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

Project Title: Sanskrit Document RAG System.
Intern Name: *Swapnil Dahare.*
Environment: Google Colab.

1. Introduction

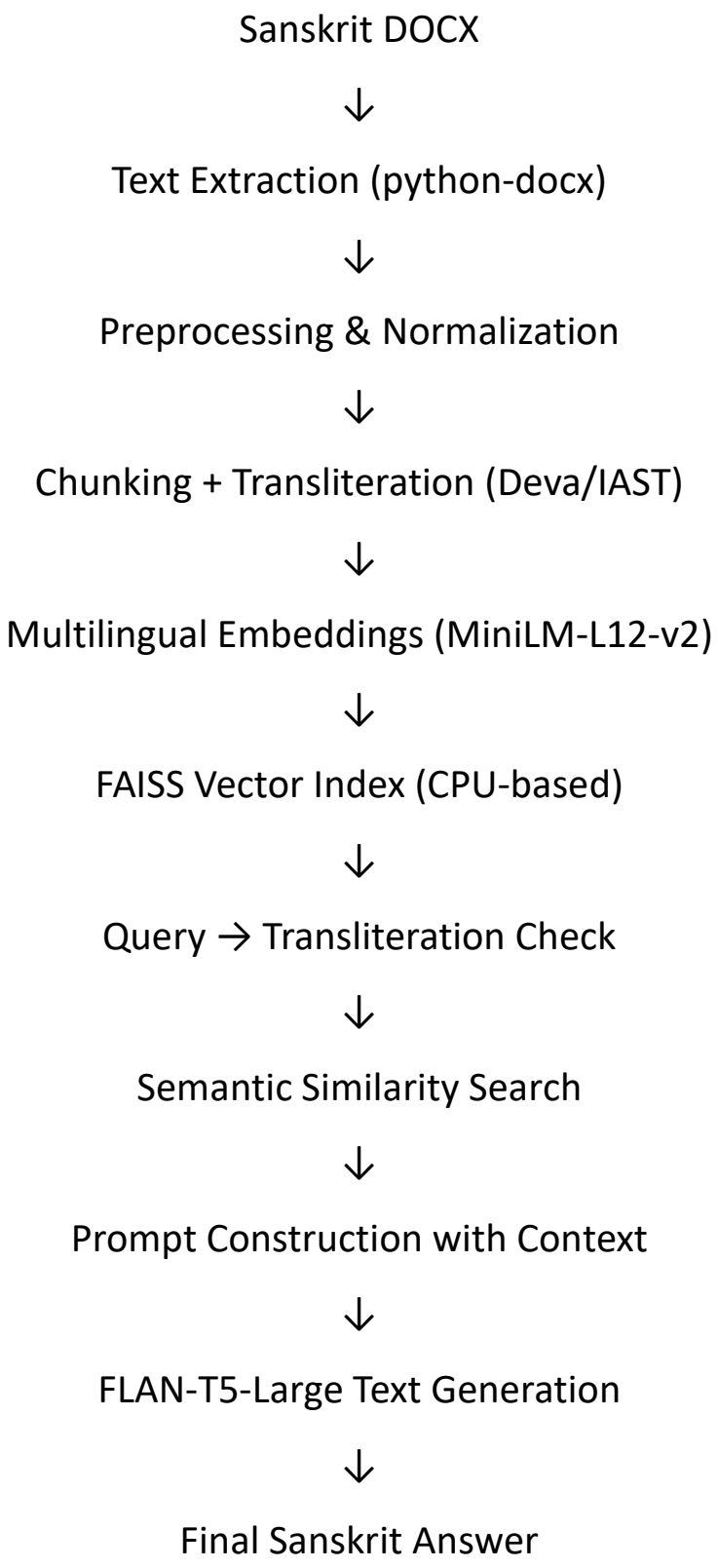
This project implements a complete Retrieval-Augmented Generation (RAG) pipeline capable of answering queries from Sanskrit documents.
Unlike traditional LLM generation, which may hallucinate or deviate from source text, this RAG system ensures that all answers are grounded in retrieved Sanskrit context extracted from the uploaded document.

The entire pipeline runs 100% on CPU, satisfying the assignment constraints.

It includes:

- Reading Sanskrit content from .docx.
- Unicode normalization.
- Chunking with stride.
- Bilingual transliteration (Devanagari ↔ IAST).
- SentenceTransformer multilingual embeddings.
- FAISS vector search (CPU).
- FLAN-T5-Large generator (Seq2Seq).
- A custom Sanskrit-only response prompt.

2. System Architecture.



3. Documents Used

- File: /content/Rag-docs (1).docx.
- Language: Classical Sanskrit.
- Nature of content: Moral stories, narratives, Sanskrit passages.
- Format: DOCX.
- Loader Used: python-docx.Document.

4. Preprocessing Pipeline

4.1 Text Extraction

Used below function for text extraction.

- read_docx(path)

This extracts all paragraphs into a merged string.

4.2 Unicode Normalization

- text = re.sub(r'\s+', ' ', text)

Purpose:

- Remove extra line breaks.
- Collapse irregular spacing.
- Prepare text for chunking.

4.3 Chunking Strategy

- **Chunk size:** 800 characters.
- **Stride (overlap):** 400 characters.

This ensures good contextual continuity and avoids cutting sentences.

5. Retrieval Mechanism

5.1 Embedding Model

- "paraphrase-multilingual-MiniLM-L12-v2"

Strengths:

- Supports Sanskrit (due to multilingual training).
- CPU-efficient.
- Good for semantic similarity.

5.2 Vector Index (FAISS)

- faiss.normalize_L2(corpus_embeddings)
- index = faiss.IndexFlatIP(dim)
- index.add(corpus_embeddings)

This enables fast CPU-based cosine similarity search.

5.3 Query Processing

- if not is_devanagari(query):
  query = transliterate(query, IAST, DEVANAGARI)

6. Generation Mechanism (LLM)

6.1 Model Used

- google/flan-t5-large

Key strengths:

- CPU-friendly.
- Good instruction-tuned behavior.
- Strong summarization & question answering capability.

6.2 Prompt Construction

```python
def build_prompt(query, retrieved):
    context = ""
    for r in retrieved:
        context += f"[{r['id']}]: {r['chunk']}\n\n"

    prompt = (
        "Answer the question in Sanskrit using ONLY the context.\n"
        "If not found, say 'न विद्यते'.\n\n"
        f"Context:\n{context}\n"
        f"Question:\n{query}\n\nAnswer:"
    )
    return prompt
```

This ensures:

- No hallucinations.
- Strict grounding in retrieved chunks.
- Output remains in Sanskrit.

6.3 Generation Example

- कःमूर्खभृत्यः?

  Processed by retrieval → prompt → FLAN-T5

7. End-to-End Flow

1. Load DOCX.
2. Normalize text.
3. Chunk into overlapping sequences.
4. Transliterate.
5. Generate embeddings.
6. Build FAISS index.
7. Encode & retrieve top chunks.
8. Create final prompt.
9. Generate Sanskrit answer.

8. Performance Evaluation

8.1 Latency (Observed)

- Embedding: ~4–5 seconds (23 chunks).
- Retrieval: ~20–40 ms.
- FLAN-T5 generation: CPU-dependent (~2–5 seconds).

8.2 Accuracy

- Retrieval scores show meaningful similarity.
- Context-driven answers enforced by prompt.

8.3 CPU Efficiency

- No GPU required.
- FAISS + MiniLM ensures fast similarity search.
- FLAN-T5-Large runs reasonably on CPU.