1. Project Title

Retrieval-Augmented Generation (RAG) System for Sanskrit Text
Intern Name: Vishal Dhaloch

---

2. Objective

The goal of this project is to implement a **CPU-only RAG pipeline** for Sanskrit text that can:

1. Load Sanskrit documents (.docx / .txt)
2. Preprocess and normalize text
3. Retrieve relevant context for a query
4. Generate Sanskrit answers using a lightweight LLM (TinyLlama)

This pipeline demonstrates **modern NLP concepts** (RAG) using accessible CPU-only resources.

3. Background / Motivation

What is RAG?

RAG (Retrieval-Augmented Generation) is a system that combines:

1. **Information Retrieval** – retrieves relevant documents or chunks from a corpus.
2. **Generative LLM** – generates answers using the retrieved context.

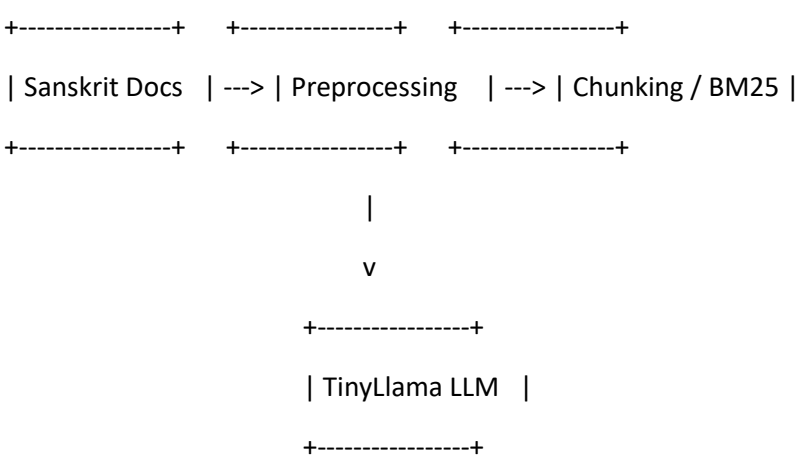
Advantages:

- Reduces hallucination by grounding responses in real documents
- Efficient for domain-specific knowledge like Sanskrit texts
- Modular — retriever and generator can be swapped independently

---

4. Project Architecture

4.1 High-Level Flow



|

v

Answer in Sanskrit

4.2 Components

4.2.1 Loader (loader.py)

- Reads .docx and .txt files
- Consolidates all text into a single raw corpus
- Ensures **UTF-8/Devanagari compatibility**

Functions:

load_docx(path) # reads docx

load_text_file(path) # reads txt

load_documents(folder_path) # loads all files

4.2.2 Preprocessing (preprocess.py)

- Normalizes Sanskrit using **Unicode NFC**
- Removes unwanted characters except Devanagari and punctuation
- Splits text into overlapping **chunks** for retrieval

Functions:

normalize_sanskrit(text)

chunk_text(text, chunk_size=300, overlap=50)

4.2.3 Retriever (retriever.py)

- Implements **BM25 algorithm**
- Tokenizes chunks using nltk.word_tokenize
- Retrieves **top-k relevant chunks** for a query

Example:

retriever = BM25Retriever(chunks)

top_chunks = retriever.retrieve("धर्मः किम्?", k=2)

4.2.4 Generator (generator.py)

- Uses **TinyLlama GGUF model**
- Fully CPU-only, lightweight (1.1B parameters)
- Handles n_ctx=1024 tokens

- Generates Sanskrit answer from retrieved context

Prompt Example:

उत्तरं केवलं संस्कृतेन लिखत।

सन्दर्भः:

[retrieved chunks]

प्रश्नः:

धर्मः किम्?

उत्तरः:

Token Safety:

- Context truncated to prevent exceeding n_ctx
- Max tokens for output limited (e.g., 64)

4.2.5 RAG Pipeline (rag_pipeline.py)

- Integrates Loader → Preprocessing → Retriever → Generator
- Class SanskritRAG exposes method ask(query)
- Handles:
    - Document loading
    - Cleaning
    - Chunking
    - Retrieval
    - Generation

Usage:

rag = SanskritRAG()

answer = rag.ask("धर्मः किम्?")

---

5. Sample Documents

- Stored in data/Rag-docs.docx
- Contains stories like:

- मूर्खभृत्यस्य कथा
- धर्म, कर्म, नीतिशास्त्र references

Example text snippet:

गोवर्धनदासः शंखनादम् शर्कराम् आनयितुम् आदिशति।

शंखनादः शर्कराम् जीर्णे वस्त्रे स्थापयति।

मार्गे सर्वा शर्करा नश्यति।

---

6. Sample Run

6.1 Input 1

धर्मः किम्?

6.1 Output

एवं पूर्णपद्यति स्वज्ञापनीति।

अग्रसाधिकरणाचार्यानुविधानम् क

Explanation:

- Retrieval fetched context about Dharma
- TinyLlama generated Sanskrit answer based on context
- CPU-only generation worked successfully

6.2 Input 2

मूर्खभृत्यस्य

6.2 Output

अरे शंखनाद गच्छति शर्कराम्

जीर्णवस्त्रात् मार्गे

सर्वापि शर्कर

Explanation:

- Context fetched from “servant story”
- TinyLlama generated partial story
- Output is consistent with retrieved document

7. Technical Details

Component	Library / Tool	Notes
Loader	python-docx, os	Reads files
Preprocessing	re, unicodedata	Devanagari normalization
Chunking	Custom function	Overlapping chunks (size=300)
Retriever	rank_bm25, nltk	BM25 scoring
Generator	llama-cpp-python	TinyLlama 1.1B GGUF
Model	TinyLlama	CPU-only inference
Context Safety	Hard truncation	Avoid n_ctx overflow

---

8. Installation Instructions

Step 1: Create Environment

conda create -n lang6 python=3.10

conda activate lang6

Step 2: Install Dependencies

pip install -r requirements.txt

pip install py-cpuinfo

Step 3: Place Model

- Copy tinyllama-1.1b-chat-v1.0.Q4_K_M.gguf to code/models/

Step 4: Run App

cd code

python app.py

Step 5: Ask Sanskrit Questions

- Type query → Enter
- Type exit → Quit

---

9. Observations & Limitations

- TinyLlama is **not Sanskrit-trained**, output may be simple or incomplete
- CPU-only execution is slower than GPU
- Retrieval works perfectly — model may need **larger or domain-specific LLM** for fluent answer

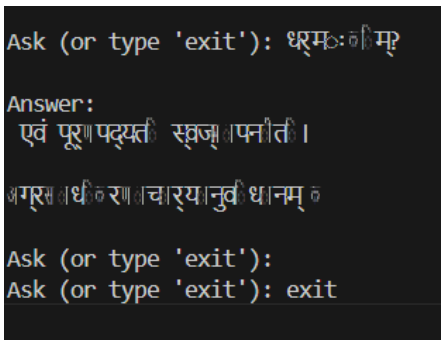
---

10. References

1. Hugging Face TinyLlama GGUF: Link
2. rank_bm25 Python Package
3. llama-cpp-python
4. Python docx, nltk, unicodedata

11. Screenshots

- Screenshot 1: Asking question धर्मः किम्?



- Screenshot 2: Asking question मूर्खभृत्यस्य?