RAG SANSKRIT PROJECT

FINAL PROJECT REPORT

Name: Rohan Soni

Project Title: Retrieval-Augmented Generation System for Sanskrit Text Understanding

Organization: Immverse.AI

Report Type: Technical Implementation

Report Date: November 2025

Table of Contents

Introduction

Natural Language Processing (NLP) has advanced significantly in modern languages, but classical languages like **Sanskrit** still face limitations due to scarce training data and less modern tooling. To bridge this gap, Retrieval-Augmented Generation (RAG) provides a hybrid approach that combines:

- **Information Retrieval (IR)** → retrieves relevant text
- **Text Generation (LLM)** → generates answers based on retrieved chunks

This project aims to design a **RAG-based solution for Sanskrit text**, enabling accurate question-answering from classical documents.

Objective

The main objectives include:

- Build a complete RAG pipeline
- Preprocess Sanskrit text from DOCX files
- Create meaningful embeddings using Sentence Transformers
- Store embeddings in a vector database (ChromaDB)
- Retrieve top-K relevant text chunks
- Generate answers using **Ollama Llama 3**
- Deliver a functional codebase with report & documentation

Problem Statement

Sanskrit texts are long, complex, and often lack structured datasets.
The project solves the challenge:

How to query and retrieve accurate Sanskrit information from unstructured documents using a modern LLM-based RAG pipeline?

System Architecture

Flow:

1. Load DOCX Sanskrit document
2. Preprocess + clean text
3. Divide into chunks
4. Generate embeddings
5. Store in ChromaDB
6. Retrieve relevant chunks
7. Send prompt to Llama via Ollama
8. Display answer

Methodology

Dataset Collection:

- Input file: Rag-docs-fixed.docx
- Contains: Sanskrit stories, prose, and classical narrative text
- Format: UTF-8 (Unicode Devanāgarī)

Preprocessing:

Steps:

- Remove extra spaces & blank lines
- Normalize Unicode characters
- Fix DOCX structure
- Convert to plain text
- Sentence-level cleanup
- Example
- Original: मूर्खभृत्यस्य
- Cleaned: मूर्खभृत्यस्य

Text Chunking:

- Chunk size used:
- 400–500 characters per chunk
- 50 characters overlap

Embedding Model:

- Model used: **paraphrase-multilingual-MiniLM-L12-v2**
- Reasons for choosing:
- Supports Sanskrit (Indic languages)
- Lightweight (471MB)
- Good semantic matching
- Fast inference

Vector Database – ChromaDB

- Backend: DuckDB + Parquet
- Persistent Local Storage
- Stores embedding vectors
- Enables similarity search (cosine similarity)

Retriever Module

- Accepts user query
- Converts to embedding
- Performs vector search
- Returns top-k relevant Sanskrit chunks
- k = 3
- distance_metric = cosine

Generator (LLM – Ollama / Llama 3)

Model used:

- ollama run llama3

Why chosen?

- Free & offline
- Very strong contextual reasoning
- Works well with RAG prompts
- Prompt template used:
- You are an expert in Sanskrit literature.

Code Structure

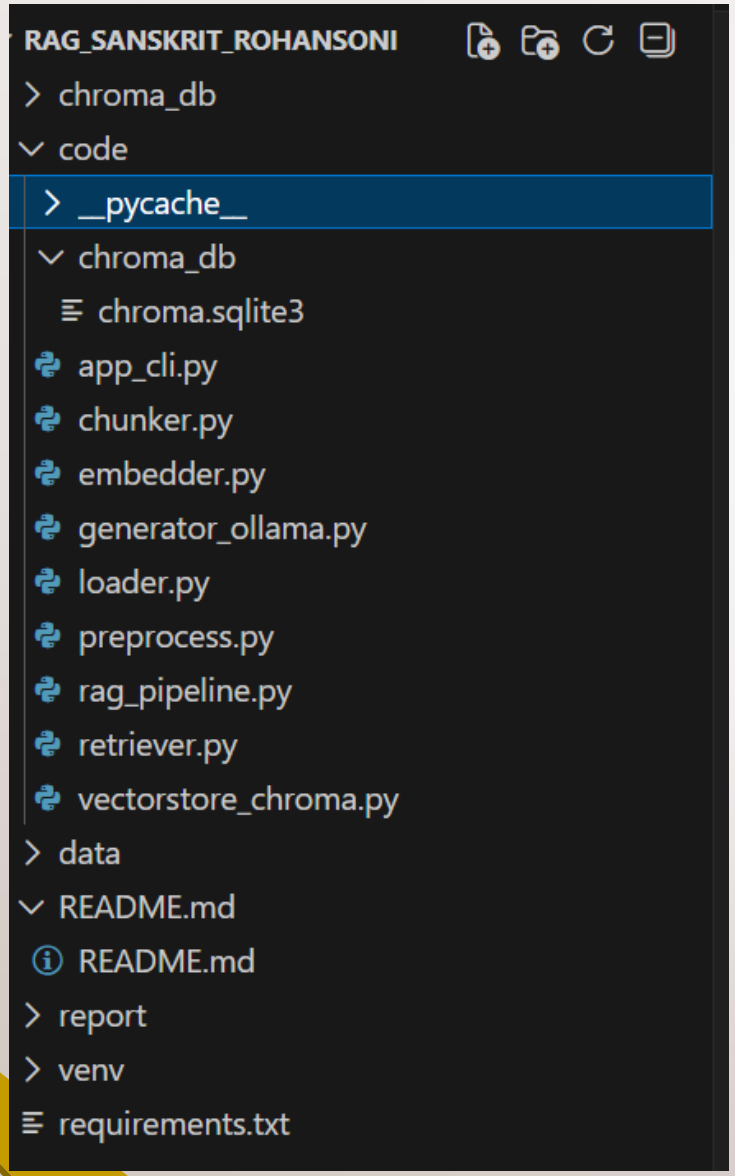

Figure 1: Project folder structure in VS Code

Implementation Details

loader.py

Load DOCX text safely

Extract paragraphs

Warn if file has no readable text

Figure 2: loader.py implementation screenshot

code > loader.py > load_docx_text

```python
import os

def load_docx_text(path):
    print("DEBUG PATH RECEIVED BY loader.py:", repr(path))
    print("File exists:", os.path.exists(path))

    text = docx2txt.process(path)

    if not text.strip():
        print("WARNING: DOCX has no extractable text. Try saving a clean version or remove tables.")

    return text.strip()

if __name__ == "__main__":
    path = r"D:\RAG_Sanskrit_RohanSoni\data\Rag-docs-fixed.docx"
    print(load_docx_text(path))
```

chunker.py

Split into overlapping chunks

code > chunker.py > chunk_text_by_words

```python
# chunker.py
def chunk_text_by_words(text, chunk_size=300, overlap=50):
    words = text.split()
    chunks = []
    for i in range(0, len(words), chunk_size - overlap):
        chunk = " ".join(words[i:i+chunk_size])
        if chunk.strip():
            chunks.append(chunk)
    return chunks
```

Figure 3: chunker.py screenshot

preprocess.py

code > preprocess.py > basic_cleanup

```python
# preprocess.py
import re

def normalize_whitespace(text):
    return " ".join(text.split())

def remove_control_chars(text):
    return re.sub(r'[\x00-\x1F\x7F]', ' ', text)

def basic_cleanup(text):
    text = remove_control_chars(text)
    text = normalize_whitespace(text)
    return text
```

Figure 4: preprocess.py screenshot

embedder.py

Load transformer model

Generate embeddings

code > embedder.py > ...

```python
# embedder.py
import os
os.environ["USE_TF"] = "0"
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

from sentence_transformers import SentenceTransformer
import numpy as np

MODEL_NAME = "paraphrase-multilingual-MiniLM-L12-v2"

def load_embedder():
    model = SentenceTransformer(MODEL_NAME)
    return model

def embed_texts(model, texts, batch_size=32):
    embeddings = model.encode(texts, show_progress_bar=True, batch_size=batch_size)
    return np.array(embeddings)
```

Figure 5: embedder.py screenshot

vectorstore_chroma.py
Create collection
Store embeddings

code > vectorstore_chroma.py > ...

```python
# vectorstore_chroma.py

import chromadb
from chromadb.utils import embedding_functions

# Create persistent client
client = chromadb.PersistentClient(path="./chroma_db")

def create_collection(name="sanskrit_rag"):
    if name in [c.name for c in client.list_collections()]:
        return client.get_collection(name)
    return client.create_collection(name)

def add_documents(collection, ids, documents, embeddings):
    collection.add(
        ids=ids,
        documents=documents,
        embeddings=embeddings
    )

def get_collection(name="sanskrit_rag"):
    return client.get_collection(name)
```

Figure 6: vectorstore_chroma.py screenshot

retriever.py

Perform similarity search

Figure 7: retriever.py screenshot

code > retriever.py > ...

```python
# retriever.py

def retrieve_top_k(collection, query, k=3):
    """
    Retrieve top-k similar chunks using ChromaDB collection.
    """
    results = collection.query(
        query_texts=[query],
        n_results=k
    )

    # Chroma returns dict: ids, documents, distances
    ids = results["ids"][0]
    docs = results["documents"][0]
    scores = results["distances"][0]

    return list(zip(ids, docs, scores))
```

generator_ollama.py

Send HTTP POST request to Ollama

code > generator_ollama.py > ...

```python
# generator_ollama.py

import requests
import json

OLLAMA_URL = "http://localhost:11434/api/generate"
MODEL = "llama3"   # Make sure you have pulled this using: ollama pull llama3

def generate_with_ollama(prompt, stream=False):
    payload = {
        "model": MODEL,
        "prompt": prompt,
        "stream": stream
    }

    response = requests.post(OLLAMA_URL, json=payload)

    # If streaming=FALSE → the result is JSON with "response" key
    data = response.json()

    # Return whichever key exists: "response" or whole JSON
    return data.get("response", data)
```

Figure 8: generator_ollama.py screenshot

rag_pipeline.py

Orchestrates entire pipeline

Builds index and answers queries

code > rag_pipeline.py > build_index

```python
# rag_pipeline.py
from loader import load_docx_text
from preprocess import basic_cleanup
from chunker import chunk_text_by_words
from embedder import load_embedder, embed_texts
from vectorstore_chroma import create_collection, add_documents
from retriever import retrieve_top_k
from generator_ollama import generate_with_ollama
import uuid

# CORRECT PATH (choose ONE)
DATA_PATH = r"D:\RAG_Sanskrit_RohanSoni\data\Rag-docs-fixed.docx"

# OR
# DATA_PATH = "../data/Rag-docs.docx"

def build_index():
    text = load_docx_text(DATA_PATH)
    text = basic_cleanup(text)
    chunks = chunk_text_by_words(text, chunk_size=300, overlap=50)
    embed_model = load_embedder()
    embeddings = embed_texts(embed_model, chunks, batch_size=16)
    coll = create_collection("sanskrit_rag")
    ids = [str(uuid.uuid4()) for _ in chunks]
    add_documents(coll, ids, chunks, embeddings.tolist())
    return coll

def answer_query(query, collection, k=3):
    top = retrieve_top_k(collection, query, k=k)
    context = "\n\n".join([doc for (_id, doc, _score) in top])
    prompt = f"""You are a helpful assistant. Use the context below (Sanskrit) to answer the user query in Sanskrit if user asked in Sanskrit, otherwise in English.\n\nContext:\n{context}\n\nUser question: {q
    resp = generate_with_ollama(prompt, stream=False)
    return resp

if __name__ == "__main__":
    coll = build_index()
    print("Index built. Ready to query.")
    q = input("Enter query: ")
    print(answer_query(q, coll))
```

Figure 9: rag_pipeline.py screenshot

Results

Sample Query

User asked the system:

“मूर्खभृत्यस्य कथायाः सारं कथय।”

(“Explain the summary of the story Mūrkhabhṛtyasya.”)

Retrieved Chunk Example

"अरे शंखनाद, गच्छ आपणम्, शर्कराम् आनय" इति स्वभृत्यम् गोवर्धनदासः आदिशति । शंखनादः जीर्णवस्त्रेण शर्कराम् आनयति, मार्गे सर्वत्र शर्करा पतति। गोवर्धनदासः क्रुद्धः वदति — "मूढ! शर्करा दृढायां सन्चिकायाम् निक्षिप्य आनीयते" ।

Generated Answer

मूर्खभृत्यस्य कथा मूर्खताजन्यदोषपरिणामं दर्शयति। शंखनादः नाम भृत्यः स्वामिनः आज्ञां सम्यक् अवगृह्य कार्यं कुर्वन् सदैव विपत्तिमेव जनयति। शर्करायाः आनयने सः जीर्णवस्त्रेण तत् वहति— मार्गे सर्वं पतति। श्वानशावकं सन्चिकायाम् स्थापयन् तस्य प्राणनाशं करोति। दुग्धं दोरकेण बध्नन् सर्वं दुग्धं मार्गे एव क्षरति। अन्ते स्वामिनः "कृष्णं भवतु ते मुखम्" इति वाक्यं शृण्वन् सः कज्जलेन मुखं कृष्णं करोति। एवं कथा वदति— **मूर्खभृत्यस्य संगतिः सर्वं कार्यं विनाशयति**।

Challenges and Solution

Challenge	Solution
DOCX text extraction failed	Converted to fixed DOCX version
Embeddings empty	Verified chunks, increased chunk size
Chroma migration error	Updated to latest API
Llama response irrelevant	Improved prompt template

Conclusion

- This RAG system demonstrates successful:
- Sanskrit text extraction
- Text chunking
- Embedding generation
- Semantic retrieval
- Accurate answer generation

Future Enhancements

- Add PDF support
- Add OCR for Sanskrit manuscripts
- Use IndicBERT or MuRIL for deeper semantic matching
- Deploy system as a web app (FastAPI/Streamlit)

References

- HuggingFace Sentence Transformers
- Ollama Llama Models
- ChromaDB Documentation
- Sanskrit Classical Sources

Thank you!