

## Contents

<b>1. Executive Summary .....</b>	<b>2</b>
<b>2. System Architecture.....</b>	<b>2</b>
<b>3. Key Features .....</b>	<b>2</b>
Multi-format Document Processing: .....	2
Multilingual Translation:.....	3
Question Answering .....	3
Document Summarization .....	3
Performance Characteristics .....	3
<b>4. Technical Implementation Details .....</b>	<b>4</b>
System Architecture .....	4
Embedding Model.....	4
Vector Database .....	4
Language Model .....	4
RAG Implementation .....	4
Summarization .....	5
User Interface.....	5
Performance Tracking .....	5
Offline Capability .....	6
<b>5. Optimizations .....</b>	<b>6</b>
<b>6. Future Improvements .....</b>	<b>6</b>
<b>7. Conclusion.....</b>	<b>6</b>

# RAG Document Q&A System - Project Report

## 1. 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.

## 2. System Architecture

The system is built around three main components:

1. **Document Processing Modules:** Extract and preprocess text from PDFs, DOCX, Excel (.xlsx, .csv) with format-specific handlers and chunking strategies.
2. **Embedding Generation:** Uses Nomic AI's embedding model for transforming text into semantic vectors.
3. **Vector Database:** Stores embeddings and metadata using ChromaDB for efficient retrieval.
4. **Language Model Backend:** Uses Meta's Llama 3.2 1B Instruct model, with support for translation using Helsinki-NLP/opus-mt and Facebook/mbart-large-50-many-to-many-mmt.
5. **Frontend:** A Streamlit interface for uploading files, submitting queries, and viewing summaries or answers.

## 3. Key Features

### Multi-format Document Processing:

- Intelligent text extraction with page number tracking
  - Supports PDF, DOCX, XLSX, and CSV files.
  - Includes tailored chunking strategies:
    - *Auto*: Smart dynamic segmentation.
    - *Row-wise*: Ideal for CSV and Excel row-based content.
    - *Column-wise*: Handles columnar datasets.

- *Semantic*: Splits text based on sentence-level semantics.
- Automatic chunking to fit model context windows
- Efficient embedding generation using Nomic Embed Text v2

## Multilingual Translation:

- Automatically detects non-English content.
- Translates using Helsinki-NLP models; fallbacks to Facebook MBART for broader language coverage.
- Enables downstream Q&A and summarization in English or a specified target language.

## 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.

## 4. 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.

## 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.

## 5. 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

## 6. 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. **Advanced Summarization Options:** Provide customization for summary length and style

## 7. 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.