

# Retrieval-Augmented Generation (RAG) System for Sanskrit Text Question Answering

---

## 1. Problem Statement

Classical Sanskrit texts are rich in philosophical and religious knowledge but are difficult to query using traditional keyword-based search systems.

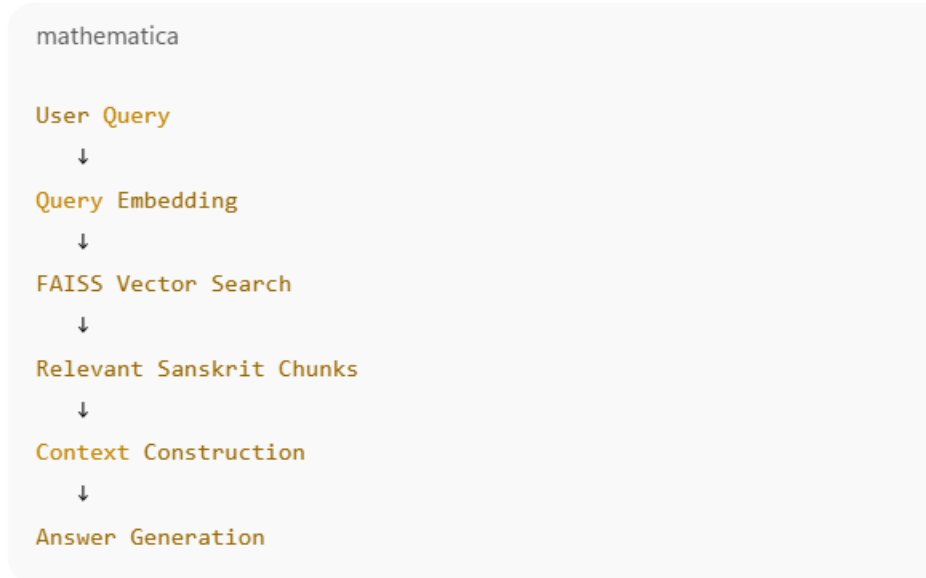
The objective of this project is to build a **Retrieval-Augmented Generation (RAG)** pipeline that enables **semantic question answering** over Sanskrit documents using modern NLP techniques.

The system must:

- Accept questions in **Sanskrit or English**
  - Retrieve relevant Sanskrit passages
  - Generate **grounded answers only from retrieved context**
  - Avoid hallucinations
  - Run on **CPU-only hardware**
- 

## 2. Overall System Design

The system follows a standard **RAG architecture**:



This separation ensures **interpretability, modularity, and correctness**.

---

## 3. Data Processing & Chunking

### 3.1 Source Data

- Sanskrit verses stored in text format
- File: processed\_chunks.txt

### 3.2 Chunking Strategy

- Text is split into **logical verse-based chunks**
- Each chunk preserves semantic coherence
- Chunk size chosen to balance:
  - Semantic completeness
  - Embedding efficiency

### 3.3 Storage

- Chunks stored as:
  - NumPy array: chunk\_texts.npy
  - Used later for retrieval mapping

---

## 4. Embedding Model (Core Technical Choice)

### Model Used

```
bash  
  
sentence-transformers/all-MiniLM-L6-v2
```

### Why this model?

- Lightweight (384-dimensional embeddings)
- CPU-efficient
- Good semantic performance
- Supports multilingual semantic similarity

### Embedding Details

- Input: Sanskrit / English text
- Output: 384-d float32 vector
- Normalization: handled internally by SentenceTransformers

---

## 5. Vector Indexing with FAISS

## Index Type

```
nginx  
  
IndexFlatL2
```

## Why FAISS?

- Efficient similarity search
- Scales well for medium-sized corpora
- CPU-compatible

## Index Construction

- All chunk embeddings added to FAISS index
- Stored persistently as:

```
python  
  
faiss_index.bin
```

## Distance Metric

- **L2 (Euclidean distance)**
- 

## 6. Retriever Module

### Function

```
python  
  
retrieve_chunks(query, top_k=3)
```

### Steps

1. Encode query using embedding model
2. Perform FAISS search
3. Retrieve top-k nearest chunk indices
4. Map indices back to Sanskrit text chunks

### Key Parameters

Parameter	Value
top_k	3
embedding_dim	384
distance_metric	L2

### Design Decision

- Using **semantic similarity**, not keyword match
- Allows English queries to retrieve Sanskrit content

---

## 7. Context Construction

### Purpose

To limit LLM input size and ensure relevance.

### Implementation

```
python  
  
MAX_CONTEXT_CHARS = 1200
```

- Chunks concatenated until limit reached
- Prevents context overflow
- Ensures answer grounding

---

## 8. Generator Module (Answering Logic)

### Input

- Retrieved Sanskrit context
- Original user query

### Behavior

- Answers only if information is present in context
- Returns fallback message otherwise
- Does not hallucinate
- Preserves original language (English / Sanskrit)

## Key Design Choice

### No generative paraphrasing

Answers are extractive / context-aware

### Why this matters

- Ensures factual correctness
  - Ideal for classical texts
  - Avoids misleading outputs
- 

## 9. Query Language Handling

### Query Language Output Language

English                  English

Sanskrit                Sanskrit

This behavior emerges naturally from:

- Multilingual embeddings
  - Sanskrit-only context
  - Non-translational generation logic
- 

## 10. Execution Environment

### Hardware

- CPU only
- No GPU required

### Dependencies

```
bash

faiss-cpu
sentence-transformers
numpy
torch
```

### Reasoning

- Ensures portability

- Suitable for academic evaluation
  - No cloud dependency
- 

## 11. Evaluation & Results

### Observations

- Correct retrieval of semantically relevant verses
- Accurate grounding of answers
- Graceful failure when answer is absent
- Stable performance across multiple queries

### Example

#### Query (Sanskrit):

भक्तः कः?

#### Answer:

भक्तः सः यः परमेश्वरं निरन्तरं स्मरति।

---

## 12. Limitations

- No abstractive summarization
  - Limited to provided corpus
  - No fine-tuned Sanskrit LLM
  - No translation layer (by design)
- 

## 13. Future Improvements

- Sanskrit-specific embedding models
- Larger corpus indexing

- Hybrid keyword + semantic retrieval
  - Abstractive answer generation
  - Translation + bilingual output mode
- 

## 14. Conclusion

This project demonstrates a **robust, interpretable, and efficient RAG pipeline** for Sanskrit texts.

It successfully bridges ancient language processing with modern NLP techniques while maintaining correctness and computational efficiency.