Lightweight RAG Enhanced SLM for Local Document Repositories

Syed Muhammad Irtiza(22K-4638) | Muhammad Ahmed Haque(22K-4232)

May 11, 2025

Abstract

This report presents a lightweight Retrieval-Augmented Generation (RAG) system integrated with a Small Language Model (SLM) for offline question-answering on local document repositories. The system processes text, PDF, and Word documents using minimal computational resources, leveraging a FAISS vector store and a locally run SLM. The project evaluates retrieval and generation performance, focusing on efficiency and practicality for consumer-grade hardware.

1 Project Objective

The objective of this project is to develop a lightweight Retrieval-Augmented Generation (RAG) system that uses a Small Language Model (SLM) to answer questions from local document repositories. The system operates entirely offline, targeting low-resource environments and supporting multiple document formats, including text, PDF, and Word files. It aims to provide efficient and accurate question-answering for users querying local documents without cloud-based APIs, addressing challenges like resource constraints and data privacy.

2 Datasets Used

The system processes a local document repository stored in a designated directory:

- **Text Files**: Includes files like `state_of_the_union.txt`, containing U.S. policy-related content, approximately 10,000 words.
- **PDF Documents**: Policy reports or public domain PDFs, typically 5–50 pages, sourced locally.
- **Word Documents**: `.docx` files with project notes or reports, varying in length.

Documents were split into 500-character chunks with a 50-character overlap to enable efficient retrieval. The repository operates offline, ensuring no external data dependencies.

3 Model(s) Used

The system employs the following models:

- **Embedding Model**: The `sentence-transformers/all-MiniLM-L6-v2` model from HuggingFace generates 384-dimensional embeddings. This lightweight model (80 MB) is optimized for CPU inference and chosen for its efficiency.
- **Small Language Model (SLM)**: The `TinyLlama/TinyLlama-1.1B-Chat-v1.0` model, with 1.1 billion parameters, handles text generation. It was selected for its low resource requirements and local inference capability.

The FAISS vector store manages document embeddings for similarity search, integrated via LangChain. Key settings include a chunk size of 500, chunk overlap of 50, and a maximum of 512 generated tokens. A custom prompt ensures answers are concise and context-based.

4 Results

The system was evaluated using three test questions on a local document repository:

- **Retrieval Performance**: Precision@k and Mean Reciprocal Rank (MRR) were 0.00, as relevant chunk indices were not annotated, indicating incomplete evaluation setup.
- **Generation Performance**: BERTScore F1 scores for generated answers were 0.82, 0.86, and 0.78, with an average of 0.82, suggesting reasonable semantic similarity to reference answers.
- **Efficiency**: Average query latency was 16.23 seconds across questions, with individual latencies of 3.82, 27.78, and 17.07 seconds. Memory usage averaged 3018.61 MB, with per-question usage around 3016–3022 MB.

The system successfully processed text, PDF, and Word files, with a FAISS index persisted for reuse.

5 Discussion on Results

The RAG system demonstrates the feasibility of offline question-answering on local documents using lightweight models. The `all-MiniLM-L6-v2` model enabled efficient embedding generation, while `TinyLlama-1.1B` produced reasonably relevant answers, as indicated by an average BERTScore F1 of 0.82. However, retrieval evaluation (Precision@k and MRR of 0.00) was inconclusive due to missing relevance annotations, highlighting the need for proper ground-truth setup. Latency varied significantly (3.82–27.78 seconds), likely due to CPU-based inference and document complexity, suggesting potential optimization with GPU support. Memory usage (3 GB) is viable for consumer hardware but could be reduced with model quantization. The systems ability to handle multiple file formats enhances its utility, though PDF and Word parsing may fail on complex formats. Limitations include `TinyLlama`'s occasional inaccuracies and the need for manual chunk annotation. Future work could involve annotating relevant chunks, using a larger SLM like `Phi-3-mini`, or automating document preprocessing.