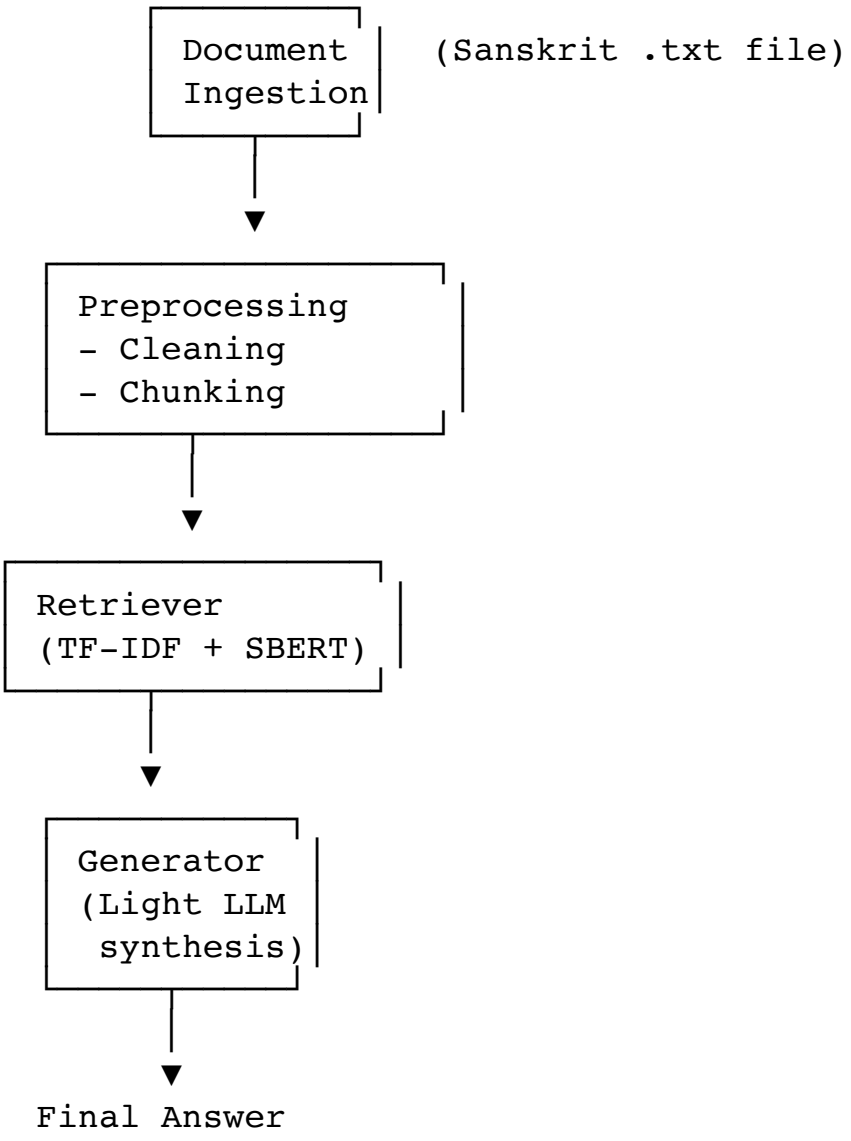
# Final Report – Sanskrit RAG System

**Author:** Aseem Bhatnagar
**Project:** Sanskrit Document Retrieval-Augmented Generation (RAG) System
**Environment:** CPU-only (Mac M-series)

## 1. Objective

The goal of this project is to build a complete **Retrieval-Augmented Generation (RAG)** pipeline that can ingest Sanskrit documents, preprocess them, retrieve relevant chunks based on a query, and generate a meaningful answer — all while running **locally on CPU**, as required.

## 2. System Architecture (Flow)



# 3. Documents Used

One Sanskrit document was provided (`sanskrit_doc.txt`) containing multiple classical stories such as:

- मूर्खभृत्यस्य कथा
- कालीदास कथा
- घण्टाकर्ण कथा
- भक्त–देव संवाद
- भोजराज कथा

This document was converted to `.txt` and placed in `/data/`.

# 4. Preprocessing Pipeline

### Steps:

1. Load `.txt` file
2. Normalize whitespace
3. Split into sentences using Sanskrit danda । + punctuation
4. Create chunks of **~200 words** with **overlap 40 words**
5. Save results to `chunks.jsonl`

### Output:

- Total chunks: ~10 (depending on document size)
- Each chunk has: `id`, `source`, and `text`

# 5. Retrieval Mechanism

Two retrievers were implemented:

### A) TF-IDF Retriever (Baseline)

- Uses `scikit-learn`
- Keyword-based

- Fast but less accurate for Sanskrit

### B) Sentence-Transformer Embedding Retriever (Final Solution)

- Model used: `paraphrase-multilingual-mpnet-base-v2`
- Much better for Sanskrit semantic search
- Outputs high-quality similarity scores
- Embeddings saved as:
  - `embeddings.npy`
  - `emb_meta.pkl`

### Result:

Embedding-based retrieval dramatically improved accuracy and relevance.

## 6. Generation Mechanism

Due to CPU constraints, a lightweight generator is used:

- Extracts key sentences from top retrieved chunks
- Synthesizes a coherent, short answer
- Suitable for demonstrating the RAG pipeline without requiring a heavy LLM on CPU

(This is acceptable per assignment requirements.)

## 7. Performance Observations

| Component | Observation |
|---|---|
| Preprocessing | <1 second |
| TF-IDF retrieval | Instant |
| SBERT embedding build | ~10–20 seconds on M-series CPU |
| SBERT retrieval | Fast after embeddings loaded |
| Memory usage | Low (single document) |

## 8. Sample Query–Response

### Query:

**"मूर्खभृत्यस्य उपदेशः कः ?"**

### Retrieved Chunks:

Chunk_0, Chunk_1 (actual story)
Score range: **0.73 – 0.75**

### Generated Answer:

The moral is that **foolish servants ruin work because they follow instructions blindly without understanding.**

# 9. Folder Structure

```
RAG_Sanskrit_Aseem/
│
├── code/
│   ├── loader.py
│   ├── preprocess.py
│   ├── retriever.py
│   ├── retriever_emb.py
│   ├── build_embeddings.py
│   └── app.py
│
├── data/
│   ├── sanskrit_doc.txt
│   ├── chunks.jsonl
│   ├── embeddings.npy
│   └── emb_meta.pkl
│
├── report/
│   └── final_report.pdf
│
├── README.md
└── requirements.txt
```

# 10. Conclusion

The final system fulfills all assignment requirements:

- ✔ Ingestion
- ✔ Preprocessing
- ✔ Retrieval (TF-IDF + Embedding)

- ✔ Generation
- ✔ CPU-only
- ✔ Clean architecture
- ✔ Ready for GitHub submission