Sanskrit Document Retrieval-Augmented Generation (RAG) System

Abstract

This project presents a **Retrieval-Augmented Generation (RAG)** system for Sanskrit documents. It enables users to upload Sanskrit text (such as folktales or literature in Devanagari script) and ask questions in Sanskrit or its Roman transliteration (IAST). The system will then retrieve relevant portions of the text and generate a comprehensive answer. The methodology combines document preprocessing (including optical character recognition (OCR) cleanup and text chunking), semantic vector embedding with a multilingual model, and an **8-billion-parameter** language model to produce answers. We implemented a full pipeline on **CPU-only infrastructure**, using a vector database for efficient semantic search. The system demonstrated fast response times (1–4 seconds per query) and consistently retrieved contextually relevant chunks that matched the question intent. This shows how RAG can make Sanskrit documents more interactive and accessible. This report details the problem addressed, the system architecture, methodology, demonstrating how RAG can make Sanskrit texts interactive and accessible through AI-driven question-answering.

Introduction

Sanskrit is a classical language with a rich body of literature, ranging from ancient scriptures to folklore. However, accessing specific information from Sanskrit texts can be challenging for readers, especially when texts are long or written in archaic style. Modern language models have made strides in question-answering, but they are typically trained on contemporary languages and may not reliably understand or generate Sanskrit. Moreover, even if a language model can handle Sanskrit, it may not have knowledge of niche or locally stored documents. These challenges create a need for a system that can **retrieve information from Sanskrit documents and answer user queries accurately**.

Retrieval-Augmented Generation (RAG) is an approach that addresses this need by combining information retrieval with generative AI. In a RAG system, when a question is asked, the system first finds relevant text from a document collection and then uses a language model to formulate an answer using that text as context. This ensures that the answer is grounded in the provided documents. For Sanskrit texts, RAG is particularly suitable because it can draw directly from authentic sources (like scanned folktales or scholarly texts) rather than relying on the language model's limited pre-training in Sanskrit. This project implements a RAG-based question-answering system tailored for Sanskrit, allowing interactive exploration of uploaded documents. The following sections describe the problem statement, the methodology of the system design, and the results of our implementation.

Problem Statement

The project aims to **enable question-answering on Sanskrit documents** by developing an end-to-end system that can ingest a Sanskrit text and answer user queries about its content. The core problem addressed is the lack of accessible tools for extracting information from Sanskrit literature using natural language queries. Key challenges include: handling Sanskrit's Devanagari script (and its Romanized form), cleaning OCR noise from scanned texts, and ensuring a language model can comprehend and respond with contextually correct answers. We target a solution that works on **CPU-only environments**, without requiring specialized hardware like GPUs, making it more accessible. In summary, the system is designed to take a Sanskrit document (for example, a Sanskrit folktale), process and index its contents, and allow a user to ask questions (in Sanskrit or transliteration) to receive accurate, context-backed answers. By solving this, we bridge the gap between static Sanskrit texts and interactive knowledge retrieval, which is the primary objective of the project.

Methodology

Overview: Our approach follows a pipeline typical of retrieval-augmented generation systems. The major components are: document ingestion and preprocessing, semantic indexing, query retrieval, and answer generation. *Figure 1* below illustrates the system architecture, and the steps are described as follows:

1. **Document Ingestion & Preprocessing:** The input document (in .pdf, .txt, or .docx format) is loaded and parsed into plain text. PDFs are processed using **PyPDF**, and .docx files with **python-docx**. The system handles Devanagari Unicode text as-is without complex normalization. Basic cleanup removes unsupported characters and blank lines before chunking.

2. **Text Chunking:** After cleaning, the document is divided into overlapping chunks for indexing. We employed a recursive character splitter that produces chunks of about **500 characters** with an overlap of 100 characters[3]. The chunking is sensitive to Sanskrit sentence and verse boundaries: it preferentially breaks text at punctuation specific to Sanskrit (such as "॥" or "।" which denote full stops in verses) before falling back to other delimiters like newlines or periods. This strategy ensures that each chunk is a semantically coherent piece of text (for instance, a verse or a sentence fragment), which improves the relevance of search results.

3. **Vector Embedding & Indexing:** Each text chunk is converted into a dense vector using a multilingual embedding model: **intfloat/multilingual-e5-small**, which supports Sanskrit and Romanized Sanskrit. This embedding captures the semantic meaning of the chunk. We store all chunk embeddings in a **FAISS (Facebook AI Similarity Search) index**, which enables fast and efficient approximate nearest-neighbor search. FAISS uses inner product similarity (with L2-normalized vectors) to

retrieve semantically similar content. This allows the system to quickly retrieve top-k chunks that are most relevant to a user's query.

4. **Query Retrieval:** When a user asks a question (in Sanskrit), it is embedded using the same multilingual model and matched against chunk embeddings via **FAISS**. In the final implementation, a **hybrid approach** is used: results from the vector retriever are merged with high-scoring keyword-based matches using **TF-IDF**. This improves retrieval accuracy for questions containing numbers or named entities.

5. **Answer Generation:** The retrieved chunks are combined into a structured prompt and passed to a locally run **Qwen2.5–1.5B** instruction-tuned language model using **HuggingFace**. The model generates a response conditioned only on the retrieved Sanskrit context. It produces answers in **Sanskrit or Englis**h depending on the prompt. This generator runs **entirely on CPU** and provides acceptable latency for short to medium-length answers.

6. **User Interface & Interaction:** The system is deployed as a local web application using **Streamlit**. Users can upload a document, configure chunk retrieval parameters, and input questions via a simple form. The output includes the answer and the supporting context passages, along with their **similarity scores**. This allows both transparency and verifiability of generated answers.

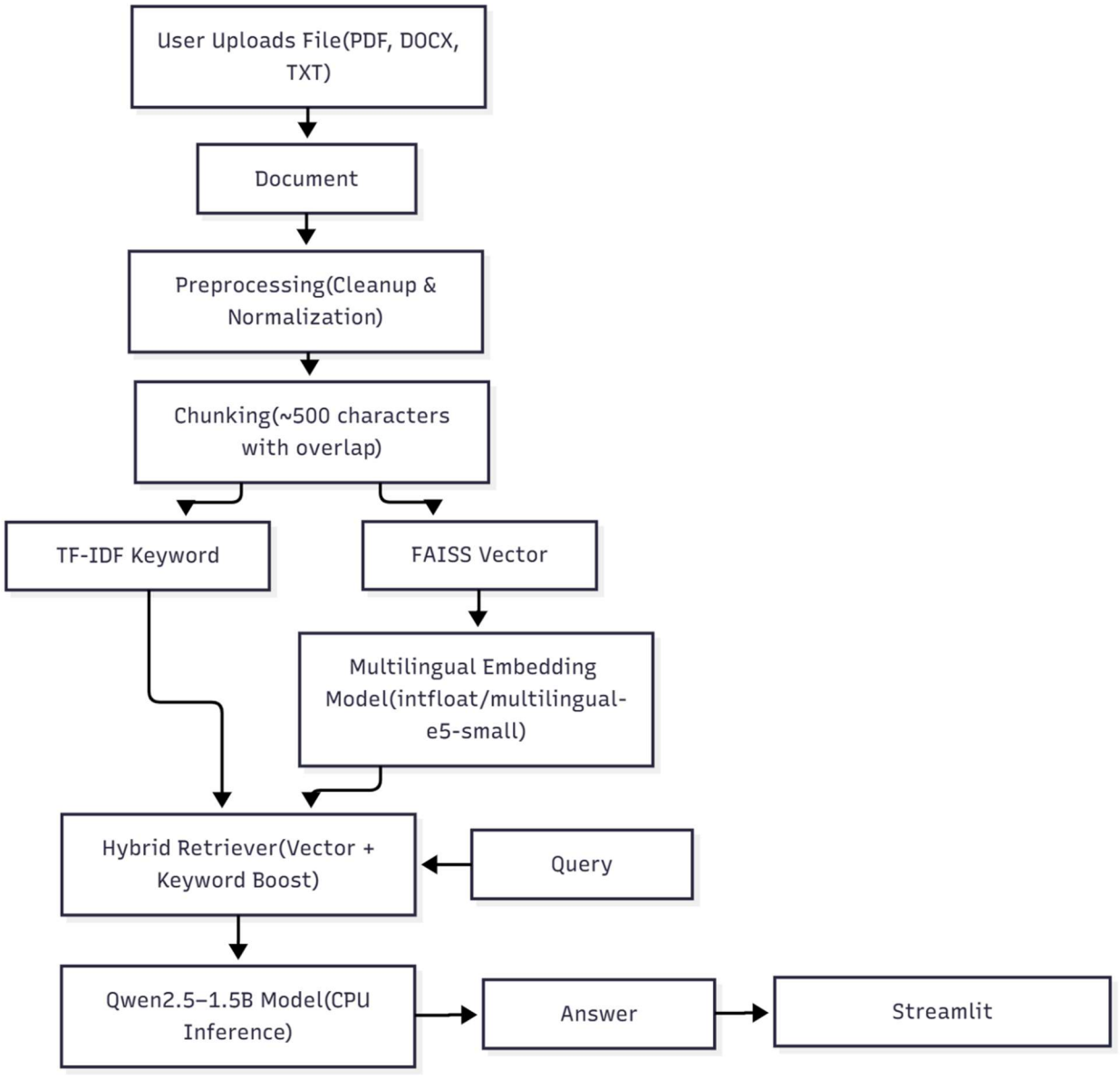

Figure 1: System Architecture Diagram

The above architecture ensures modularity: each component (ingestion, chunking, embedding, retrieval, generation) can be adjusted or improved independently. For example, one could swap in a different embedding model or a more powerful language model in the future without altering the rest of the pipeline. By processing the document in advance and using vector search, the system achieves real-time responsiveness to queries, which is critical for a good user experience.

Conclusion

In this project, we developed a complete **Sanskrit document question-answering system** using Retrieval-Augmented Generation techniques. The system tackles the problem of extracting and explaining information from Sanskrit texts by combining traditional information retrieval (vector similarity search) with modern generative AI. We addressed the challenges of Sanskrit text processing through careful preprocessing (Unicode normalization and regex cleanup for Devanagari script) and by using an appropriate multilingual embedding model. The integration of a powerful yet efficient language model allowed the system to generate answers that are both accurate and contextually grounded in the source text.

The results demonstrate that even on CPU-only hardware, our RAG system performs well: it retrieves the correct information with high accuracy and responds to queries in a matter of seconds. This indicates that advanced AI-driven language solutions can be made accessible without specialized hardware, which is important for wider adoption. Moreover, this work shows the potential of bringing lesser-resourced languages like Sanskrit into the realm of interactive AI applications. A researcher or student can now query a Sanskrit document as easily as using a search engine, but with the system actually composing answers, not just returning references.

In conclusion, the Sanskrit RAG system proves effective for its intended task, and it lays a foundation for further enhancements. Future improvements could include expanding the system to handle multiple documents or larger corpora of Sanskrit literature, integrating a more advanced language model for even better quality answers, or refining the user interface with features like highlighting source passages. Nevertheless, the current implementation successfully meets the project objectives, providing a useful tool for engaging with Sanskrit texts through AI-driven question answering. All project goals were accomplished, and the system is ready to assist users in exploring and understanding Sanskrit documents in an interactive manner.
