Mini RAG System Design Report

Overview

The objective of this project was to build a Retrieval-Augmented Generation (RAG) system capable of processing email documents, retrieving relevant information, and generating accurate answers grounded in those documents. The system was designed to be modular, efficient, and interpretable, while avoiding end-to-end RAG frameworks such as LangChain or LlamaIndex. The final architecture consists of document parsing and metadata extraction, chunking, embedding and indexing, hybrid retrieval, and grounded answer generation using a language model.

Metadata Extraction Strategy

As a first step, each email is processed by a language model to extract structured metadata, including a short summary, topics, action items, urgency level, and people mentioned. This metadata serves as a semantic abstraction of the email content. The motivation behind this step is that raw text alone may not always capture the key intent of the document efficiently, especially when the retrieval query is high-level. Metadata provides a compressed semantic representation that improves retrieval relevance and enables better filtering and interpretability. Additionally, storing metadata separately allows the retrieval system to leverage both raw content and semantic descriptors.

Chunking Strategy

I used paragraph-based chunking with a sliding window and 25% overlap. Specifically, each chunk consists of four consecutive paragraphs with one paragraph overlap between adjacent chunks. This approach was chosen because emails naturally follow paragraph-based semantic structure, and paragraph boundaries tend to align with coherent thoughts. Fixed token chunking was avoided because it can split important information mid-sentence and reduce semantic coherence. The sliding window overlap ensures that information located near chunk boundaries is not lost during retrieval. Each chunk also includes the associated metadata and email headers (subject, sender, recipient), which helps provide additional retrieval signals and improves answer grounding.

Embedding Model Selection

For embeddings, I selected the all-MiniLM-L6-v2 model from the sentence-transformers library. This model was chosen because it provides a strong balance between performance and efficiency. It produces semantically meaningful embeddings while being lightweight enough to run locally without requiring GPUs or external APIs. This makes the system cost-effective, fast, and reproducible. All embeddings were L2-normalized so that cosine similarity could be computed efficiently using inner product operations in FAISS. Local embeddings also ensure data privacy and eliminate dependency on external embedding services.

Vector Storage and Indexing

I used FAISS with the IndexFlatIP configuration to store and search embeddings. This index performs exact nearest neighbor search using inner product similarity, which corresponds to cosine similarity when vectors are normalized. The choice of a flat index was appropriate because the dataset is relatively small (approximately a few hundred chunks), so exact search is fast and avoids approximation errors. FAISS was chosen because it is highly optimized, widely used in industry and research, and easy to integrate into Python pipelines.

Hybrid Retrieval Strategy

The retrieval system uses a hybrid approach combining semantic similarity search and keyword-based search using BM25. Embedding-based retrieval captures semantic similarity between queries and documents, allowing it to retrieve relevant chunks even when exact keywords do not match. However, embedding-only retrieval can sometimes miss exact matches such as names, subjects, or specific phrases. BM25 complements this by capturing exact keyword overlap and lexical signals. The system retrieves candidates using both methods and combines their scores using a weighted sum. This hybrid approach improves both recall and precision compared to using either method alone. Additionally, metadata filtering based on sender or subject provides an additional layer of precision when queries specify structured constraints.

Prompting and Answer Generation Strategy

After retrieving relevant chunks, the system uses a retrieval-then-synthesize approach to generate answers. The retrieved chunks are formatted and passed to a language model along with the user query. The prompt explicitly instructs the model to answer using only the provided chunks and to cite chunk IDs when supporting facts are used. This helps ensure grounded responses and reduces hallucinations. The temperature was set to zero to ensure deterministic and fact-based outputs rather than creative generation. This design ensures that answers are traceable to source documents and improves reliability.

Synchronous Pipeline Design

The system uses a synchronous pipeline consisting of query embedding, retrieval, and generation. This design was chosen because it is simple, easy to implement, and sufficient for the dataset size. Since the dataset is small and embedding computation is fast, synchronous processing does not introduce significant latency. This approach also makes the system easier to debug and maintain.

Storage and Metadata Design

All chunk metadata, embeddings, and index files are stored locally. Each chunk contains its chunk ID, email file reference, subject, sender, recipient, metadata summary, and chunk text. This structured storage allows the system to trace answers back to source documents and enables future extensions such as metadata-based filtering or reranking.

Tradeoffs and Alternatives Considered

Several alternative approaches were considered during design. Fixed token chunking was rejected because it can break semantic coherence. Approximate nearest neighbor indexes such as IVF or HNSW were not used because exact search is fast enough for the dataset size. End-to-end frameworks such as LangChain were avoided to maintain full control over the pipeline and demonstrate understanding of core RAG components. Cloud embedding models were not used to avoid cost and dependency on external services.

Conclusion

The final system combines semantic metadata extraction, paragraph-based chunking with overlap, efficient local embeddings, FAISS indexing, hybrid retrieval, and grounded language model generation. These design choices ensure that the system is accurate, efficient, interpretable, and scalable. The hybrid retrieval approach improves robustness, while metadata extraction enhances semantic understanding. Overall, the system successfully demonstrates the core principles of retrieval-augmented generation while maintaining simplicity and strong performance on the email dataset.