Stori’s Generative AI Challenge

Problem Statement

I need a fast, cheap, high-quality question-answering system over a very small knowledge base (a 20-page PDF). Constraints included:

- Volume: Only a single ~20-page PDF (plus optionally others)
- Latency: Not critical, vector lookups and LLM calls acceptable at hundreds of milliseconds
- Cost: Unrestricted, but optimized for economical per-request spend
- Extensibility: Able to run both locally (Docker) and serverlessly in AWS

The main challenge: accurate, context-grounded answers without building a massive retrieval infrastructure.

Proposed Solutions

Retrieval-Augmented Generation (RAG)

- Chunking: Split the PDF into ~512-token overlapping fragments to preserve semantic coherence.
- Embeddings: Generate 1536-dimensional vectors via text-embedding-3-small for cost-efficient similarity search.
- Vector Store: Use Chroma (local, file-based) to index embeddings; ideal for small datasets.
- Chat Chain: Use LangChain’s ConversationalRetrievalChain with ChatOpenAI(gpt-4o-mini-2024-07-18) for answer generation, feeding in top-k (k=3) nearest chunks.

Alternative: Graph Database + RAG

- For richer, multi-document or multi-entity scenarios, one could store PDF entities and relationships in a graph database (e.g., Amazon Neptune, neo4, etcj) and augment retrieval by traversing relationships before doing semantic search.

- Hybrid graph+vector gives structured reasoning plus free-text context.

Technology Stack & Rationale

Layer	Technology	Why?
PDF Ingestion	PyMuPDF	Lightweight, accurate text extraction
Chunking	Long Chain Recursive Character Text Splitter	Maintains semantic boundaries; easy overlap configuration
Embeddings	OpenAI text-embedding-3-small	1536-dims for compatibility; lower cost than larger models
Vector Store	Chroma	File-based, no external service needed, perfect for this small DB
LLM API	OpenAI gpt-4o-mini-2024-07-18	High-quality, low-latency “mini” variant
API Layer	FastAPI	ASGI performance, auto-docs, easy to containerize
Orchestration & Local	Docker Compose	Reproducible dev environment
CI/CD & Infra	AWS CDK	Code-first infra; deploy to Lambda, ECS, OpenSearch, SageMaker
Serverless Inference	AWS Lambda + API Gateway	Autoscaling, pay-per-use
Vector DB (cloud)	Amazon OpenSearch (k-NN plugin)	Fully managed k-NN; scales beyond local
Bedrock Integration	Amazon Bedrock	Managed LLM endpoints; pluggable marketplace models
Fine-tuning	Amazon SageMaker JumpStart	Low-code fine-tuning and hosting

Future Improvements Suggestions

LangChain Memory Integration

Leverage LangChain’s built-in memory modules (e.g. ConversationBufferMemory, ConversationSummaryMemory) so that the chain itself persists and retrieves prior exchanges. This removes the need to pass the entire message history on every request, minimizing tokens and simplifying prompt management.

Session Management Service

Build a lightweight session-management layer (e.g. backed by Redis, DynamoDB or similar) that tracks conversation state server-side. Clients would only provide a session ID; the service automatically loads, updates and persists message history behind the scenes, freeing clients from bundling prior messages with each call.

Graph | Vector Hybrid

Integrate a property graph (Neptune/JanusGraph) for entity linking and path-based retrieval.

Fine-Tuning

Use SageMaker to fine-tune a smaller foundation model on the PDF content for even faster, cheaper on-prem inference.

Metrics & Measurements

Test Questions

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?

Question	Variant	Time (s)	In Tokens	Out Tokens	Total Tokens	Cost (USD)
1	Raw Model	24.29	30	670	700	$0.4065
	RAG Model	16.87	33	149	182	$0.0919
2	Raw Model	13.30	43	662	705	$0.4037
	RAG Model	14.77	46	184	230	$0.1139
3	Raw Model	9.68	31	340	371	$0.2087
	RAG Model	7.87	34	95	129	$0.0596

Latency: RAG incurs a small retrieval overhead but still outperforms the raw model on 2 of 3 queries (average ≈ 13.17 s vs. 15.75 s).

Token Efficiency: RAG cuts output token count by ~75–80%, drastically reducing per-query cost.

Cost Savings: RAG delivers roughly 75–85% cost reduction, translating to ~4× more queries for the same budget.

Answer Quality: Beyond metrics, the RAG-powered answers were more concise, accurate, and included explicit source citations, enhancing trust and traceability.

Conclusion of Tests

1. Dramatically lower cost: By anchoring responses to a small, relevant knowledge base, RAG slashes token consumption (and thus API spend) by around 80%.

2. Comparable or better latency: Even with an extra vector lookup step, response times remain in the 7-17 s range, acceptable for non-real-time use cases.

3. Higher accuracy & traceability: RAG provides grounded answers with chunk-level citations, avoiding the generic or partially incorrect responses produced by the raw model.

4. Ideal for small-scale POCs: With a corpus is limited (in this case 20 pages), local vector search + a high-quality “mini” model (gpt-4o-mini-2024-07-18) strikes the perfect balance between speed, cost, and precision.