Technical Report: CPU-based Sanskrit RAG System

Author: Krish Kumar **Date:** December 28, 2025

1. Objective

The primary objective of this project was to design and implement a **Retrieval-Augmented Generation (RAG)** system tailored for processing and answering queries based on **Sanskrit documents**. A critical constraint of the project was ensuring the system operates entirely on **CPU-based inference**, prohibiting the use of GPUs. The system aims to ingest Sanskrit text, index it for efficient retrieval, and generate coherent responses using a lightweight Large Language Model (LLM).

2. Dataset Description

The system ingested a collection of Sanskrit narratives and moral stories provided in Rag-docs.docx. The corpus consists of approximately 9,000 characters and includes several key texts:

- **Mūrkha-bhṛtyasya (The Foolish Servant):** A satire on a servant named Shankhanada who follows instructions too literally, leading to disasters with sugar, a puppy, and milk.
- **Caturasya Kālidāsasya (Clever Kalidasa):** A narrative about King Bhoj offering a prize for a new poem, which scholars cheat to claim, until Kalidasa tricks them using a trap poem about debt.
- **Vṛddhāyāḥ Cāturyam (The Old Woman's Cleverness):** A fable where villagers fear a "Ghantakarna Rakshasa" (Bell-Eared Demon), which an old woman discovers is simply monkeys playing with a stolen bell.
- **Devabhakta (The Devotee):** A moral story about a devotee whose wagon gets stuck in the mud, learning that God helps those who exert effort (udyam).

3. System Architecture & Research

3.1 Overview

The architecture follows a modular RAG pipeline designed for CPU efficiency. It consists of three core components:

1. **Document Ingestion & Preprocessing**
2. **Retriever (Vector Store)**
3. **Generator (LLM)**

3.2 Component Selection

- **Document Loader:** python-docx was selected to extract raw text while preserving paragraph structures, essential for maintaining context in Sanskrit narratives.
- **Embedding Model:** sentence-transformers/paraphrase-multilingual-mpnet-base-v2. This model was chosen for its strong multilingual support (mapping semantically similar concepts across languages) and its balance of accuracy vs. inference speed on CPU.
- **Vector Database:** FAISS (CPU). Facebook AI Similarity Search is the industry standard for dense retrieval. The CPU version was used to strictly adhere to hardware constraints.
- **Large Language Model:** TinyLlama/TinyLlama-1.1B-Chat-v1.0. A 1.1 billion parameter model explicitly designed for compute-constrained environments. It was loaded with torch.float32 to optimize for system RAM.

4. Preprocessing Pipeline

To optimize the Sanskrit text for RAG, the following pipeline was implemented:

1. **Normalization:** Regex cleaning (re.sub) was applied to remove irregular newlines and excessive whitespace common in extracted .docx text.
2. **Chunking:** The text was segmented into chunks of **500 characters** with a **50-character overlap**.
3. **Rationale:** The 500-character window captures full sentences or *shlokas* to maintain semantic completeness. The overlap ensures context is not lost if a query refers to information located at the boundary of two chunks.

5. Performance Evaluation

The system was tested on a standard CPU environment with 7 distinct queries ranging from direct extraction to summarization.

5.1 Latency Observations

Inference times were significant due to the CPU-only constraint for the 1.1B parameter model:

- **Fastest Response:** ~59.92 seconds (Query: *Bhojarajasya sabhayam...*)
- **Slowest Response:** ~150.48 seconds (Query: *Murkhabhrtyasya katham...*)
- **Average Latency:** ~109 seconds per query.

5.2 Retrieval & Generation Quality

- **Retrieval:** The FAISS retriever successfully identified relevant chunks for context-specific queries (e.g., retrieving the "Ghantakarna" story when asked about the demon).

- **Generation Issues:** The TinyLlama model, while CPU-efficient, demonstrated limited proficiency in generating Devanagari Sanskrit. It frequently defaulted to transliterated text or mixed English headers (e.g., "Question/Answer" tags) into the output.

6. Conclusion

The project successfully demonstrated a functional RAG pipeline on CPU hardware. While the retrieval mechanism proved robust for Sanskrit texts, the generation latency (~1.5 minutes/query) and linguistic limitations of the 1.1B model suggest that for production use, a specialized Indic-LLM or further quantization (e.g., GGUF format) would be required to improve user experience.