Comprehensive RAG System Documentation

Introduction to RAG

Retrieval-Augmented Generation (RAG) is a technique that enhances large language models by providing them with relevant context retrieved from a knowledge base. This approach combines the strengths of retrieval-based and generation-based methods to produce more accurate and contextually relevant responses. The RAG pipeline typically consists of several stages: document ingestion, text extraction, chunking, embedding generation, vector storage, retrieval, and finally response generation. Each stage plays a crucial role in the overall system performance.

Document Ingestion

Document ingestion is the first stage of the RAG pipeline. It involves accepting various document formats (PDF, TXT, MD, etc.) and preparing them for processing. The ingestion system must handle file validation, format detection, and error handling for corrupted or unsupported files. Key considerations for document ingestion include: - File size limits and validation - Format detection and routing to appropriate extractors - Metadata extraction and preservation - Error handling and user feedback - Batch processing for multiple files

Text Extraction

Text extraction converts raw document bytes into structured text that can be processed by downstream components. Different document formats require different extraction strategies. PDF extraction is particularly challenging due to the variety of PDF types (searchable, scanned, mixed) and complex layouts. The extraction pipeline should: - Preserve document structure where possible - Handle multi-column layouts correctly - Extract tables and preserve their structure - Detect and handle empty or near-empty pages - Provide confidence scores for extraction quality

Chunking Strategies

Chunking divides extracted text into smaller segments suitable for embedding and retrieval. The chunking strategy significantly impacts retrieval quality. Common approaches include fixed-size chunking, semantic chunking, and structure-aware chunking that respects document boundaries. Important chunking parameters: - Chunk size (typically 512-2048 tokens) - Overlap between chunks (typically 10-20%) - Boundary detection (sentences, paragraphs, sections) - Metadata preservation per chunk

Embedding and Vector Storage

Embeddings are dense vector representations of text chunks that capture semantic meaning. These vectors are stored in a vector database that supports efficient similarity search. Popular embedding

models include OpenAI's text-embedding-3-small and various open-source alternatives. Vector storage considerations: - Embedding dimension (typically 384-1536) - Index type (HNSW, IVF, etc.) - Distance metric (cosine, euclidean, dot product) - Scalability and query performance