

RA G

— Retrieval Augmented Generation



Mª Rosa Cuenca

Introducción

¿Por qué RAG?

¿Qué es RAG?

Fases del RAG

**Implementar un
Sistema RAG**

**Preparación e
Ingesta de Datos**

**Vectorización e
Indexación**

**Bases de Datos
Vectoriales**

**Generación de
Respuestas**

Métricas

Ejemplo

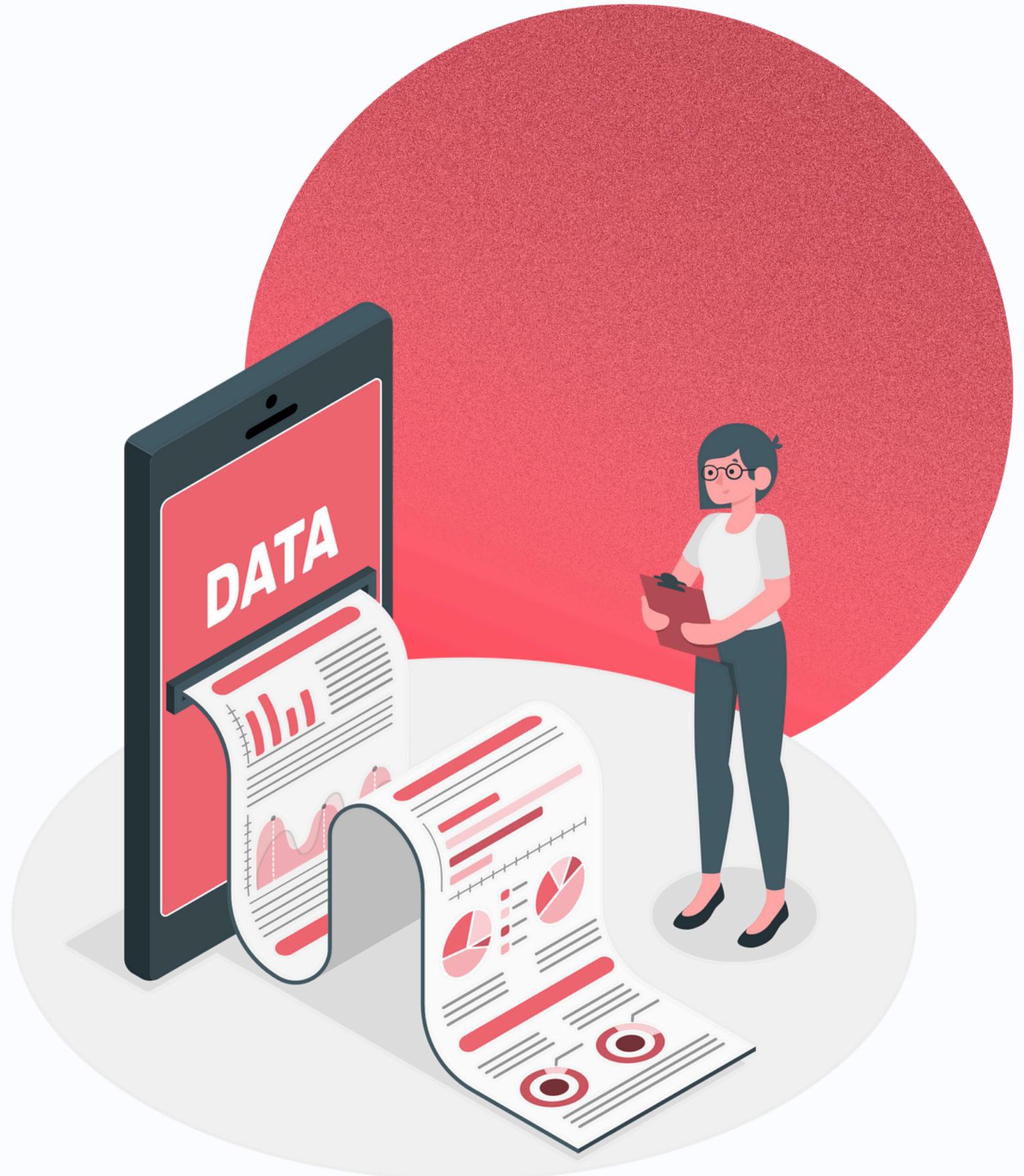
Referencias

Introducción

A estas alturas, los modelos de lenguaje de gran escala (LLM) son ampliamente conocidos. Sin embargo, debido al alto costo del entrenamiento, las organizaciones buscan maneras eficientes de aprovecharlos sin gastar una fortuna. Entonces, ¿Cómo obtener una ventaja competitiva sin gastar una fortuna?

Una solución es emplear la **Generación Aumentada por Recuperación** (RAG).

Esta técnica permite ampliar las capacidades de un LLM utilizando datos propios. Al superar la principal limitación de los LLM—el entrenamiento intensivo y los datos requeridos—las organizaciones pueden mejorar los resultados aprovechando su propia información. Con RAG, sus datos internos se convierten en un factor diferenciador clave.



¿Por qué RAG?

La búsqueda tradicional se centra en palabras clave, lo que puede resultar en información excesiva, insuficiente o incorrecta porque el sistema no comprende el significado detrás de las palabras. Por ejemplo, una consulta basada en palabras clave sobre especies de árboles autóctonos de Francia usando "árboles" y "Francia" puede no recuperar toda la información porque es demasiado literal: los árboles autóctonos de Normandía podrían no incluirse, aunque estén en Francia, porque faltaba esa palabra clave.

Un LLM, funciona generando respuestas basadas únicamente en el conocimiento adquirido durante su entrenamiento previo. Analiza la entrada del usuario y, utilizando patrones y estructuras lingüísticas aprendidas de grandes conjuntos de datos, produce la respuesta que considera apropiada, que puede no ser exacta.

Con RAG, el modelo interpreta la pregunta, busca en datos pertinentes y genera una respuesta basada en la información más relevante, proporcionando respuestas precisas y sin errores. Esta es la lógica básica detrás de un sistema RAG.

O O O

O O O

O O O

O O O

Pero, ¿Qué es RAG?

La generación aumentada de recuperación (RAG) es una técnica que complementa la generación de texto con información de fuentes de datos privadas o propietarias. Combina un modelo de recuperación, que está diseñado para buscar grandes sets de datos o bases de conocimiento, con un modelo de generación como un modelo de lenguaje grande (LLM), que toma esa información y genera una respuesta de texto legible.

RAG puede mejorar una búsqueda, al agregar contexto de fuentes de datos adicionales y complementando la base de conocimientos original de un LLM. Esto mejora el resultado del modelo, sin tener que volver a entrenarlo. Las fuentes de información adicionales pueden variar desde información nueva en Internet en la que el LLM no recibió capacitación hasta contexto comercial propietario o documentos internos confidenciales que pertenecen a empresas.

Implementa métodos de recuperación de búsqueda (normalmente búsqueda semántica o búsqueda híbrida) para responder a la intención del usuario y ofrecer resultados más relevantes.

Fases del RAG

RECUPERACIÓN

El sistema busca información relevante en el conjunto proporcionado e identifica fragmentos de datos relacionados con la consulta realizada. Esto garantizan que la información obtenida sea precisa y actualizada para responder la pregunta.

AUMENTO

Esta información se combina con el contexto de la consulta, para enriquecer al LLM garantizando que este tiene acceso a datos relevantes, estructurados y contextualizados. De esta forma aseguramos respuestas coherentes y fundamentadas.

GENERACIÓN

El modelo utiliza la información aumentada para elaborar una respuesta final, produciendo un texto que responde a la consulta, con consistencia, precisión y claridad, manteniéndose alineado con la base de datos inicial y el contexto proporcionado.

1 RETRIEVAL

RECUPERACIÓN

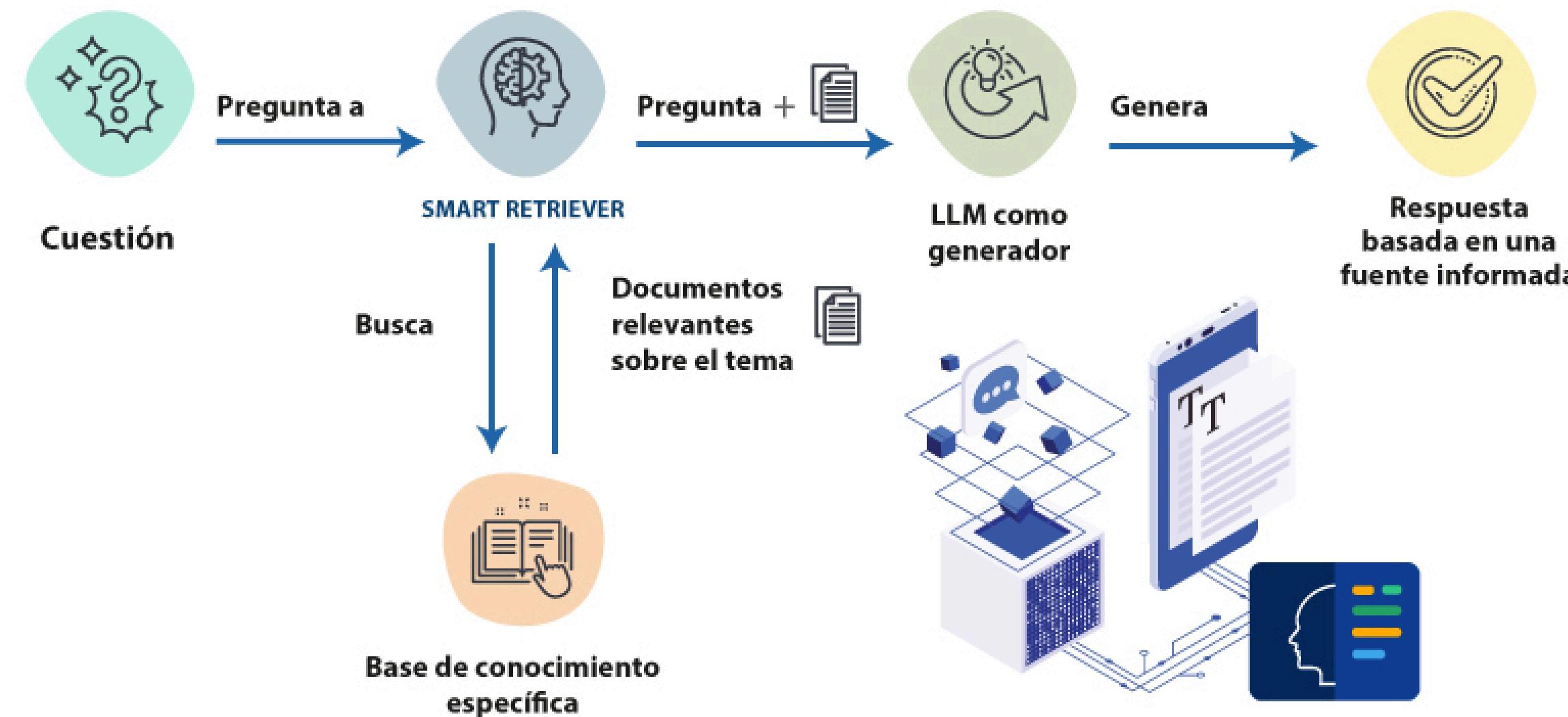
2 AUMENTED

AUMENTO

3 GENERATION

GENERACIÓN

El proceso es el siguiente:

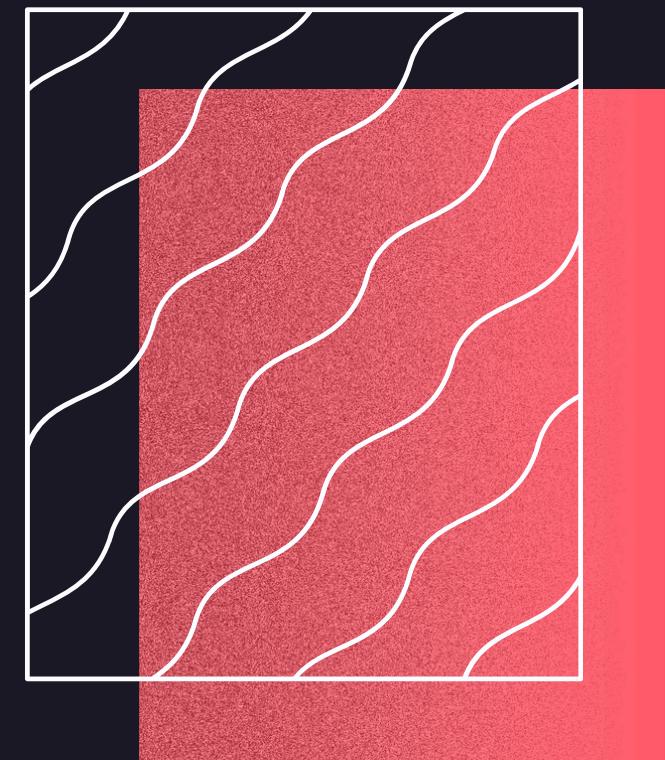


¿Y CÓMO HACE TODO ESO?



IMPLEMENTAR UN SISTEMA RAG

Con **LangChain**, la implementación de un sistema **RAG** se convierte en un proceso modular y eficiente. Al integrar herramientas como **loaders**, **text splitters**, **embeddings**, **bases vectoriales** y **agentes**, podemos construir sistemas potentes capaces de ofrecer respuestas contextualizadas y actualizadas.



Preparación e Ingesta de Datos

Extracción de Contenido

El primer paso es la recopilación y preparación de los datos que serán utilizados como fuente externa. Los datos pueden provenir de documentos, bases de datos, páginas web, o aplicaciones SaaS.

Los **Document Loaders** facilitan la carga de datos desde distintos formatos:

- Para PDF: bibliotecas como **PyMuPDF**, **pypdf** o **pdfminer** para extraer texto.
- Archivos CSV: se pueden cargar con **Pandas**.
- Markdown: como texto plano.
- **LangChain Document Loaders:**
 - **TextLoader**: para documentos en formato .txt o Markdown.
 - **CSVLoader**: para archivos .csv.
 - **PyPDFLoader**: para archivos PDF. Requiere instalar pypdf (pip install pypdf).

Preparación e Ingesta de Datos

Preprocesamiento y División en Fragmentos (Chunks)

Los documentos deben ser divididos en trozos (chunks) para que puedan ser procesados eficientemente por los modelos LLM, ya que estos tienen límites de tokens. Conserva la semántica y facilita consultas precisas en grandes volúmenes de texto.

LangChain **CharacterTextSplitter**:

- Divide por cantidad de caracteres.
- Rápido, pero puede cortar frases.
- Para textos simples

O O O

LangChain **RecursiveCharacterTextSplitter**:

O O O

- Divide texto preservando la estructura semántica: párrafos → oraciones → palabras.
- Preciso, pero más lento.

O O O

- Para textos complejos

O O O

Preparación e Ingesta de Datos

Ejemplo

```
from langchain_community.document_loaders import TextLoader  
  
loader = TextLoader("../how_to/state_of_the_union.txt")  
documents = loader.load()  
text_splitter = CharacterTextSplitter(chunk_size=1000,  
chunk_overlap=0)  
docs = text_splitter.split_documents(documents)
```

Vectorización e Indexación

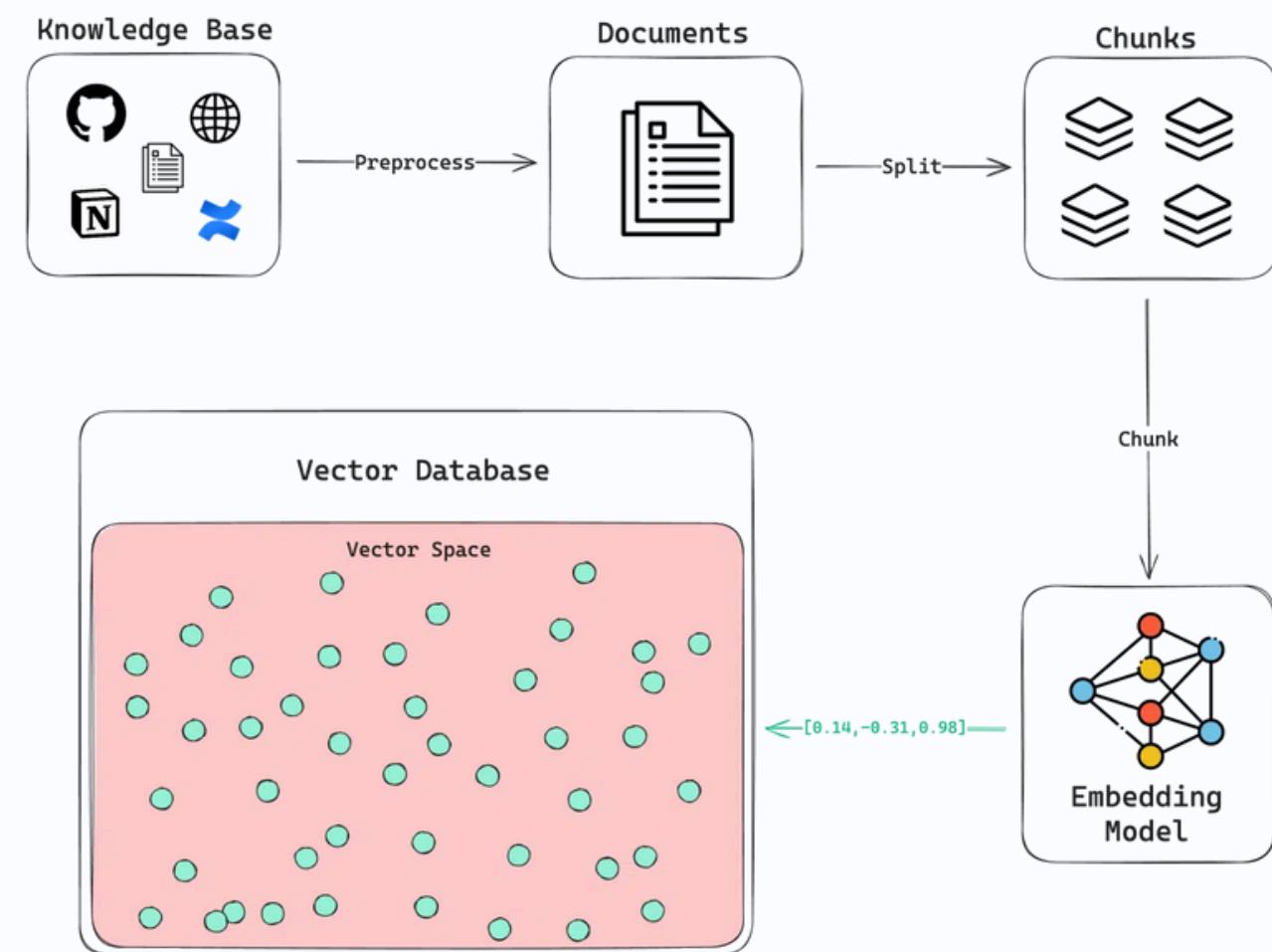
Generación de Embeddings

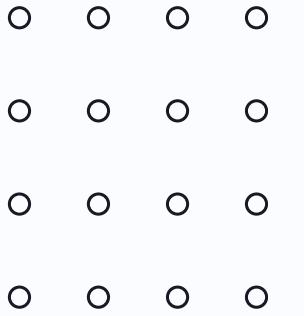
Los **embeddings** son representaciones numéricas de texto que capturan su significado semántico. Este proceso **convierte los chunks de texto en vectores numéricos** para que los modelos puedan analizar, comparar y buscar similitudes entre ellos.

La extracción de estas representaciones numéricas nos permite recuperar eficientemente textos similares. **Cuanto más parecidas sean las representaciones, más similar será el tema del que hablan, y mejor contexto** proporcionarán a la pregunta del usuario.

Estos **Embedding Models** están siendo constantemente desarrollados y mejorados. Para elegir el mejor modelo, puedes utilizar algunas de las **tablas de clasificación** comunes para encontrar el que mejor se adapte a tus requisitos.

LangChain ofrece un extenso **catálogo de integraciones de modelos de representaciones de texto** para la mayoría de los proveedores y modelos.





Vectorización e Indexación

Generación de Embeddings

Los embeddings se generan mediante un modelo que convierte texto en embeddings (como text-embedding-ada-002 de OpenAI), despues cada chunk de texto (fragmento previamente dividido) se pasa al modelo para transformarlo en un vector de números. Resultado: Una lista de **vectores que representan cada chunk de forma matemática**, representando las características semánticas del texto.

```
from langchain_community.embeddings import OpenAIEmbeddings

# Inicializamos el modelo que convierte texto en embeddings
embeddings = OpenAIEmbeddings(model="text-embedding-ada-002")

# Convertimos los chunks de texto en vectores
vectors = embeddings.embed_documents([chunk.page_content for chunk in chunks])
```

Bases de Datos Vectoriales

Una vez generados los embeddings, estos se almacenan en una **base de datos vectorial**.

- Estas permiten realizar búsquedas rápidas basadas en similitudes entre los vectores.
- Almacenan tanto los vectores como los datos originales (o referencias a ellos).
- **FAISS**: Rápido y eficiente para búsquedas de similitud.
- **Chroma**: Sencillo y con soporte avanzado para aplicaciones modernas.
- **Qdrant**: Diseñado para casos de uso escalables.
- **ElasticSearch**: Potente motor de búsqueda y análisis que soporta tanto búsquedas vectoriales como tradicionales basadas en texto.

```
from langchain_community.vectorstores import FAISS
vectorstore = FAISS.from_documents(chunks, embeddings)
```

Texto → Chunks → Embeddings → Almacenamiento en Base Vectorial → **Búsqueda Semántica**: Recuperar los chunks más relevantes comparando los embeddings en la base vectorial



○ ○ ○ ○
○ ○ ○ ○

Generación de Respuestas

En esta etapa, se combinan la **consulta del usuario y los datos recuperados (chunks)** para generar una **respuesta contextualizada** utilizando un **LLM**. Al seleccionar un LLM para un sistema RAG, se deben considerar factores como el tamaño del modelo, los datos de entrenamiento y el rendimiento en tareas de lenguaje relevantes para nuestra app, así como el precio.

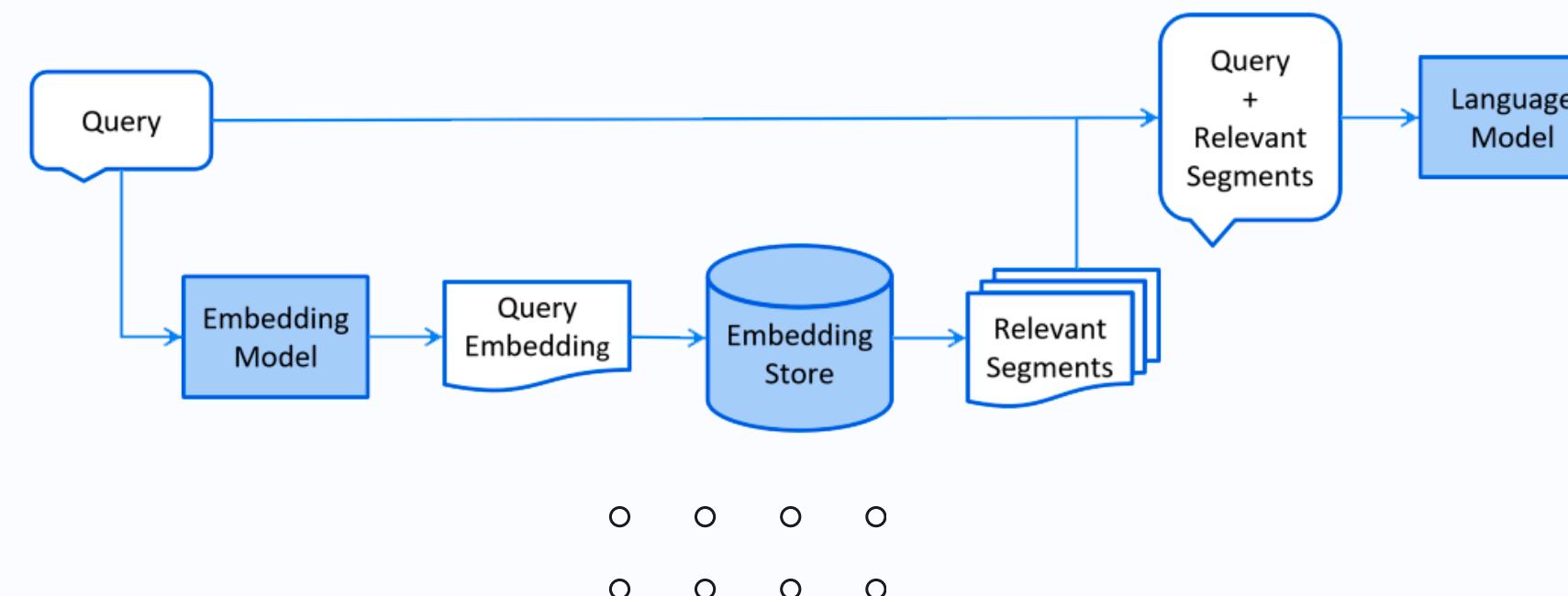
LangChain ofrece una gran compatibilidad con la mayoría de los proveedores.

Responder Citando la Documentación Recuperada

Una vez que se recupera el contexto relevante, se construye mensaje más complejo con una “plantilla de pregunta”.

Esta plantilla instruye al modelo para responder a la consulta de los usuarios con información recuperada de la documentación.

Además, el sistema puede ser diseñado para citar los documentos o secciones específicas que utilizó, proporcionando transparencia y permitiendo a los usuarios profundizar en el tema si es necesario.



Generación de Respuestas

Es crucial diseñar el prompt adecuadamente para que el modelo comprenda tanto la consulta como el contexto.

- **Instrucciones claras sobre límites:** El modelo se limita al contexto proporcionado, evitando suposiciones o invenciones.
- **Gestión de incertidumbre:** Define cómo actuar cuando falta información (respuesta honesta en lugar de especulaciones).
- **Citación explícita:** Incluye referencias específicas para construir confianza y facilitar la exploración adicional por parte del usuario.
- **Estructuración de la respuesta:** Hace que la salida sea organizada y fácil de entender.
- **Optimización del uso de tokens:** Asegúrate de que el prompt sea lo más conciso posible sin perder información relevante.

Eres un asistente de IA para Factoría F5, diseñado para proporcionar respuestas claras, útiles y basadas en datos fiables. Tu objetivo es ayudar a los compañeros de trabajo de Factoría F5 a encontrar información precisa y relevante utilizando los fragmentos de contexto proporcionados.

Por favor, utiliza únicamente los datos proporcionados en el contexto para responder a la pregunta.

Si no encuentras suficiente información en el contexto, responde con: "No tengo suficiente información para responder a esta pregunta con precisión."

Siempre cita explícitamente los fragmentos o secciones del contexto que respaldan tu respuesta para mayor transparencia.

Contexto disponible: {contexto}

Pregunta: {pregunta}

Respuesta estructurada:

- Respuesta: [Tu respuesta aquí basada en el contexto]
- Fuentes: [Lista de fragmentos o secciones utilizados del contexto]

¿Cuánto tiempo dura el curso de IA?

Respuesta estructurada:

- Respuesta: El curso de IA en Factoría F5 tienen una duración de 9 meses.
- Fuentes: Fragmento 3 del contexto.

MÉTRICAS EVALUAR Y OPTIMIZAR



Evaluación de Métricas en LLMs

Las métricas evalúan el desempeño y la calidad de las respuestas de los LLMs. Estas pueden clasificarse en:

Métricas basadas en LLMs:

- El propio modelo calcula métricas de evaluación según especificaciones predefinidas.
- Ejemplo: Usar un LLM para calificar la coherencia o precisión de otra respuesta.

Métricas tradicionales de NLP:

Herramientas externas que comparan respuestas generadas con la verdad de referencia

- **BLEU**: Evalúa coincidencias de palabras o frases en el texto.
- **ROUGE**: Mide similitudes en palabras clave o frases importantes.
- **Text Match/String Presence**: Verifica la presencia literal de palabras específicas.



Evaluación en RAG

En RAG, se evalúan dos bloques principales:

Recuperación

Verifica si los fragmentos relevantes se recuperan correctamente.

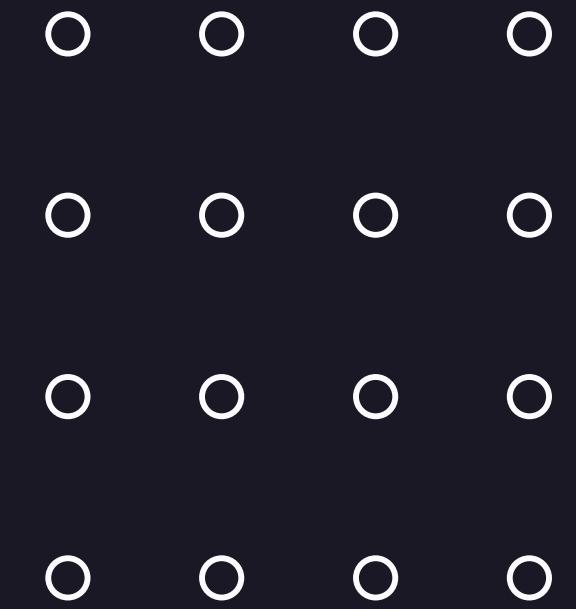
Métrica clave: Context Recall (recall del contexto):

- Mide qué tan bien la información recuperada coincide con la verdad de referencia.
- Valores entre 0 y 1 (más alto, mejor)

Generación:

Verifica la consistencia de las respuestas generadas con los datos del contexto y la verdad de referencia usando las métricas:

- Fidelidad
- Relevancia
- Corrección
- Similitud semántica



EJEMPLO DE CONTEXT RECALL

RECUPERACIÓN

Preguntamos:
"¿Dónde está la Torre Eiffel?".

Contexto:
"París es una ciudad de Francia conocida por puntos de referencia como la Torre Eiffel".

La verdad de referencia sería:
"La Torre Eiffel está en París".

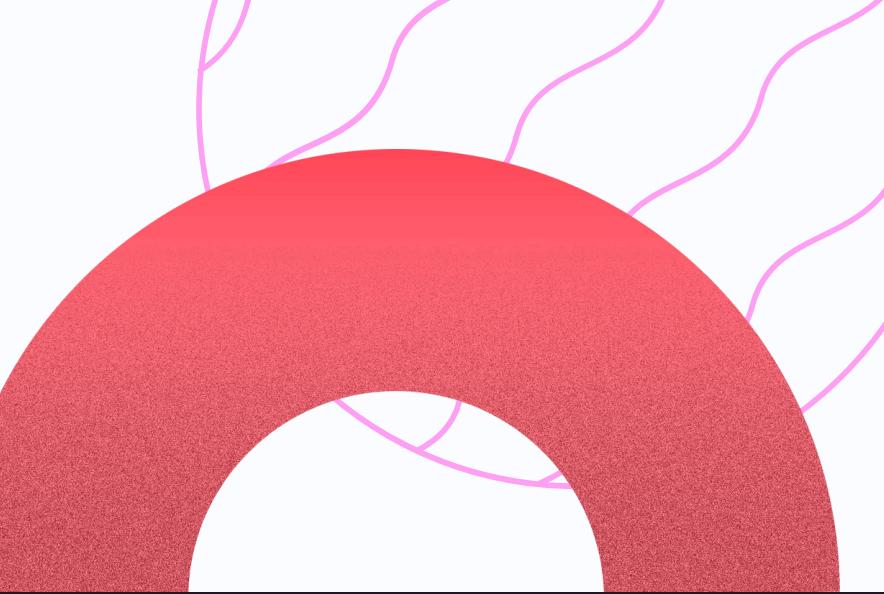
Si el modelo responde correctamente ("La Torre Eiffel está en París"), la métrica "Context Recall" será 1.

AUMENTO

Preguntamos:
"¿Dónde y cuándo nació Einstein?"

Con el contexto que indica que Einstein nació en Alemania el 14 de marzo de 1879.

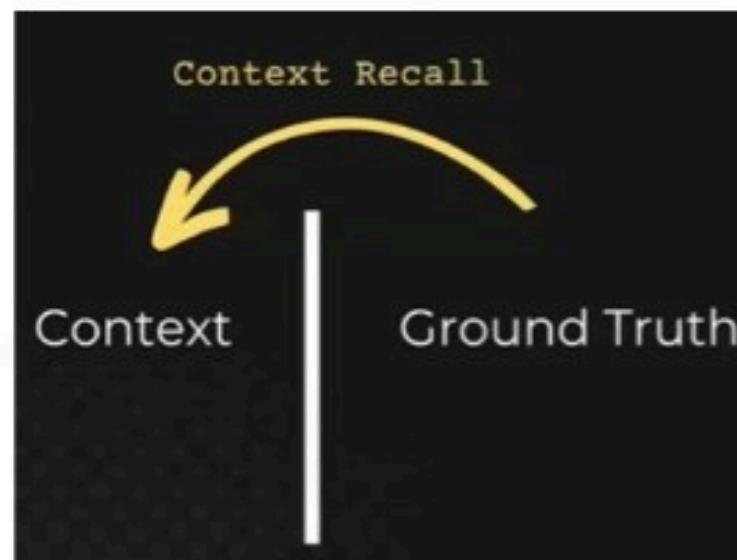
Si el modelo responde
"En Alemania el 20 de marzo",
la métrica de generación sería baja (Ej: 0.5), ya que acertó el país pero erró la fecha.



Retrieval

Context Recall

Evaluates how many relevant fragments are retrieved correctly. Range between 0-1. The higher the better.



Context Recall Calculation

Example

```
1 {
  'question': 'Where is the Eiffel Tower located?',
  'contexts': 'Paris is a major city in France known for its landmarks like the Eiffel Tower.',
  'ground_truths': ['The Eiffel Tower is located in Paris'],
```

LLM



```
'answer': 'The Eiffel Tower is located in Paris'
```

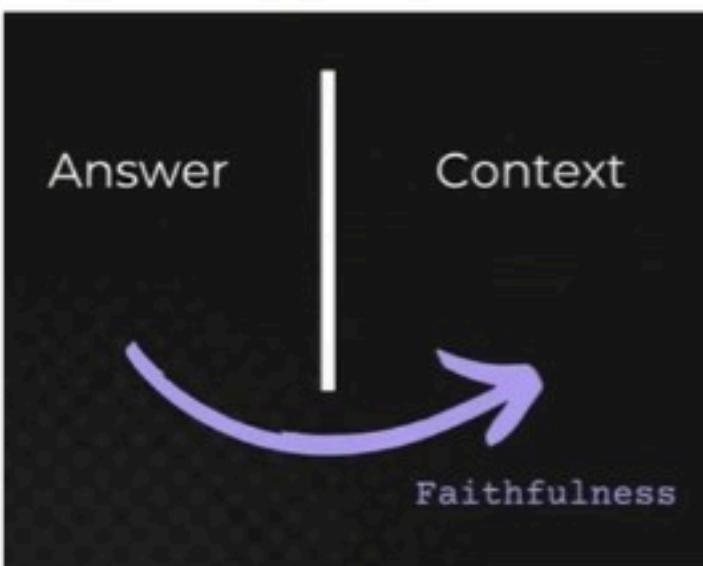
$$\text{context recall} = \frac{|\text{GT claims that can be attributed to context}|}{|\text{Number of claims in GT}|}$$

$$\text{Context Recall} = \frac{1}{1} = 1.0$$

Generation

Faithfulness

Measures consistency of generated facts in the answer, against a given context. Range between 0-1. The higher the better.



Faithfulness Score Calculation

Example

Example

Question: Where and when was Einstein born?

Context: Albert Einstein (born 14 March 1879) was a German-born theoretical physicist, widely held to be one of the greatest and most influential scientists of all time

High faithfulness answer: Einstein was born in Germany on 14th March 1879.

Low faithfulness answer: Einstein was born in Germany on 20th March 1879.

LLM

$$\text{Faithfulness Score} = \frac{\text{Number of claims in the generated answer that can be inferred from the given context}}{\text{Total number of claims in the generated answer}}$$

$$\text{Faithfulness} = \frac{1}{2} = 0.5$$

NOTEBOOK DE EJEMPLO



REFERENCIAS

- [¿Qué es la generación aumentada de recuperación \(RAG\)?](#)
- Amazon: [¿Qué es la RAG \(generación aumentada por recuperación\)?](#)
- Elasticsearch: [¿Qué es la RAG \(generación aumentada de recuperación\)?](#)
- [Beyond RAG basics: Advanced strategies for AI applications](#)
- [Vector search and state of the art retrieval for Generative AI apps | BRK206H](#)
- [Langchain](#)
- [RAG - Retrieval Augmented Generation: La llave que abre la puerta de la precisión a los modelos del lenguaje](#)
- [Conversando con su documentación: LLM y RAG para mejorar la recuperación de información](#)
- [Aplicaciones con Modelos de Lenguaje: Crea tu propio asistente de IA generativa](#)
- [Tutorial de LangChain: Introducción a la creación de aplicaciones impulsadas por LLM](#)
- [RAG with HuggingFace + FAISS + Langchain | Retrievel Augmented Generation](#)





GRACIAS