## Name: Rainee Jain

## ImmverseAI Assignment

**Project Title:** Sanskrit Document Retrieval-Augmented Generation (RAG) System

### Introduction:

Sanskrit is a classical language with a rich corpus of philosophical, literary, and instructional texts. However, due to its complex grammar, compound words (sandhi), and limited modern NLP resources, automated question answering over Sanskrit documents remains challenging.

Large Language Models (LLMs) alone often hallucinate facts or fail to ground answers in source texts. To address this, Retrieval-Augmented Generation (RAG) combines information retrieval with generative models, ensuring that generated answers are grounded in relevant document content.

### 1.Objective:

The objective of this project is to design and implement a CPU-only Retrieval-Augmented Generation (RAG) system capable of:

- Processing Sanskrit documents
- Retrieving relevant textual context
- Generating coherent, context-grounded answers in response to user queries

The system is designed to operate entirely on CPU, ensuring accessibility and compliance with deployment constraints.

## 2.System Architecture:

The system follows a standard RAG pipeline with clear separation between retrieval and generation components:

User Query (Sanskrit / Transliteration)

↓

Query Encoding

↓

Vector Retriever (FAISS)

↓

Relevant Sanskrit Chunks

↓

Prompt Construction

↓

CPU-Based Language Model

↓

Generated Answer

## Modular Design:

The system is divided into the following independent modules:

1. Document Loader & Preprocessor
2. Text Chunker
3. Vector Retriever
4. Language Model Generator
5. Query Interface

This modular design improves maintainability, scalability, and clarity.

## 3.Sanskrit Document Description:

### 3.1 Document Format

The source documents are Sanskrit texts provided in Unicode format, stored as .docx files. The content includes narrative prose and didactic stories written in Devanagari script.

### 3.2 Domain

The documents belong to the domain of classical Sanskrit literature, including moral stories and scholarly anecdotes.

## 4. Preprocessing Pipeline

### 4.1 Text Extraction

Since LangChain does not natively support .docx files, the python-docx library was used to extract paragraph-level text while preserving Unicode characters.

### 4.2 Unicode Normalization

To ensure consistency in Sanskrit character representation, Unicode Normalization Form C (NFC) was applied:

- Prevents character mismatch during embedding
- Ensures reliable semantic similarity computation

### 4.3 Text Chunking

Due to the long length of Sanskrit prose, documents were divided into overlapping chunks using a recursive character-based strategy:

- Chunk size: 400 characters
- Overlap: 80 characters

This ensures:

- Semantic completeness
- Reduced information loss across chunk boundaries

## 5. Retrieval Mechanism

### 5.1 Embedding Model

A multilingual sentence embedding model was used:

**Model:**
sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2

**Reasons for selection:**

- Supports Indic languages
- Lightweight and CPU-efficient
- Good semantic retrieval performance

### 5.2 Vector Store

The embeddings were indexed using FAISS (Facebook AI Similarity Search) in CPU mode.

**Advantages:**

- Fast similarity search
- Low memory overhead
- Scalable for large corpora

### 5.3 Retrieval Strategy

For each user query:

- The query is embedded
- Top-k (k=4) most relevant document chunks are retrieved using cosine similarity

## 6. Generation Mechanism

### 6.1 Language Model Selection

**Model Used:**
google/mt5-small

**Reasons for selection:**

- Trained on multilingual corpora including Sanskrit
- Better Indic language generation than FLAN-T5
- Efficient CPU inference

### 6.2 Prompt Engineering

A structured prompt was used to guide the model:

- Enforces answer grounding in retrieved context
- Encourages Sanskrit output
- Reduces hallucination

**Prompt Structure:**

- System role definition (Sanskrit scholar)
- Retrieved context
- User question
- Answer instruction

## 7. End-to-End Query Flow

1. User enters a Sanskrit query
2. Query is passed to the retriever
3. Relevant chunks are fetched from FAISS
4. Retrieved context is injected into the prompt
5. The LLM generates a grounded response
6. Final answer is returned to the user

**8. CPU Optimization Strategies**

To ensure efficient CPU-only execution:

- Small multilingual embedding model
- Lightweight FAISS-CPU index
- mt5-small instead of large LLMs
- Limited token generation (max_new_tokens)
- No GPU dependencies
- Batch size = 1

**9. Performance Observations**

**9.1 Latency**

- Retrieval time: ~40–70 ms
- Generation time: ~1.5–2.5 seconds per query

**9.2 Resource Usage**

- RAM usage: ~2–3 GB
- CPU utilization: Moderate, stable

**9.3 Accuracy**

- High factual accuracy for document-based queries
- Minimal hallucination due to retrieval grounding

**10. Future Enhancements**

- Hybrid BM25 + vector retrieval
- Sandhi splitting for better semantic matching
- Transliteration support (IAST ↔ Devanagari)
- Web-based UI using Streamlit
- Fine-tuned Sanskrit-specific LLM

## 11. Conclusion

- This project successfully demonstrates a CPU-based Retrieval-Augmented Generation system for Sanskrit documents. By combining semantic retrieval with multilingual language generation, the system provides accurate, grounded answers while remaining computationally efficient.
- The modular architecture, CPU optimization, and end-to-end functionality make the system suitable for academic research, digital humanities, and Sanskrit knowledge preservation applications.