports:
  - "${CHATBOT_API_PORT}:${CHATBOT_API_PORT}"
env_file:
  - .env
```

Con el fin de asegurar la continuidad del servicio, incluso tras reinicios del sistema anfitrión, se ha configurado la directiva `restart` con la política `unless-stopped`. Esto implica que el contenedor se reiniciará automáticamente salvo que se haya detenido manualmente, mejorando la robustez del servicio:

#### Código 5-10: Política de reinicio del contenedor

```
restart: unless-stopped
```

Un aspecto esencial de este despliegue es el uso de GPU para la ejecución del modelo Llama, lo cual se habilita mediante la sección `deploy`. En este bloque se reservan los recursos de hardware requeridos, indicando que el servicio debe ejecutarse con el controlador NVIDIA y utilizando todas las GPU disponibles en el sistema anfitrión:

#### Código 5-11: Configuración de recursos con soporte GPU

```
deploy:
  resources:
    reservations:
      devices:
        - driver: nvidia
          count: all
          capabilities: [gpu]
```

Para garantizar la persistencia de la información cargada en el sistema, se ha configurado un volumen denominado `chatbot_docs`. Este volumen permite almacenar los documentos y la base de datos del sistema RAG de forma persistente, evitando que se pierdan tras eliminar el contenedor. Con ello, nos aseguramos de que la base de conocimiento no pierda información nueva en el caso de que se despliegue un nuevo contenedor. Esto se debe a que, por defecto, los contenedores Docker se crean con una copia limpia del sistema de ficheros de la imagen original, siendo necesario el uso de volúmenes para almacenar información de forma persistente:

### Código 5-12: Configuración de un volumen persistente

```
# Volume config to persist docs data
volumes:
  - chatbot_docs:/app/docs

volumes:
  chatbot_docs:
    name: chatbot_docs
```

Finalmente, para desplegar el contenedor que ejecuta el backend, basta con situarse en el mismo directorio en el que se encuentra el fichero `docker-compose.yml` y ejecutar el comando “`docker compose up`”:

```
sergio@OMEN16:~/chatbot_backend$ docker compose up
WARN[0000] /home/sergio/chatbot_backend/docker-compose.yml: the attribute `version` is obsolete,
ase remove it to avoid potential confusion
[+] Running 2/2
  ✓ Network chatbot_backend_default   Created
  ✓ Container chatbot_backend         Created
Attaching to chatbot_backend
Loading checkpoint shards: 100%|██████████| 4/4 [00:22<00:00, 5.58s/it]
chatbot_backend | Device set to use cuda:0
chatbot_backend | INFO:      Started server process [7]
chatbot_backend | INFO:      Waiting for application startup.
chatbot_backend | INFO:      Application startup complete.
chatbot_backend | INFO:      Uvicorn running on http://0.0.0.0:5000 (Press CTRL+C to quit)
chatbot_backend | INFO:      172.19.0.1:35272 - "OPTIONS /chat/completions HTTP/1.1" 200 OK
chatbot_backend | INFO:      172.19.0.1:35272 - "POST /chat/completions HTTP/1.1" 200 OK
```

Figura 5-4: Ejecución del backend con docker compose

De forma similar, podemos ejecutar el contenedor en segundo plano con la opción “`-d`”, y parar su ejecución con el comando “`docker compose down`”:

```
sergio@OMEN16:~/chatbot_backend$ docker compose up -d
WARN[0000] /home/sergio/chatbot_backend/docker-compose.yml: the attribute `version` is obsolete,
ase remove it to avoid potential confusion
[+] Running 2/2
  ✓ Network chatbot_backend_default   Created
  ✓ Container chatbot_backend         Started
sergio@OMEN16:~/chatbot_backend$ docker ps
CONTAINER ID   IMAGE                                COMMAND                                CREATED
NAMES
b8bed7f41f96   sergarlop2/chatbot_backend:1.0      "sh -c 'uvicorn app:..."           About a minute ago
0:5000->5000/tcp, [::]:5000->5000/tcp  chatbot_backend
sergio@OMEN16:~/chatbot_backend$ docker compose down
WARN[0000] /home/sergio/chatbot_backend/docker-compose.yml: the attribute `version` is obsolete,
ase remove it to avoid potential confusion
[+] Running 2/2
  ✓ Container chatbot_backend         Removed
  ✓ Network chatbot_backend_default   Removed
sergio@OMEN16:~/chatbot_backend$
```

Figura 5-5: Ejecución y parada del backend con docker compose

El uso de Docker Compose aporta una capa adicional de abstracción y simplicidad al proceso de despliegue, centralizando en un único fichero la configuración necesaria para ejecutar el backend del sistema. Gracias a ello,

se puede poner en marcha el servicio de manera reproducible, consistente y con poco esfuerzo de configuración.

### 5.3 Despliegue del frontend

El despliegue del frontend de este proyecto se ha realizado mediante Vite [115], una herramienta de desarrollo y compilación de interfaces de usuario que destaca por su rapidez y eficiencia. Vite aprovecha la carga nativa de módulos y librerías en el navegador, lo que permite reducir de forma notable los tiempos de arranque de la aplicación. Además, incluye un sistema de compilación optimizado capaz de generar un conjunto de ficheros estáticos listos para ser distribuidos en producción.

Este proceso se apoya en Node.js [116], que actúa como servidor en las fases de desarrollo y compilación. Node se encarga de gestionar las dependencias del proyecto, levantar el servidor local y orquestar la construcción de la aplicación final. Gracias a ello, la aplicación puede desarrollarse y probarse de manera fluida, y posteriormente distribuirse como un conjunto de archivos independientes, preparados para ejecutarse en cualquier servidor web convencional.

En combinación con React, Vite ofrece un flujo de trabajo ágil, con recarga instantánea de módulos y compatibilidad nativa con JavaScript y TypeScript. Esto permite que la fase de desarrollo sea más interactiva y eficiente, simplificando además la preparación del frontend para entornos reales y garantizando un rendimiento óptimo en producción.

Durante la etapa de desarrollo, podemos ejecutar el frontend en un servidor local con Node mediante el comando “npm run dev”. Este servidor expone la aplicación en un puerto configurable, que por defecto es el 5173, pudiendo realizar pruebas y cambios en el código en tiempo real, sin necesidad de recompilar la aplicación por completo:

```
sergio@OMEN16:~/chatbot_frontend$ npm run dev
> chatbot_frontend@0.0.0 dev
> vite

VITE v6.3.5 ready in 361 ms

  → Local:    http://localhost:5173/
  → Network:  http://10.255.255.254:5173/
  → Network:  http://172.26.27.85:5173/
  → press h + enter to show help
```

Figura 5-6: Ejecución del frontend con Node

Una vez finalizada la fase de desarrollo, Vite permite compilar la aplicación y generar una versión optimizada mediante el comando “npm run build”. Este comando genera un conjunto de ficheros HTML, CSS y JavaScript, y los agrupa en el directorio “dist”. Estos ficheros pueden ser desplegados en cualquier servidor web o servicio de hosting estático, como Nginx, Apache, GitHub Pages o plataformas en la nube, sin necesidad de que Node.js esté presente en el entorno de producción:

```
sergio@OMEN16:~/chatbot_frontend$ npm run build

> chatbot_frontend@0.0.0 build
> tsc -b && vite build

vite v6.3.5 building for production...
✓ 39 modules transformed.
dist/index.html          0.46 kB | gzip: 0.30 kB
dist/assets/index-Bkhj9275.css 10.86 kB | gzip: 2.46 kB
dist/assets/index-uttxMIKK.js 199.88 kB | gzip: 62.73 kB
✓ built in 1.25s
sergio@OMEN16:~/chatbot_frontend$ ls -l dist/
total 252
drwxr-xr-x 2 sergio sergio  4096 Aug 26 11:02 assets
-rw-r--r-- 1 sergio sergio 245884 Aug 26 11:02 fading_logo.png
-rw-r--r-- 1 sergio sergio   463 Aug 26 11:02 index.html
sergio@OMEN16:~/chatbot_frontend$ |
```

Figura 5-7: Frontend compilado con Node

De esta forma, el despliegue del frontend resulta mucho más sencillo que el del backend, ya que, tras la compilación, la aplicación se reduce a conjunto de ficheros estáticos que pueden instalarse en cualquier servidor web. Esto elimina la necesidad de contar con un entorno de ejecución complejo con dependencias de gran tamaño, garantizando así una mayor portabilidad y facilidad en la distribución de la aplicación en múltiples plataformas. Gracias a esta simplicidad, el frontend puede desplegarse de manera ligera e independiente, sin tener que llegar a encapsularlo en una imagen Docker.

# 6 CONCLUSIONES Y LÍNEAS FUTURAS

---

*Cien fracasos no importarían si un simple logro pudiese cambiar el destino del mundo.*

*- Arthur C. Clarke -*

En el capítulo final de esta memoria, se llevará a cabo una reflexión sobre los objetivos alcanzados tras la realización del proyecto, destacando las principales aportaciones del sistema propuesto, así como las dificultades encontradas durante su desarrollo. Asimismo, tomando como referencia los resultados y la experiencia obtenida, se plantearán posibles líneas de trabajo orientadas a mejorar y ampliar el sistema presentado en este Trabajo Fin de Máster.

## 6.1 Conclusiones

El desarrollo de este Trabajo Fin de Máster ha permitido llevar a cabo la adaptación de un modelo de lenguaje open source, como Llama 3.1, hacia un asistente especializado en el ámbito de los sistemas de comunicaciones digitales, alcanzando todos los objetivos inicialmente planteados. El proyecto ha demostrado la viabilidad de combinar un LLM con un sistema RAG, proporcionando un marco flexible y escalable para la generación de respuestas precisas y contextualizadas en un dominio técnico específico.

Entre las principales virtudes del sistema desarrollado destaca su capacidad de especialización sin necesidad de volver a entrenar el modelo de lenguaje. Gracias a la incorporación del sistema RAG, es posible dotar al asistente de conocimiento actualizado y específico mediante la gestión de una base de datos documental, lo que le permite responder con mayor precisión a consultas técnicas y conceptuales. Esta característica también abre la puerta a su adaptación a otros campos de conocimiento distintos al de las comunicaciones digitales, ya que bastaría con modificar el system prompt y reemplazar los documentos de la base de datos por aquellos propios del nuevo dominio, posibilitando la creación de asistentes especializados en áreas como la biología, astronomía, geología, derecho o cualquier otro ámbito que requiera documentos de referencia.

Otra fortaleza del proyecto radica en el diseño de una arquitectura modular y fácilmente desplegable. La implementación de una API REST y de una interfaz web permite que el sistema sea accesible desde distintos entornos y dispositivos. Asimismo, la creación de una imagen Docker optimizada, ligera y compacta, que contiene tanto el backend del sistema como el modelo Llama, facilita su despliegue en prácticamente cualquier infraestructura, eliminando la necesidad de volver a descargar el modelo y reduciendo los requisitos del hardware.

El uso de LangChain ha sido uno de los principales aciertos de este proyecto, al facilitar la implementación del sistema RAG y la integración con el modelo de lenguaje mediante una serie de utilidades y abstracciones que

simplifican el desarrollo. Su capacidad para gestionar flujos complejos de razonamiento y consulta lo convierte en una herramienta versátil no solo para sistemas RAG, sino también para arquitecturas más avanzadas que requieren coordinar distintos modelos y organizar múltiples flujos de información.

Por otra parte, la elección de FastAPI para la creación de la API REST ha aportado varias ventajas, como su alto rendimiento, su sencillez de uso y la generación automática de documentación, lo que facilitó tanto el desarrollo como la integración del backend. De igual forma, el ecosistema open source de Hugging Face ha resultado esencial para este proyecto, ya que proporciona acceso a modelos, librerías y herramientas que han agilizado el proceso de experimentación, adaptación y despliegue del modelo Llama. En lo que respecta al diseño de la interfaz, la elección de React resultó adecuada por su capacidad para desarrollar aplicaciones web dinámicas de manera sencilla y modular. Gracias a sus componentes reutilizables y a la gestión eficiente del estado de la aplicación, fue posible construir una interfaz ligera y funcional que ofrece al usuario una experiencia fluida e intuitiva.

En cuanto a la validación del sistema, el asistente alcanzó una precisión global del 70%, un valor razonable si se tiene en cuenta la limitación de trabajar con un modelo relativamente pequeño como es Llama 3.1 8B. Además, durante la evaluación del modelo, se observó que parte de los errores estaban asociados a la falta de una depuración más exhaustiva de los documentos almacenados en la base de datos, especialmente en fórmulas y desarrollos matemáticos, lo que abre la puerta a mejoras significativas a través de un proceso de limpieza y curación más riguroso de los datos.

El desarrollo del sistema también ha planteado desafíos significativos, cuya resolución ha resultado fundamental para el éxito del proyecto. Entre ellos, destaca la necesidad de equilibrar el tiempo de inferencia del modelo con la precisión de las respuestas, ajustando parámetros de configuración y cuantización del modelo, así como la habilitación de la aceleración por GPU. Por otro lado, la optimización del sistema RAG para maximizar la relevancia de la información recuperada supuso un esfuerzo notable, incluyendo la elección adecuada del algoritmo de búsqueda semántica, la estrategia de segmentación de documentos, y el modelo elegido para la creación de embeddings. De igual manera, la definición de un system prompt lo más óptimo posible requirió pruebas iterativas para garantizar que el modelo interpretara correctamente el contexto de las consultas, así como para estructurar de forma adecuada sus respuestas y dotarlo del comportamiento digno de un tutor especialista.

Un reto adicional fue la generación del dataset de validación para cubrir los aspectos más relevantes de las comunicaciones digitales, realizada de forma manual para garantizar una mayor fiabilidad y transparencia en los resultados obtenidos. Esta tarea demandó un tiempo y esfuerzo considerable, dada la necesidad de recopilar un conjunto de preguntas y respuestas representativo y equilibrado, que permitiera evaluar de manera exhaustiva el desempeño del asistente sin depender de automatizaciones externas. Finalmente, la creación de una imagen Docker lo más ligera posible supuso también un reto técnico adicional, al requerir la integración eficiente del backend y del modelo cuantizado, asegurando compatibilidad, portabilidad y facilidad de despliegue.

En términos generales, en este proyecto no solo se han logrado los objetivos planteados, sino que se han sentado las bases para futuras mejoras y extensiones. El diseño modular del sistema, su capacidad de adaptación a distintos dominios de conocimiento y la facilidad de despliegue constituyen elementos que lo consolidan como una herramienta de gran potencial en entornos educativos, de investigación y profesionales, permitiendo la creación de asistentes especializados que aporten valor añadido mediante la combinación de inteligencia artificial y recuperación de información de fuentes contrastadas.

## 6.2 Líneas futuras

En base a los resultados y experiencias obtenidos en el proyecto, a continuación, enumeramos los distintos aspectos que podrían mejorarse, así como nuevas funcionalidades o ampliaciones del sistema:

- **Inclusión de nuevos modelos de lenguaje:** Se podrían integrar nuevos modelos open source en el sistema, como Mistral, Gemma, o versiones actualizadas de Llama, de forma que el usuario pueda elegir el modelo con el que desea interactuar. Por otro lado, el uso de modelos más grandes permitiría mejorar la precisión y coherencia de las respuestas generadas, así como ampliar la capacidad de gestión de contextos más extensos y consultas complejas. No obstante, esto supondría un aumento del tiempo de

inferencia y la necesidad de disponer de hardware con mayores prestaciones.

- **Mejor capacidad de cálculo mediante el uso de herramientas:** La integración de llamadas a herramientas externas basadas en ejecución de código permitiría al modelo delegar operaciones matemáticas a funciones específicas. De esta manera, el modelo Llama, o uno más avanzado, podría identificar que una consulta requiere realizar un cálculo, determinar los argumentos necesarios para la función a ejecutar (suma, multiplicación, integral definida, etc), y emplear el resultado obtenido en su respuesta. Este enfoque, siguiendo la tendencia actual de los agentes de IA, evitaría alucinaciones en los cálculos realizados por el propio modelo y garantizaría una mayor precisión en la resolución de problemas matemáticos o técnicos.
- **Ampliación y limpieza de la base de datos vectorial:** Una revisión y depuración más exhaustiva de los documentos almacenados en la base de datos permitiría eliminar inconsistencias, errores de formato y duplicidades, especialmente en contenidos con fórmulas y desarrollos matemáticos. Además, se podría ampliar la cobertura temática del asistente incluyendo documentos adicionales, como artículos de investigación, capítulos de libros científicos y especificaciones técnicas de la ITU, entre otros.
- **Dotación de mayor flexibilidad al sistema RAG:** Sería conveniente que el usuario pudiera configurar los parámetros del sistema de recuperación de información, como por ejemplo, la forma de segmentar los documentos, incluyendo el número y tamaño de los fragmentos, el solapamiento entre ellos y el modelo de embedding utilizado para la vectorización. Esta flexibilidad favorecería una adaptación más precisa a distintos tipos de documentos y formatos.
- **Ampliación del dataset de validación:** La generación de un dataset de validación más amplio y representativo permitiría realizar evaluaciones más robustas del sistema, facilitando la detección de errores y áreas de mejora. Incluir nuevos tipos de preguntas, escenarios de consulta más complejos y casos prácticos contribuiría a aumentar la fiabilidad de las métricas de desempeño, especialmente después de ampliar la información almacenada en el sistema RAG.
- **Implementación de sesiones de chat independientes:** Actualmente, el frontend solo soporta una sesión de chat única. La implementación de sesiones múltiples permitiría al usuario mantener conversaciones separadas sobre distintos temas o proyectos, facilitando la organización de la información y ofreciendo una experiencia más cercana a la de asistentes como ChatGPT o Gemini.
- **Mejora de la seguridad de la API y del frontend:** La creación de un sistema de autenticación con registro e inicio de sesión, junto con el uso de tokens en la API, garantizaría un acceso controlado al asistente. Además, la habilitación de HTTPS en la API y el frontend en entornos de producción protegería la confidencialidad de los datos transmitidos y reforzaría la privacidad de los usuarios de la aplicación.
- **Creación de bases de conocimiento independientes:** La posibilidad de disponer de múltiples bases de conocimiento, ya sea por usuario o por área temática, permitiría optimizar la recuperación de información y ofrecer una mayor capacidad de personalización. Esta característica también facilitaría la gestión de permisos y el mantenimiento de documentos específicos para distintas personas o grupos de trabajo.



## REFERENCIAS

- [1] M. A. K. Raiaan, M. S. H. Mukta, K. Fatema, N. M. Fahad, S. Sakib, M. M. J. Mim, J. Ahmad, M. E. Ali y S. Azam, «A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges,» *IEEE Access*, 2024.
- [2] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes y A. Mian, «Comprehensive Overview of Large Language Models,» *arXiv preprint*, 2024.
- [3] A. Solanki, «Advancements in Artificial Intelligence: A Comprehensive Review and Future Prospects,» *International Journal of Artificial Intelligence Research and Development (IJAIRD)*, 2024.
- [4] OpenAI, «GPT-4 Technical Report,» 2023. [En línea]. Available: <https://openai.com/index/gpt-4-research/>.
- [5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser y I. Polosukhin, «Attention Is All You Need,» *Proc. 31st Conf. Neural Information Processing Systems (NIPS 2017)*, 2017.
- [6] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [7] A. Gupta, «How Vector Embeddings and Vector Databases works?,» 2023. [En línea]. Available: <https://aayushtechblog.hashnode.dev/how-vector-embeddings-and-vector-databases-works>.
- [8] «Hugging Face Meta Llama,» [En línea]. Available: <https://huggingface.co/meta-llama>.
- [9] «Llama 3.1 405b vs Leading Closed-Source Models,» 2024. [En línea]. Available: <https://www.vellum.ai/blog/evaluating-llama-3-1-405b-against-leading-closed-source-competitors>.
- [10] Universidad de Sevilla, «Sistemas de Comunicaciones, Máster Universitario en Ingeniería de Telecomunicación,» [En línea]. Available: <https://www.us.es/estudiar/que-estudiar/oferta-de-masteres/master-universitario-en-ingenieria-de-telecomunicacion-0/52040012>.
- [11] LangChain, «LangChain, the platform for reliable agents,» [En línea]. Available: <https://www.langchain.com/>.
- [12] ChromaDB, «Chroma is the open-source AI application database. Batteries included.,» [En línea]. Available: <https://www.trychroma.com/>.
- [13] «Hugging Face All MpNet Base V2,» [En línea]. Available: <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>.
- [14] «Hugging Face Transformers,» [En línea]. Available: <https://huggingface.co/docs/transformers/index>.

- [15] React, «React, la biblioteca para interfaces de usuario web y nativas,» [En línea]. Available: <https://es.react.dev/>.
- [16] FastAPI, «FastAPI framework, high performance, easy to learn, fast to code, ready for production,» [En línea]. Available: <https://fastapi.tiangolo.com/>.
- [17] R. Gozalo-Brizuela y E. C. Garrido-Merchán, «A survey of Generative AI Applications,» *arXiv preprint arXiv:2306.02781v2*, 2023.
- [18] L. Triess, «Generative vs Discriminative models,» [En línea]. Available: [https://www.linkedin.com/posts/ltruess\\_generativeai-data-neuralnetworks-activity-7100857248603615233-U4Gb/](https://www.linkedin.com/posts/ltruess_generativeai-data-neuralnetworks-activity-7100857248603615233-U4Gb/).
- [19] D. P. Kingma y M. Welling, «Auto-Encoding Variational Bayes,» *arXiv preprint arXiv:1312.6114*, 2013.
- [20] R. Winter, F. Noé y D.-A. Clevert, «Auto-Encoding Molecular Conformations,» *arXiv preprint, arXiv:2101.01618*, 2021.
- [21] A. Sharma, «Variational Autoencoder in TensorFlow,» [En línea]. Available: <https://learnopencv.com/variational-autoencoder-in-tensorflow/>.
- [22] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville y Y. Bengio, «Generative Adversarial Nets,» *arXiv preprint arXiv:1406.2661v1*, 2014.
- [23] T. Karras, S. Laine y T. Aila, «A Style-Based Generator Architecture for Generative Adversarial Networks,» *arXiv preprint arXiv:1812.04948v3*, 2019.
- [24] M. Bansal, «Generative Adversarial Networks (Beginner's Guide),» [En línea]. Available: <https://levelup.gitconnected.com/generative-adversarial-networks-unlocking-the-power-of-artificial-creativity-beginners-guide-7353a70a8709>.
- [25] J. Ho, A. Jain y P. Abbeel, «Denoising Diffusion Probabilistic Models,» *arXiv preprint arXiv:2006.11239*, 2020.
- [26] P. Dhariwal y A. Nichol, «Diffusion Models Beat GANs on Image Synthesis,» *arXiv preprint arXiv:2105.05233*, 2021.
- [27] OpenAI, «DALL·E 3,» [En línea]. Available: <https://openai.com/index/dall-e-3/>.
- [28] A. Vahdat, «The forward and backward processes of the diffusion model,» [En línea]. Available: [https://www.researchgate.net/figure/The-forward-and-backward-processes-of-the-diffusion-model-The-credit-of-the-used-images\\_fig1\\_382128283](https://www.researchgate.net/figure/The-forward-and-backward-processes-of-the-diffusion-model-The-credit-of-the-used-images_fig1_382128283).
- [29] A. Radford, K. Narasimhan, T. Salimans y I. Sutskever, «Improving Language Understanding by Generative Pre-Training,» *OpenAI*, 2018.
- [30] Wikipedia, «Transformador generativo preentrenado,» [En línea]. Available: [https://es.wikipedia.org/wiki/Transformador\\_generativo\\_preentrenado](https://es.wikipedia.org/wiki/Transformador_generativo_preentrenado).

- [31] P. Isola, J.-Y. Zhu, T. Zhou y A. A. Efros, «Image-to-Image Translation with Conditional Adversarial Networks,» *arXiv preprint arXiv:1611.07004*, 2017.
- [32] R. Rombach, A. Blattmann, D. Lorenz, P. Esser y B. Ommer, «High-Resolution Image Synthesis with Latent Diffusion Models,» *arXiv preprint arXiv:2112.10752v2*, 2022.
- [33] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior y K. Kavukcuoglu, «WaveNet: A Generative Model for Raw Audio,» *arXiv preprint arXiv:1609.03499*, 2016.
- [34] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue y A. Roberts, «GANSynth: Adversarial Neural Audio Synthesis,» *arXiv preprint arXiv:1902.08710*, 2019.
- [35] W. Yan, Y. Zhang, P. Abbeel y A. Srinivas, «VideoGPT: Video Generation using VQ-VAE and Transformers,» *arXiv preprint arXiv:2104.10157*, 2021.
- [36] S. Tulyakov, M.-Y. Liu, X. Yang y J. Kautz, «MoCoGAN: Decomposing Motion and Content for Video Generation,» *arXiv preprint arXiv:1807.04759*, 2018.
- [37] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger y I. Sutskever, «Learning Transferable Visual Models From Natural Language Supervision,» *arXiv preprint, arXiv:2103.00020*, 2021.
- [38] Google, «MUM: A new AI milestone for understanding information,» 2021. [En línea]. Available: <https://blog.google/products/search/introducing-mum/>.
- [39] Bitvero, «What is Dall-E 2: How does it work?,» [En línea]. Available: <https://www.bitvero.in/what-is-dall-e-2-how-does-it-work/>.
- [40] L. F. Freedman, «Lawyers Sanctioned for Citing AI Generated Fake Cases,» [En línea]. Available: <https://natlawreview.com/article/lawyers-sanctioned-citing-ai-generated-fake-cases>.
- [41] Wikipedia, «Cross-entropy,» [En línea]. Available: <https://en.wikipedia.org/wiki/Cross-entropy>.
- [42] S. Takase, K. Yamazaki, R. Zhao, K. Kawamura, I. E. Yen, H. Kajino y Y. Aoyama, «Large Vocabulary Size Improves Large Language Models,» *arXiv preprint arXiv:2406.16508 [cs.CL]*, 2024.
- [43] S. Ranjan, «How do language models (LLM) work? We call it ChatGPT,» [En línea]. Available: <https://www.linkedin.com/pulse/how-do-language-modelsllm-work-we-call-chatgpt-mishra-fdqsc/>.
- [44] A. Holtzman, J. Buys, L. Du, M. Forbes y Y. Choi, «The Curious Case of Neural Text Degeneration,» *arXiv preprint arXiv:1904.09751*, 2020.
- [45] OpenAI, «Learning to reason with LLMs,» 2024. [En línea]. Available: <https://openai.com/index/learning-to-reason-with-llms/>.
- [46] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le y D. Zhou, «Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,» *arXiv preprint arXiv:2201.11903*, 2023.
- [47] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan y e. al, «Language Models are Few-Shot Learners,» *arXiv preprint arXiv:2005.14165*, 2020.

- [48] C. Wolfe, «Advanced Prompt Engineering,» [En línea]. Available: <https://medium.com/data-science/advanced-prompt-engineering-f07f9e55fe01>.
- [49] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel y D. Kiela, «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,» *arXiv preprint arXiv:2005.11401*, 2021.
- [50] A. Chintala, «Retrieval Augmented Generation (RAG): What, Why and How?,» [En línea]. Available: <https://lmlstack.ai/blog/retrieval-augmented-generation>.
- [51] T. Mikolov, K. Chen, G. Corrado y J. Dean, «Efficient Estimation of Word Representations in Vector Space,» *arXiv preprint arXiv:1301.3781*, 2013.
- [52] J. Pennington, R. Socher y C. D. Manning, «GloVe: Global Vectors for Word Representation,» *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.
- [53] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,» *arXiv preprint arXiv:1810.04805*, 2019.
- [54] Pinecone, «Pinecone's mission is to make AI knowledgeable,» [En línea]. Available: <https://www.pinecone.io/>.
- [55] Weaviate, «The AI-native database for a new generation of software,» [En línea]. Available: <https://weaviate.io/>.
- [56] S. Maameri, «Building a Multi-Document Reader and Chatbot With LangChain and ChatGPT,» [En línea]. Available: <https://medium.com/better-programming/building-a-multi-document-reader-and-chatbot-with-langchain-and-chatgpt-d1864d47e339>.
- [57] Y. A. M. a. D. A. Yashunin, «Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs,» *arXiv preprint arXiv:1603.09320*, 2016.
- [58] Wikipedia, «K-Nearest Neighbors algorithm,» [En línea]. Available: [https://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm).
- [59] R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi y D. Petrelli, «Hybrid Search: Effectively Combining Keywords and Semantic Searches,» *European Semantic Web Conference, Tenerife, Spain*, 2008.
- [60] Wikipedia, «BM25,» [En línea]. Available: [https://en.wikipedia.org/wiki/Okapi\\_BM25](https://en.wikipedia.org/wiki/Okapi_BM25).
- [61] J. Carbonell y J. Goldstein, «The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries,» *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '98), Melbourne, Australia*, 1998.
- [62] C. Li, «OpenAI's GPT-3 Language Model: A Technical Overview,» 2020. [En línea]. Available: <https://lambdalabs.com/blog/demystifying-gpt-3>.
- [63] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal y e. al,

- «Training language models to follow instructions with human feedback,» *arXiv preprint arXiv:2203.02155*, 2022.
- [64] J. Schulman, F. Wolski, P. Dhariwal, A. Radford y O. Klimov, «Proximal Policy Optimization Algorithms,» *arXiv preprint arXiv:1707.06347*, 2017.
- [65] S. Casper, X. Davies, C. Shi, T. K. Gilbert, J. Scheurer, J. Rando, R. Freedman, T. Korbak, D. Lindner, P. Freire, T. Wang, S. Marks, C.-R. Segerie, M. Carroll, A. Peng, P. Christoffersen, M. Damani, S. Slocum, U. Anwar, A. Siththaranjan, M. Nadeau y E. J., «Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback,» *arXiv preprint arXiv:2307.15217v2*, 2023.
- [66] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano y G. Irving, «Fine-Tuning Language Models from Human Preferences,» *arXiv preprint arXiv:1909.08593*, 2019.
- [67] M. T. R. Laskar, S. Alqahtani, M. S. Bari, M. Rahman, M. A. M. Khan, H. Khan y a. et, «A Systematic Survey and Critical Review on Evaluating Large Language Models: Challenges, Limitations, and Recommendations,» *arXiv preprint arXiv:2407.04069*, 2024.
- [68] A. B. M. A. Rahman, S. Anwar, M. Usman y A. Mian, «DefAn: Definitive Answer Dataset for LLMs Hallucination Evaluation,» *arXiv preprint arXiv:2406.09155*, 2024.
- [69] Y. Fan, Y. Liu, Z. Yao, J. Yu, L. Hou y J. Li, «Evaluating Generative Language Models in Information Extraction as Subjective Question Correction,» *arXiv preprint arXiv:2404.03532*, 2024.
- [70] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song y J. Steinhardt, «Measuring Massive Multitask Language Understanding,» *arXiv preprint arXiv:2009.03300*, 2021.
- [71] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda y a. et, «Evaluating Large Language Models Trained on Code,» *arXiv preprint arXiv:2107.03374*, 2021.
- [72] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi y Y. Choi, «HellaSwag: Can a Machine Really Finish Your Sentence?,» *arXiv preprint arXiv:1905.07830*, 2019.
- [73] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse y J. Schulman, «Training Verifiers to Solve Math Word Problems,» *arXiv preprint arXiv:2110.14168*, 2021.
- [74] «Chatbot Arena (formerly LMSYS): Free AI Chat to Compare & Test Best AI Chatbots,» [En línea]. Available: <https://lmarena.ai/>.
- [75] xAI, «Grok 3 Beta — The Age of Reasoning Agents,» 2025. [En línea]. Available: <https://x.ai/news/grok-3>.
- [76] D. Li, B. Jiang, L. Huang, A. Beigi, C. Zhao, Z. Tan, A. Bhattacharjee, Y. Jiang, C. Chen, T. Wu, K. Shu, L. Cheng y H. Liu, «From Generation to Judgment: Opportunities and Challenges of LLM-as-a-Judge,» *arXiv preprint arXiv:2411.16594*, 2024.
- [77] OpenAI, «Hello GPT-4o,» 2024. [En línea]. Available: <https://openai.com/index/hello-gpt-4o/>.
- [78] OpenAI, «Introducing GPT-5,» [En línea]. Available: <https://openai.com/index/introducing-gpt-5/>.
- [79] B. Goose, «How GPT-5 Compares to GPT-4o,» [En línea]. Available:

- <https://medium.com/@leucopsis/how-gpt-5-compares-to-gpt-4o-b493d1b8812b>.
- [80] Anthropic, «Claude 3.5 Sonnet,» 2024. [En línea]. Available: <https://www.anthropic.com/news/claude-3-5-sonnet>.
- [81] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li y Z. Gao, «DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning,» *arXiv preprint arXiv:2501.12948*, 2025.
- [82] Meta, «Introducing Llama 3.1: Our most capable models to date,» 2024. [En línea]. Available: <https://ai.meta.com/blog/meta-llama-3-1/>.
- [83] Alibaba, «Qwen2.5-Max: Exploring the Intelligence of Large-scale MoE Model,» 2025. [En línea]. Available: <https://qwenlm.github.io/blog/qwen2.5-max/>.
- [84] Alibaba, «Alibaba Cloud's Qwen Models Attract over 90,000 Enterprise Adoptions Within its First Year,» 2024. [En línea]. Available: [https://www.alibabacloud.com/blog/alibaba-clouds-qwen-models-attract-over-90000-enterprise-adoptions-within-its-first-year\\_601130](https://www.alibabacloud.com/blog/alibaba-clouds-qwen-models-attract-over-90000-enterprise-adoptions-within-its-first-year_601130).
- [85] Google, «Introducing Gemini 2.0: our new AI model for the agentic era,» 2024. [En línea]. Available: <https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/#ceo-message>.
- [86] Meta, «The Llama 3 Herd of Models,» *arXiv preprint arXiv:2407.21783v3*, 2024.
- [87] H. Touvron, L. Martin, K. Stone y e. al, «Llama 2: Open Foundation and Fine-Tuned Chat Models,» *arXiv preprint arXiv:2307.09288v2*, 2023.
- [88] J. Ainslie, J. Lee-Thorp, M. d. Jong, Y. Zemlyanskiy, F. Lebrón y S. Sanghai, «GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints,» *arXiv preprint arXiv:2305.13245*, 2023.
- [89] OpenAI, «Tiktoken,» [En línea]. Available: <https://github.com/openai/tiktoken>.
- [90] Meta, «Llama 3.1 Model Card,» [En línea]. Available: [https://github.com/meta-llama/llama-models/blob/main/models/llama3\\_1/MODEL\\_CARD.md](https://github.com/meta-llama/llama-models/blob/main/models/llama3_1/MODEL_CARD.md).
- [91] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning y C. Finn, «Direct Preference Optimization: Your Language Model is Secretly a Reward Model,» *arXiv preprint arXiv:2305.18290v3*, 2024.
- [92] «Hugging Face TRL,» [En línea]. Available: <https://huggingface.co/docs/trl/index>.
- [93] «Hugging Face PEFT,» [En línea]. Available: <https://huggingface.co/docs/peft/index>.
- [94] «Hugging Face Datasets,» [En línea]. Available: <https://huggingface.co/docs/datasets/index>.
- [95] Meta, «Model Cards & Prompt formats Llama 3.1,» [En línea]. Available: [https://www.llama.com/docs/model-cards-and-prompt-formats/llama3\\_1/](https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_1/).

- [96] Ollama, «Llama 3.1,» [En línea]. Available: <https://ollama.com/library/llama3.1>.
- [97] K. Paul, «Meta says its Llama AI models being used by banks, tech companies,» Reuters, [En línea]. Available: <https://www.reuters.com/technology/artificial-intelligence/meta-says-its-llama-ai-models-being-used-by-banks-tech-companies-2024-08-29/>.
- [98] Accenture, «Accenture Pioneers Custom Llama LLM Models with NVIDIA AI Foundry,» [En línea]. Available: <https://newsroom.accenture.com/news/2024/accenture-pioneers-custom-llama-llm-models-with-nvidia-ai-foundry>.
- [99] The Scale Team, «Meta and Scale Partner to Drive Enterprise Adoption of Llama 3.1 405B Using Scale GenAI Platform,» [En línea]. Available: <https://scale.com/blog/meta-llama-3-1-launch-partner>.
- [100] «Unsloth,» [En línea]. Available: <https://unsloth.ai/>.
- [101] Google, «NotebookLM,» [En línea]. Available: <https://notebooklm.google/>.
- [102] E. Tufino, «NotebookLM: An LLM with RAG for active learning and collaborative tutoring,» *arXiv preprint arXiv:2504.09720v2*, 2025.
- [103] S. Johnson, «NotebookLM gets a new look, audio interactivity and a premium version,» [En línea]. Available: <https://blog.google/technology/google-labs/notebooklm-new-features-december-2024/>.
- [104] «Reglamento (UE) 2016/679,» Boletín Oficial del Estado, [En línea]. Available: <https://www.boe.es/buscar/doc.php?id=DOUE-L-2016-80807>.
- [105] OpenAI, «New tools for building agents,» 2025. [En línea]. Available: <https://openai.com/index/new-tools-for-building-agents/>.
- [106] HP, «Portátil gaming OMEN 16 pulgadas Intel,» [En línea]. Available: <https://www.hp.com/es-es/gaming-pc/laptops/2023-omen-16-intel.html>.
- [107] W.-M. Lee, «Optimizing Vector Search for Your RAG Applications,» [En línea]. Available: <https://ai.gopubby.com/optimizing-vector-search-for-your-rag-applications-3040993d5059>.
- [108] OpenAPI, «The world's most widely used API description standard,» [En línea]. Available: <https://www.openapis.org/>.
- [109] OpenAI, «OpenAI platform API reference,» [En línea]. Available: <https://platform.openai.com/docs/api-reference/introduction>.
- [110] Postman, «AI needs context. APIs deliver it.,» [En línea]. Available: <https://www.postman.com/>.
- [111] Docker Hub, «Pytorch 2.5.1 CUDA 12.4 cuDNN9 Runtime,» [En línea]. Available: <https://hub.docker.com/layers/pytorch/pytorch/2.5.1-cuda12.4-cudnn9-runtime/images/sha256-c8268a92a69bd500f8be0e665b2630ee006dadaf7bfbcb24249141b15ff622755>.
- [112] «Hugging Face Gated models,» [En línea]. Available: <https://huggingface.co/docs/hub/models-gated>.
- [113] «Hugging Face User access tokens,» [En línea]. Available: <https://huggingface.co/docs/hub/security-tokens>.

[114] Uvicorn, «An ASGI web server, for Python,» [En línea]. Available: <https://www.uvicorn.org/>.

[115] Vite, «The Build Tool for the Web,» [En línea]. Available: <https://vite.dev/>.

[116] Node.js, «Run JavaScript Everywhere,» [En línea]. Available: <https://nodejs.org/es/>.



# ANEXO

En este anexo se incluyen los ficheros de código más relevantes del backend, junto con el Dockerfile utilizado para crear su imagen Docker y el fichero docker-compose.yml empleado para desplegarlo como contenedor. El código completo del backend y del frontend está disponible en sus respectivos repositorios GitHub:

Backend: [https://github.com/sergarlop2/chatbot\\_backend](https://github.com/sergarlop2/chatbot_backend)

Frontend: [https://github.com/sergarlop2/chatbot\\_frontend](https://github.com/sergarlop2/chatbot_frontend)

## download\_models.py

```
from langchain_huggingface import HuggingFaceEmbeddings
from transformers import AutoModelForCausalLM, AutoTokenizer
from dotenv import load_dotenv, find_dotenv
import os
import torch

EMBEDDING_MODEL_NAME = "sentence-transformers/all-mpnet-base-v2"
LLAMA_MODEL_NAME = "meta-llama/Llama-3.1-8B-Instruct"

#_ = load_dotenv(find_dotenv())
HF_TOKEN = os.getenv("HUGGINGFACEHUB_API_TOKEN")

embedding = HuggingFaceEmbeddings(
    model_name=EMBEDDING_MODEL_NAME,
    model_kwargs={"device": str(torch.device("cuda" if
torch.cuda.is_available() else "cpu"))},
    encode_kwargs={"normalize_embeddings": True}
)

llm = AutoModelForCausalLM.from_pretrained(
    LLAMA_MODEL_NAME,
    torch_dtype=torch.float16,
    device_map="auto",
    token=HF_TOKEN
)

tokenizer = AutoTokenizer.from_pretrained(LLAMA_MODEL_NAME, token=HF_TOKEN)
```

**load\_docs.py**

```
from langchain_community.document_loaders import PyPDFLoader
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_chroma import Chroma
from langchain_core.documents import Document
import torch
import os

EMBEDDING_MODEL_NAME = "sentence-transformers/all-mpnet-base-v2"
DOCS_DIR = "docs/"
PERSIST_DIR = "docs/chroma/"
CHUNK_SIZE_PAGES = 5 # Number of pages per chunk
CHUNK_OVERLAP_PAGES = int(0.2 * CHUNK_SIZE_PAGES) # 20% overlap

# Get pdf files from the docs directory
pdf_files = [os.path.join(DOCS_DIR, f) for f in os.listdir(DOCS_DIR) if
f.endswith(".pdf")]

# Load PDFs
docs = []
for pdf_file in pdf_files:
    loader = PyPDFLoader(pdf_file)
    docs.extend(loader.load())

# Group pages by source document
doc_groups = {}
for doc in docs:
    source = doc.metadata["source"]
    if source not in doc_groups:
        doc_groups[source] = []
    doc_groups[source].append(doc)

# Create chunks with overlap
split_docs = []
for source, pages in doc_groups.items():
    # Order pages and prepare for chunking
    sorted_pages = sorted(pages, key=lambda x: x.metadata["page"])
    total_pages = len(sorted_pages)
    start = 0

    while start < total_pages:
        end = start + CHUNK_SIZE_PAGES
        chunk_pages = sorted_pages[start:end]

        # Combine text and metadata
        combined_text = "\n\n".join([p.page_content for p in chunk_pages])
        metadata = {
            "source": source,
```

```
        "start_page": chunk_pages[0].metadata["page"],
        "end_page": chunk_pages[-1].metadata["page"],
        "total_chunk_pages": len(chunk_pages)
    }

    split_docs.append(Document(
        page_content=combined_text,
        metadata=metadata
    ))

    start += (CHUNK_SIZE_PAGES - CHUNK_OVERLAP_PAGES)

# Add chunk-specific metadata
for idx, chunk in enumerate(split_docs):
    chunk.metadata.update({
        "chunk_id": idx,
        "chunk_length": len(chunk.page_content)
    })

# Create embeddings and persist in Chroma
embedding = HuggingFaceEmbeddings(
    model_name=EMBEDDING_MODEL_NAME,
    model_kwargs={"device": str(torch.device("cuda" if
torch.cuda.is_available() else "cpu"))},
    encode_kwargs={"normalize_embeddings": True} # better for cosine
similarity
)
vectordb = Chroma.from_documents(
    documents=split_docs,
    embedding=embedding,
    persist_directory=PERSIST_DIR,
    collection_metadata={"hnsw:space": "cosine"}
)

print("Number of document splits:", len(split_docs))
print("Number of collections in DB:", vectordb._collection.count())
print(f"Example text: {split_docs[18]}")
```

**app.py**

```
import sys
import os
import time
import torch
import uuid
import shutil
from dotenv import load_dotenv, find_dotenv
from transformers import pipeline, AutoTokenizer, AutoModelForCausalLM,
BitsAndBytesConfig
from langchain_chroma import Chroma
from langchain_core.documents import Document
from langchain_community.document_loaders import PyPDFLoader
from langchain_huggingface import HuggingFaceEmbeddings, HuggingFacePipeline
from fastapi import FastAPI, HTTPException
from fastapi import UploadFile, File
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import FileResponse
from pydantic import BaseModel
from typing import List, Optional

torch.manual_seed(42)

# Read env variables
# _ = load_dotenv(find_dotenv())
HF_TOKEN = os.getenv("HUGGINGFACEHUB_API_TOKEN")

DOCS_FOLDER = "docs/"
PERSIST_DIR = "docs/chroma/"
CHUNK_SIZE_PAGES = 5 # Number of pages per chunk
CHUNK_OVERLAP_PAGES = int(0.2 * CHUNK_SIZE_PAGES) # 20% overlap

LLAMA_MODEL_NAME = "meta-llama/Llama-3.1-8B-Instruct"
EMBEDDING_MODEL_NAME = "sentence-transformers/all-mpnet-base-v2"

CHAT_MODELS = {
    "llama-3.1-8b-instruct": {
        "id": "llama-3.1-8b-instruct",
        "name": LLAMA_MODEL_NAME,
        "object": "model",
        "owned_by": "user"
    }
}

# Schemas for requests and responses
class Message(BaseModel):
    role: str # "system", "user" and "assistant"
    content: str
```

```
class ChatRequest(BaseModel):
    model: str
    messages: List[Message]
    use_rag: Optional[bool] = False

class UploadResponse(BaseModel):
    filename: str
    message: str

class DeleteResponse(BaseModel):
    message: str

# Load the embedding model from HuggingFace
embedding = HuggingFaceEmbeddings(
    model_name=EMBEDDING_MODEL_NAME,
    model_kwargs={"device": str(torch.device("cuda" if
torch.cuda.is_available() else "cpu"))},
    encode_kwargs={"normalize_embeddings": True}
)
vectordb = Chroma(
    persist_directory=PERSIST_DIR,
    embedding_function=embedding
)
retriever = vectordb.as_retriever(
    search_kwargs={"k": 5}
)

# Load quantized Llama model
quant_config = BitsAndBytesConfig(load_in_8bit=True, llm_int8_threshold=6.0)
llm = AutoModelForCausalLM.from_pretrained(
    LLAMA_MODEL_NAME,
    torch_dtype=torch.float16,
    device_map="auto",
    quantization_config=quant_config,
    token=HF_TOKEN
)

# Load tokenizer
tokenizer = AutoTokenizer.from_pretrained(LLAMA_MODEL_NAME, token=HF_TOKEN)
tokenizer.pad_token = tokenizer.eos_token

# Text generation pipeline
llm_pipe = pipeline(
    "text-generation",
    model=llm,
    tokenizer=tokenizer,
```

```
    temperature=sys.float_info.epsilon, # Use a very low temperature for
deterministic output
    max_new_tokens=128000,
    eos_token_id=tokenizer.eos_token_id,
    pad_token_id=tokenizer.pad_token_id,
    return_full_text=False
)

# Wrap the pipeline in HuggingFacePipeline (LangChain)
hf_pipeline = HuggingFacePipeline(pipeline=llm_pipe)

# Initialize FastAPI app and enable CORS
app = FastAPI(docs_url="/swagger", redoc_url="/redoc")
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Endpoints
@app.get(
    "/models",
    tags=["Models"],
    summary="List available models",
    description="Returns a list of available language models that can be used
to generate responses."
)
def list_models():
    return {"object": "list", "data": list(CHAT_MODELS.values())}

@app.get(
    "/models/{model_id}",
    tags=["Models"],
    summary="Get model details",
    description="Returns the metadata of a specific model."
)
def get_model(model_id: str):
    if model_id not in CHAT_MODELS:
        raise HTTPException(status_code=404, detail="Model not found")
    return CHAT_MODELS[model_id]

@app.post(
    "/chat/completions",
    tags=["Chat"],
    summary="Send messages to the language model",
    description=""
```

```

        Generates a response from the language model based on the provided
        messages.
        If `use_rag` is set to True, a search is performed in a vector
        database to provide additional
        context in the prompt.
        """
    )
def create_completion(request: ChatRequest):
    if request.model not in CHAT_MODELS:
        raise HTTPException(status_code=400, detail="Unsupported model")

    # Create a copy of the messages to avoid modifying the original request
    messages = request.messages.copy()

    # Check if RAG is enabled and retrieve documents if necessary
    if request.use_rag:
        user_message = messages[-1].content.strip()
        docs = retriever.invoke(user_message)
        context = "\n\n".join([d.page_content for d in docs])

        # New user message with context
        rag_prompt = (
            f"Context: ```{context}```\n\n"
            f"Question: ```{user_message}```\n\n"
            f"Answer:"
        )

        # Replace the last user message with the new one with context
        messages.pop()
        messages.append(Message(role="user", content=rag_prompt))

    # Build the prompt from messages (Llama 3.1 chat template)
    prompt = "<|begin_of_text|>"
    for msg in messages:
        role = msg.role
        content = msg.content.strip()
        prompt +=
f"<|start_header_id|>{role}<|end_header_id|>\n\n{content}\n\n"
        prompt += "<|eot_id|><|start_header_id|>assistant<|end_header_id|>"

    # Generate response
    start_time = time.time()
    response_text = hf_pipeline.invoke(prompt)
    elapsed_time = time.time() - start_time

    comp_id = str(uuid.uuid4())
    created = int(time.time())

```

```
response = {
    "id": comp_id,
    "object": "chat.completion",
    "created": created,
    "model": request.model,
    "message": {
        "role": "assistant",
        "content": response_text,
    },
    "elapsed_time": elapsed_time
}

# Add sources if RAG is enabled
if request.use_rag:
    response["sources"] = [doc.metadata for doc in docs]

return response

@app.get(
    "/docs",
    tags=["Documents"],
    summary="List stored PDF documents",
    description="Returns the list of PDF documents stored in the RAG system."
)
def list_documents():
    try:
        files = [
            f for f in os.listdir(DOCS_FOLDER)
            if os.path.isfile(os.path.join(DOCS_FOLDER, f)) and
            f.lower().endswith(".pdf")
        ]
        return {"docs": files}
    except Exception as e:
        raise HTTPException(status_code=500, detail="Could not list
documents")

@app.get(
    "/docs/{filename}",
    tags=["Documents"],
    summary="Get a PDF document",
    description="Returns the requested PDF document if it exists in the RAG
system."
)
def get_pdf_document(filename: str):
    if ".." in filename or filename.startswith("/"):
        raise HTTPException(status_code=400, detail="Invalid filename")

    file_path = os.path.join(DOCS_FOLDER, filename)
```

```

    if not os.path.isfile(file_path) or not filename.lower().endswith(".pdf"):
        raise HTTPException(status_code=404, detail="Document not found")

    return FileResponse(file_path, media_type="application/pdf",
filename=filename)

@app.put(
    "/docs",
    response_model=UploadResponse,
    tags=["Documents"],
    summary="Upload and vectorize a PDF document",
    description="""
        Uploads a PDF document, splits it into chunks, and stores it in the
vector database
        of the RAG system.
    """
)
def put_pdf(file: UploadFile = File(...)):
    if not file.filename.endswith(".pdf"):
        raise HTTPException(status_code=400, detail="You can only upload PDF
files")

    filename = file.filename
    file_path = os.path.join(DOCS_FOLDER, filename)

    # If the file exists, delete the old file and its chunks from the vector
database
    if os.path.exists(file_path):
        os.remove(file_path)
        source_path = os.path.join(DOCS_FOLDER, filename)
        vectordb._collection.delete(where={"source": source_path})

    # Save the new file
    with open(file_path, "wb") as f:
        shutil.copyfileobj(file.file, f)

    # Load and chunk the new PDF
    loader = PyPDFLoader(file_path)
    pages = loader.load()

    # Order pages and prepare for chunking
    sorted_pages = sorted(pages, key=lambda x: x.metadata["page"])
    total_pages = len(sorted_pages)
    start = 0
    split_docs = []

    while start < total_pages:
        end = start + CHUNK_SIZE_PAGES

```

```
chunk_pages = sorted_pages[start:end]

# Combine text and metadata
combined_text = "\n\n".join([p.page_content for p in chunk_pages])
metadata = {
    "source": file_path,
    "start_page": chunk_pages[0].metadata["page"],
    "end_page": chunk_pages[-1].metadata["page"],
    "total_chunk_pages": len(chunk_pages)
}
split_docs.append(Document(page_content=combined_text,
metadata=metadata))
start += (CHUNK_SIZE_PAGES - CHUNK_OVERLAP_PAGES)

# Add chunk-specific metadata
for idx, chunk in enumerate(split_docs):
    chunk.metadata.update({"chunk_id": idx, "chunk_length":
len(chunk.page_content)})

vectordb.add_documents(split_docs)

return UploadResponse(filename=filename, message="PDF uploaded and
vectorized successfully")

@app.delete(
    "/docs/{filename}",
    response_model=DeleteResponse,
    tags=["Documents"],
    summary="Delete a PDF document and its vectors",
    description="""
        Deletes a PDF document along with its associated chunks stored in the
        vector database
        of the RAG system.
        """)
)
def delete_pdf(filename: str):
    if ".." in filename or filename.startswith("/") or not
filename.endswith(".pdf"):
        raise HTTPException(status_code=400, detail="Invalid filename")

    file_path = os.path.join(DOCS_FOLDER, filename)

    if not os.path.exists(file_path):
        raise HTTPException(status_code=404, detail="File not found")

    os.remove(file_path)

# Remove vectors related to the document
source_path = os.path.join(DOCS_FOLDER, filename)
```

```
vectordb._collection.delete(where={"source": source_path})  
  
return DeleteResponse(message=f"PDF and chunks deleted for: {filename}")
```

## Dockerfile

```
FROM pytorch/pytorch:2.5.1-cuda12.4-cudnn9-runtime

ARG HUGGINGFACEHUB_API_TOKEN
ENV HUGGINGFACEHUB_API_TOKEN=${HUGGINGFACEHUB_API_TOKEN}

WORKDIR /app

COPY requirements.txt .

RUN apt update && apt install -y --no-install-recommends git \
    && apt autoremove -y \
    && apt clean \
    && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/* \
    && pip install --no-cache-dir -r requirements.txt \
    && rm -rf /root/.cache/pip

COPY app.py .
COPY download_models.py .
COPY load_docs.py .
COPY docs ./docs

RUN python3 download_models.py \
    && python3 load_docs.py \
    && rm -rf /root/.cache/huggingface/xet

EXPOSE 5000

CMD ["sh", "-c", "uvicorn app:app --host ${CHATBOT_API_HOST:-0.0.0.0} --port\n${CHATBOT_API_PORT:-5000}"]
```

## docker-compose.yml

```
version: "3.8"

services:
  chatbot_backend:
    build:
      context: .
      dockerfile: Dockerfile
    args:
      HUGGINGFACEHUB_API_TOKEN: ${HUGGINGFACEHUB_API_TOKEN}
    container_name: chatbot_backend
    image: sergarlop2/chatbot_backend:1.0
    ports:
      - "${CHATBOT_API_PORT}:${CHATBOT_API_PORT}"
    env_file:
      - .env
    restart: unless-stopped
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
    # Volume config to persist docs data
    volumes:
      - chatbot_docs:/app/docs

volumes:
  chatbot_docs:
    name: chatbot_docs
```