Technical Report: Sanskrit Retrieval-Augmented Generation (RAG) System

Project Title: Sanskrit Question Answering via Retrieval-Augmented Generation

Developer: Nikhil Pratap singh

Email: nikhilpratap5683@gmail.com

Execution Mode: CPU-Only

Date: December 28, 2025

Objective

Setting up a functional and explainable RAG system for Sanskrit documents, ensuring accurate and contextually relevant answers derived solely from the provided textual data.

1. System Architecture and Workflow

The proposed system operates on the **Retrieval-Augmented Generation (RAG)** framework. This approach ensures that generated responses are grounded in specific textual content from source documents, rather than relying on the general knowledge of the AI model. The workflow proceeds in a defined sequence:

Architecture Overview

The system's operation follows this structured sequence:

1. **Document Ingestion**

   Sanskrit documents, primarily in PDF format, are placed into the designated `data/` directory of the project.

2. **Document Loading**

   The `PyPDFDirectoryLoader` utility is employed to efficiently extract raw text content from each page of all PDFs within the `data/` directory.

3. **Preprocessing & Chunking**

   To manage the characteristic length and complexity of Sanskrit texts, documents are divided into smaller, overlapping segments (chunks). This strategy is crucial for:

   - Improving the alignment and precision of retrieval results.
   - Maintaining the continuity of meaning across segments.
   - Enhancing the relevance of retrieved information to user queries.

   The specific chunk configuration is set at:

   - **Chunk Size:**
     Approximately 600 characters
   - **Overlap:**
     Approximately 200 characters

4. **Generation of Embeddings**

   Each prepared text chunk is converted into a vector representation (embedding) using the multilingual sentence-transformers model: `sentence-transformers/distiluse-base-multilingual-cased-v2`.
   Rationale for model selection:

   - Explicit support for the Sanskrit language.
   - Optimized for lightweight operation on CPU resources.
   - Demonstrated stable performance across multiple languages.

5. **Vector Storage (ChromaDB)**

   Generated embeddings are stored and managed locally within ChromaDB, a robust vector database. Key advantages include:

   - Persistent data storage, allowing for reusability.
   - High-speed similarity search capabilities.
   - Reliable and consistent retrieval performance.

6. **User Query Interface**

   Users interact with the system by submitting their questions in Sanskrit via the terminal interface, executed through the `ask.py` script.

7. **Information Retrieval**

   The user's Sanskrit query is first converted into an embedding, which is then compared against the vectorized chunks stored in ChromaDB. The system retrieves the top-ranking, most semantically similar Sanskrit text segments.

8. **LLM Answer Generation**

   The retrieved Sanskrit text segments are fed into the Google Gemini Model. The model is explicitly instructed via a system prompt to:

   - Provide answers solely based on the provided context (retrieved text).
   - Strictly avoid hallucination or generating information not present in the source.
   - Respond with "I don’t know" if the answer cannot be found within the retrieved material.

This meticulous setup ensures that the system produces controlled, meaningful, and strictly document-grounded outputs.

2. Details of Sanskrit Documents Employed

The system's efficacy is evaluated using Sanskrit PDF documents stored within the `data/` folder.

Document Nature

The repertoire of included materials is diverse, comprising:

- Narrative Sanskrit texts
- Literary passages from classical and medieval periods
- Educational narratives focused on morals and ethics
- Proseworks representative of Classical Sanskrit literature

Rationale for Text Selection

The selection of these specific documents was driven by the following objectives:

- To rigorously assess the question-answering capabilities of the RAG system.
- To test the system's comprehension and retrieval accuracy with authentic Sanskrit script and vocabulary.
- To explore its proficiency in retrieving specific narrative elements, character details, and moral lessons.
- To validate the meaningfulness and contextual accuracy of extracted information.

Representative texts include classic stories about King Bhoja and traditional fables featuring various animal characters, among others. This selection enables a comprehensive assessment of the system's performance across both comprehension-based and fact-based Sanskrit queries.

3. Preprocessing Pipeline for Sanskrit Documents

A structured and systematic preprocessing pipeline is employed to ensure the smooth and efficient operation of the RAG system when handling Sanskrit documents.

1. **Step 1 — PDF Ingestion**

   The `PyPDFDirectoryLoader` systematically reads all PDF files provided in the `data/` directory.

2. **Step 2 — Text Extraction**

   Textual content is accurately extracted from each page of the ingested PDF documents.

3. **Step 3 — Chunk Generation**

   The `RecursiveCharacterTextSplitter` is utilized to divide extensive texts into manageable, smaller chunks. Crucially, these chunks are designed to overlap, which is vital for maintaining semantic continuity and improving the precision of contextual matching during retrieval.

4. **Step 4 - Metadata Attribution**

   Each generated chunk is systematically annotated with essential metadata to facilitate traceability and management:
   - A unique identifier for each chunk.
   - The source document from which the chunk was derived.
   - Specific page number information.

This systematic organization ensures robust data management and aids in debugging and auditing.

4. Retrieval and Generation Mechanisms

Retrieval Process

- A user submits a question formulated in Sanskrit.
- This query is computationally transformed into a vector embedding.
- The query embedding is then systematically compared against the embeddings of all stored document chunks using a similarity search algorithm.
- The system identifies and retrieves the top-most relevant Sanskrit text segments that best match the query's semantic intent.

Generation Process

The content retrieved from the documents is then passed to the Gemini Large Language Model. A precisely crafted system prompt guides the LLM's output:

System Prompt: "No external assumptions. Stick only to the information provided in Sanskrit. If unavailable, respond with 'I don’t know.'"

This directive is critical for enforcing reliable, document-based responses and substantially reducing the occurrence of hallucinations, thereby ensuring the trustworthiness of the system's answers.

5. Technologies and Tools Used (detailed)

- **Programming Language**
  Python 3.11+
- **Vector Database**
  ChromaDB: Chosen for its local deployment capability, persistent storage, and efficient similarity search, enabling fast retrieval performance.
- **Embedding and NLP Processing**
  - **Sentence Transformers:** Model `sentence-transformers/distiluse-base-multilingual-cased-v2` is used for converting Sanskrit text into vector embeddings.
  - Supports Sanskrit text input and embedding generation.
  - Designed for CPU-compatible operations.
  - Utilized via `langchain-community` for document loading capabilities.
  - `langchain-text-splitters` library is employed for effective chunking of long Sanskrit texts.
  - `pypdf` library handles the task of PDF text extraction.
- **Environment and Configuration**
  - `python-dotenv`: Used for secure handling and loading of sensitive API keys, such as `GEMINI_API_KEY`.
  - **CPU Enforcement:** GPU usage is explicitly disabled by setting `os.environ["CUDA_VISIBLE_DEVICES"] = ""`, guaranteeing that the system runs purely on CPU resources.
- **Large Language Model (LLM)**
  - **Google Gemini API:** Leveraged through the `google-genai` library for the final stage of answer generation.
  - The API is configured to prioritize and emphasize strict adherence to the provided document context.

6. Performance Observations

Latency

- **Database Creation:** Initial setup for database creation incurs some time consumption due to the computationally intensive embedding generation process.
- **Retrieval Speed:** Retrieval operations are notably fast, attributed to the efficient, local indexing and search capabilities of ChromaDB.
- **LLM Response Time:** The latency for LLM-generated responses is primarily dependent on external factors such as network conditions and the response time of the Gemini API itself.

Accuracy and Relevance

- The system consistently provides correct answers that are directly supported by the textual material present in the source documents.
- The implementation of an overlapping chunk design significantly contributes to increased retrieval accuracy and contextual understanding.
- A key feature is the system's robust handling of absent information: if a query cannot be answered from the provided text, it systematically and reliably responds with "I don’t know," preventing any fabrication.

Resource Utilization

- The system is entirely CPU-based, eliminating any dependency on or requirement for GPU hardware.
- Memory usage scales proportionally with the total number of text chunks processed.
- The system demonstrates stable performance on standard laptop hardware configurations.

Conclusion

- **The developed system successfully establishes a functional Sanskrit Retrieval-Augmented Generation (RAG) framework, meeting all stipulated assignment requirements.**
- **It provides a complete RAG architecture designed for Sanskrit content.**
- **Demonstrates proper handling and processing of Sanskrit documents.**
- **Features an efficient and accurate retrieval pipeline.**
- **Ensures controlled, document-grounded, and reliable answer generation.**
- **The implementation is strictly CPU-oriented for broad accessibility.**
- **Adheres to reproducible practices for development and deployment.**

Deliverables

- Fully runnable source code package.
- Configured local ChromaDB database instance.
- A functional CPU-only embedding and retrieval pipeline.
- A comprehensive README file detailing execution steps.
- This Technical Report (document).