Retrieval-Augmented Generation (RAG) Plan: (Offline Windows RAG Plan with Local Document Corpus)

Prepared by: PARAS SAXENA

1. **Architecture** – *Local RAG pipeline design*
2. **Document Chunking:** Split PDFs/DOCX into ~800–1000 token chunks with ~100 token overlaps (avoids cutting context mid-sentence)[1]. This ensures each chunk is semantically coherent and fits the embedding model's input limits.
3. **Embeddings (SentenceTransformers):** Compare lightweight models (e.g. `all-MiniLM-L6-v2`, `all-MiniLM-L12-v2`, `all-mpnet-base-v2`). Default to **MiniLM-L6-v2** for CPU efficiency – it's ~5× faster than mpnet with only slight quality trade-off[2], ideal for responsive local querying.
4. **Vector Store:** Use **Chroma** (embedded DuckDB) to index & persist vectors locally, for simplicity and quick setup[3]. (By contrast, Faiss offers raw speed and billion-scale capability but lacks built-in persistence and DB features[4], making Chroma more convenient for a local CPU app.)
5. **Retrieval & Re-Ranking:** On query, embed the question and retrieve top **K=5** chunks by cosine similarity. Optionally apply a cross-encoder (e.g. MiniLM-based reranker) on the top 10 to re-score relevancy before final answer. This two-stage retrieval boosts recall and precision – reranking **dramatically improves** relevant results in RAG pipelines[5].
6. **Guardrails & refusal** – *Safety checks and answer validity*
7. **No-Answer Threshold:** If the best match's similarity is below *0.30* (or overall confidence is low), treat the query as unanswerable. The system returns a polite refusal (e.g. "Sorry, I couldn't find information on that") instead of risking a hallucinated answer[6]. This prevents unsupported outputs when documents don't contain the answer.
8. **Prompt Injection & PII:** Implement input validation and content filtering. Before querying, strip or ignore instructions in user input that try to manipulate the system. Filter out hateful or disallowed content and mask any detected PII (emails, phone numbers) in responses. These **guardrails** on inputs/outputs (to block malicious prompts and sensitive data leaks) are standard for a robust RAG system[7].
9. **Citation Enforcement:** Always require source attributions in the answer. The prompt to the LLM will instruct it to only answer using retrieved documents and to **cite the source of each fact**. If it cannot, it should refuse. Enforcing citations by design keeps the model grounded in the corpus[8], ensuring no answer is given without evidence from the indexed files.
10. **Evaluation & ops** – *Testing, performance, and scalability*

11. **Metrics:** Evaluate retrieval quality with **Recall@K** and **Precision@K** (ensure relevant doc chunks appear in top results) and **MRR (Mean Reciprocal Rank)** for ranking quality. For end-to-end QA, do spot-checks where the ground-truth answer is known to see if the system finds and cites the right info. Incorporate a quick **qualitative review loop** – manually verify a sample of answers for correctness and adequate sourcing[9].
12. **Offline Setup & Caching:** The first run will download the selected embedding model (~100MB) to a local cache. Use HuggingFace's cache (default `.cache/huggingface` in the user home) so subsequent runs don't re-download. Embed and index documents in **batches** to optimize CPU usage. Persist the vector index to disk (so repeated CLI sessions don't require re-indexing) and document this in the README. Provide Windows-specific tips (e.g. use OS-specific path separators, avoid very long file paths) to ensure the system runs smoothly on Windows.
13. **Scaling & Concurrency:** To handle corpus growth (10× more documents), switch to an approximate nearest neighbor index (FAISS or HNSW in Chroma) for faster searches on larger vector sets[10]. For higher query volumes (100+ concurrent users), run the pipeline as a local web service with multi-threading or async processing, and possibly preload multiple model instances for parallel embedding queries. Also consider request rate limiting and caching frequent queries to maintain responsiveness under load.

---

[1] RAG Chunking Strategies For Better Retrieval

https://customgpt.ai/rag-chunking-strategies/

[2] Pretrained Models — Sentence Transformers documentation

https://www.sbert.net/docs/sentence_transformer/pretrained_models.html

[3] [4] [10] Top 7 Open-Source Vector Databases: Faiss vs. Chroma & More

https://research.aimultiple.com/open-source-vector-databases/

[5] Rerankers and Two-Stage Retrieval | Pinecone

https://www.pinecone.io/learn/series/rag/rerankers/

[6] [8] [9] 23 RAG Pitfalls and How to Fix Them

https://www.nb-data.com/p/23-rag-pitfalls-and-how-to-fix-them

[7] Building Guardrail Around Your RAG Pipeline

https://www.nb-data.com/p/building-guardrail-around-your-rag