Contents

RAG Document Q&A System - Project Report

Executive Summary

The RAG Document Q&A System is a sophisticated application that leverages Retrieval-Augmented Generation (RAG) to provide accurate answers to questions about uploaded documents. The system integrates document processing, vector embeddings, and language model generation to create a powerful knowledge retrieval tool with summarization capabilities.

System Architecture

The system is built around three main components:

1. **Document Processing Pipeline**: Handles document uploads (PDF, DOCX), text extraction, chunking, and embedding generation
2. **Vector Database**: ChromaDB for semantic storage and retrieval of document chunks
3. **Language Model Backend**: Using Meta's Llama 3.2 1B Instruct model for generating answers and summaries

The architecture follows a modular design pattern with clear separation of concerns:

- app.py: Main application interface and orchestration
- rag.py: Core RAG functionality and conversation management
- summary.py: Document summarization capabilities

Key Features

Document Processing

- Support for PDF and DOCX files
- Intelligent text extraction with page number tracking
- Automatic chunking to fit model context windows
- Efficient embedding generation using Nomic Embed Text v2

Question Answering

- Semantic retrieval of relevant document chunks
- Conversation history tracking for context-aware responses

- Source attribution for transparent information provenance
- Performance metrics tracking (tokens, generation time, etc.)

Document Summarization

- Complete document summarization capability
- Memory-efficient processing of large documents
- ROUGE score evaluation against extractive summaries
- Summary persistence for future retrieval

Performance Characteristics

The system demonstrates robust performance characteristics:

- **Embedding Generation**: The system uses the Nomic Embed Text v2 model for semantic embeddings, with metrics tracking tokens per second.
- **Retrieval Speed**: ChromaDB vector database enables sub-second semantic search capabilities.
- **Generation Quality**: Llama 3.2 1B Instruct provides high-quality responses while maintaining reasonable inference speed on consumer hardware.
- **Memory Efficiency**: The optimized architecture loads document text only when needed and clears it from memory after processing.

Technical Implementation Details

System Architecture

The RAG Document Q&A System is built with a modular architecture that combines document processing, vector database storage, language modeling, and a user interface. The system processes PDF and DOCX documents, chunks them appropriately, generates embeddings, stores them in a vector database, and enables question-answering and summarization capabilities through a streamlined web interface.

Embedding Model

For generating document embeddings, the system utilizes the Nomic-AI embeddings model (nomic-ai/nomic-embed-text-v2-moe). On first run, the model is downloaded and cached locally in the "embed_model" directory. Subsequent runs check if the model already exists in the local cache and load it directly, avoiding redundant downloads. This embedding model transforms text chunks into high-dimensional vectors that capture semantic meaning, enabling efficient similarity search for the RAG pipeline.

Vector Database

The system employs ChromaDB as the vector database, configured as a persistent local database in the "chroma_db" directory. ChromaDB stores document embeddings, original text chunks, and associated metadata (like filenames and page numbers). The `vector_db.py` module handles the embedding generation and database interactions, tracking performance metrics such as embedding time and tokens per second during the process.

Language Model

For generating answers and summaries, the system uses Meta's Llama-3.2-1B-Instruct model. Similar to the embedding model, the LLM is downloaded on first run and cached locally in the "llm_dir" directory. The implementation leverages the Hugging Face Transformers library, which means users would need a Hugging Face token for the initial download. The model is loaded to the available device (CUDA GPU if available, otherwise CPU), which significantly impacts performance.

RAG Implementation

The RAG (Retrieval-Augmented Generation) system is implemented in the `rag.py` module. The `RAGQueryEngine` class handles:

1. Embedding user queries
2. Retrieving relevant document chunks using vector similarity
3. Creating prompts with retrieved context
4. Generating answers using the LLM
5. Tracking conversation history for context-aware responses

The system includes a `ConversationManager` that maintains conversation history, enabling the model to reference previous questions and answers for more coherent interactions.

Document Processing

The `text_processing.py` module handles document intake and processing. It includes functions to:

- Read PDF files with PyPDF2
- Process DOCX files with python-docx
- Split documents into pages
- Chunk text into suitable sizes for embedding and retrieval
- Attach metadata for source tracking

Summarization

The summarization capability in `summary.py` provides both extractive and abstractive document summarization:

- Extractive summarization selects important sentences from the document
- Abstractive summarization generates a coherent summary using the LLM
- ROUGE metrics evaluate the quality of generated summaries
- Performance metrics track token counts and generation speeds

The system can either summarize from chunked text or directly from full document text, with appropriate handling for documents that exceed context window limitations.

User Interface

The frontend is built with Streamlit, providing an intuitive web interface with:

- Document upload and processing
- Question-asking capability with relevant sources
- Document summarization functionality
- Performance metrics display
- Organized tabs for Q&A and summary viewing

The UI includes expandable sections showing source documents and performance metrics, giving users insight into how answers are generated and system performance.

Performance Tracking

Throughout the system, comprehensive performance metrics are tracked:

- Embedding generation time
- Document retrieval speed
- Token counts for input and output
- Generation time for answers and summaries
- Overall tokens per second

These metrics are displayed in the UI, helping users understand system behavior and performance characteristics across different hardware configurations.

Offline Capability

A key advantage of this implementation is its entirely local operation. After the initial model downloads, the system can run offline with all processing happening on the user's machine. This approach enhances privacy and reduces dependency on external API services, though it requires sufficient local computing resources.

Optimizations

Several key optimizations enhance system performance:

1. **On-demand Document Loading**: Documents are loaded only when needed for processing
2. **Summary Persistence**: Summaries are stored in the vector database to avoid regeneration
3. **Token Tracking**: Comprehensive metrics on token usage for performance monitoring
4. **Memory Management**: Text is cleared from memory after processing
5. **Device Adaptation**: Automatic detection and utilization of GPU acceleration when available

Future Improvements

Potential enhancements for future versions:

1. **Improved Document Processing**: Add support for more file formats (HTML, Markdown, etc.)
2. **Enhanced Retrieval**: Implement hybrid retrieval combining semantic and keyword search
3. **User Feedback Loop**: Add rating system for answers to improve retrieval quality
4. **Multi-document Queries**: Enable questions spanning multiple documents
5. **Export Capabilities**: Allow saving of conversation histories and summaries
6. **Advanced Summarization Options**: Provide customization for summary length and style

Conclusion

The RAG Document Q&A System demonstrates the powerful capabilities of combining modern embedding techniques with language models for document understanding. Its modular design, memory efficiency, and performance optimizations make it a valuable tool for document exploration and knowledge extraction.