Desafío de IA Generativa de Stori

Planteamiento del Problema

Necesito un sistema de preguntas y respuestas rápido, económico y de alta calidad sobre una base de conocimiento muy reducida (un PDF de 20 páginas). Las restricciones incluyeron:

- Volumen: Solo un único PDF de ~20 páginas (opcionalmente otros).
- Latencia: No crítica; búsquedas vectoriales y llamadas al LLM aceptables en cientos de milisegundos.
- Costo: Sin límite, pero optimizado para un gasto por solicitud económico.
- Extensibilidad: Debe poder ejecutarse tanto localmente (Docker) como de forma serverless en AWS.
  El principal desafío: respuestas precisas y fundamentadas en el contexto, sin construir una infraestructura de recuperación masiva.

Soluciones propuestas

- Generación Aumentada por Recuperación (RAG)
    - Fragmentación (Chunking): Dividir el PDF en fragmentos superpuestos de ~512 tokens para preservar la coherencia semántica.
    - Embeddings: Generar vectores de 1 536 dimensiones con text-embedding-3-small de OpenAI para búsquedas de similitud eficientes en coste.
    - Almacén Vectorial: Usar Chroma (local, basado en archivos) para indexar los embeddings; ideal para bases de datos pequeñas.
    - Cadena de Chat: Utilizar ConversationalRetrievalChain de LangChain con ChatOpenAI(gpt-4o-mini-2024-07-18) para generar respuestas, alimentando los k=3 fragmentos más relevantes.
- Alternativa: Base de Datos de Grafos + RAG
    - Para escenarios con múltiples documentos o entidades, almacenar entidades y relaciones del PDF en una base de grafos (p. ej. Amazon Neptune, Neo4j) y enriquecer la recuperación mediante recorridos relacionales antes de la búsqueda

semántica.

- Híbrido grafo+vector: combina razonamiento estructurado con contexto de texto libre.

Pila Tecnológica y Justificación

Capa	Tecnología	¿Por qué?
Ingestión de PDF	PyMuPDF	Ligero y extracción de texto precisa
Fragmentación (Chunking)	Long Chain Recursive Character Text Splitter	Mantiene límites semánticos; solapa configurable
Embeddings	OpenAI text-embedding-3-small	1 536 dimensiones; menor coste que modelos más grandes
Almacén Vectorial	Chroma	Basado en archivos; sin servicio externo, perfecto para este caso
API LLM	OpenAI gpt-4o-mini-2024-07-18	Variante “mini” de alta calidad y baja latencia
Capa de API	FastAPI	Rendimiento ASGI, documentación automática, fácil de contener
Orquestación local	Docker Compose	Entorno reproducible de desarrollo
CI/CD e Infraestructura	AWS CDK	Infraestructuras como código; despliegue en Lambda, ECS, SageMaker
Inferencia serverless	AWS Lambda + API Gateway	Autoscaling y pago por uso
Vector DB (cloud)	Amazon OpenSearch (plugin k-NN)	k-NN gestionado; escala más allá de lo local
Integración Bedrock	Amazon Bedrock	Endpoints LLM gestionados; modelos del marketplace
Ajuste fino	Amazon SageMaker JumpStart	Low-code para fine-tuning y hosting

Sugerencias de Mejoras Futuras

- ### Integración de Memoria en LangChain

  Aprovechar los módulos de memoria incorporados (por ejemplo, ConversationBufferMemory, ConversationSummaryMemory) para que la cadena persista y recupere intercambios previos, eliminando la necesidad de pasar todo el historial en cada solicitud.

- ### Servicio de Gestión de Sesiones

  Construir una capa ligera de gestión de sesiones (p. ej. con Redis o DynamoDB) que almacene el estado de la conversación en servidor. Los clientes solo enviarían un ID de sesión; el servicio cargaría y actualizaría el historial automáticamente.

- ### Híbrido Grafo | Vector

  Integrar un grafo de propiedades (Neptune/JanusGraph) para el enlace de entidades y recuperación basada en rutas.

- ### Ajuste Fino (Fine-Tuning)

  Utilizar SageMaker para afinar un modelo base más pequeño con el contenido del PDF, logrando inferencia on-premise más rápida y económica.

Métricas y Mediciones

Preguntas de Prueba

1. ¿Qué condiciones políticas y sociales propiciaron la aparición de los “Clubes Liberales” alrededor de 1900?

2. ¿Cómo influyó la publicación del periódico Regeneración en la radicalización de los hermanos Flores Magón y en la conformación del Partido Liberal Mexicano (PLM)?

3. ¿Cuál fue el papel del Congreso Liberal de 1902 en San Luis Potosí y quiénes encabezaron su organización?

Pregunta	Variante	Tiempo (s)	Tokens de entrada	Tokens de salida	Tokens totales	Costo (USD)
1	Modelo bruto	24.29	30	670	700	$0.4065
	Modelo RAG	16.87	33	149	182	$0.0919
2	Modelo bruto	13.30	43	662	705	$0.4037
	Modelo RAG	14.77	46	184	230	$0.1139
3	Modelo bruto	9.68	31	340	371	$0.2087
	Modelo RAG	7.87	34	95	129	$0.0596

- Latencia: RAG añade un pequeño sobrecoste de recuperación pero supera al modelo bruto en 2 de 3 consultas (promedio ≈ 13.17 s vs. 15.75 s).

- Eficiencia de tokens: RAG reduce el conteo de tokens de salida en ~75–80 %, disminuyendo drásticamente el coste por consulta.

- Ahorro de costos: RAG ofrece una reducción de ~75–85 % en coste, permitiendo realizar ~4× más consultas con el mismo presupuesto.

- Calidad de las respuestas: Más allá de métricas, las respuestas RAG fueron más concisas, precisas e incluyeron citas explícitas por fragmento, mejorando la confianza y trazabilidad.

Conclusión de las Pruebas

- Costo drásticamente menor: Anclando respuestas a una base relevante pequeña, RAG reduce el consumo de tokens (y por ende el gasto API) en torno al 80 %.

- Latencia comparable o mejor: Incluso con la búsqueda vectorial adicional, los tiempos de respuesta oscilan entre 7–17 s, aceptables para casos no en tiempo real.

- Mayor precisión y trazabilidad: RAG proporciona respuestas fundamentadas con citas por fragmento, evitando respuestas genéricas o parcialmente incorrectas del modelo bruto.

- Ideal para POCs a pequeña escala: Con un corpus limitado (20 páginas), la búsqueda vectorial local + un modelo “mini” de alta calidad (gpt-4o-mini-2024-07-18) ofrece un equilibrio óptimo entre rapidez, coste y precisión.