svnapro

RAG-BASED PDF QUESTION ANSWERING SYSTEM

USING RETRIEVAL- AUGMENTED GENERATION WITH OPEN SOURCE LLMS

PROBLEM STATEMENT

Challenge: Traditional LLMs have limitations when dealing with specific documents or proprietary information because they only know what they were trained on.

Problems:

1. **LLMs cannot access real-time or document-specific information**
2. **Risk of hallucination when asked about unfamiliar content**
3. **No ability to cite sources for generated answers**
4. **Cannot learn from new documents without retraining**

OUR SOLUTION

Build a RAG system that allows LLMs to answer questions from uploaded PDFs with accurate source citations.

WHAT IS RAG?

Retrieval-Augmented Generation (RAG)

RAG combines two powerful techniques:

1. **Retrieval: Finding relevant information from a knowledge base**
2. **Generation: Using an LLM to generate natural language answers**

WHAT IS RAG?

How it works:

1. Document is split into chunks and stored in a vector database
2. User asks a question
3. System retrieves most relevant chunks
4. LLM generates answer based on retrieved context
5. Sources are displayed for transparency

Key Benefit: Grounded, factual responses with source attribution

RAG ARCHITECTURE

System Flow:

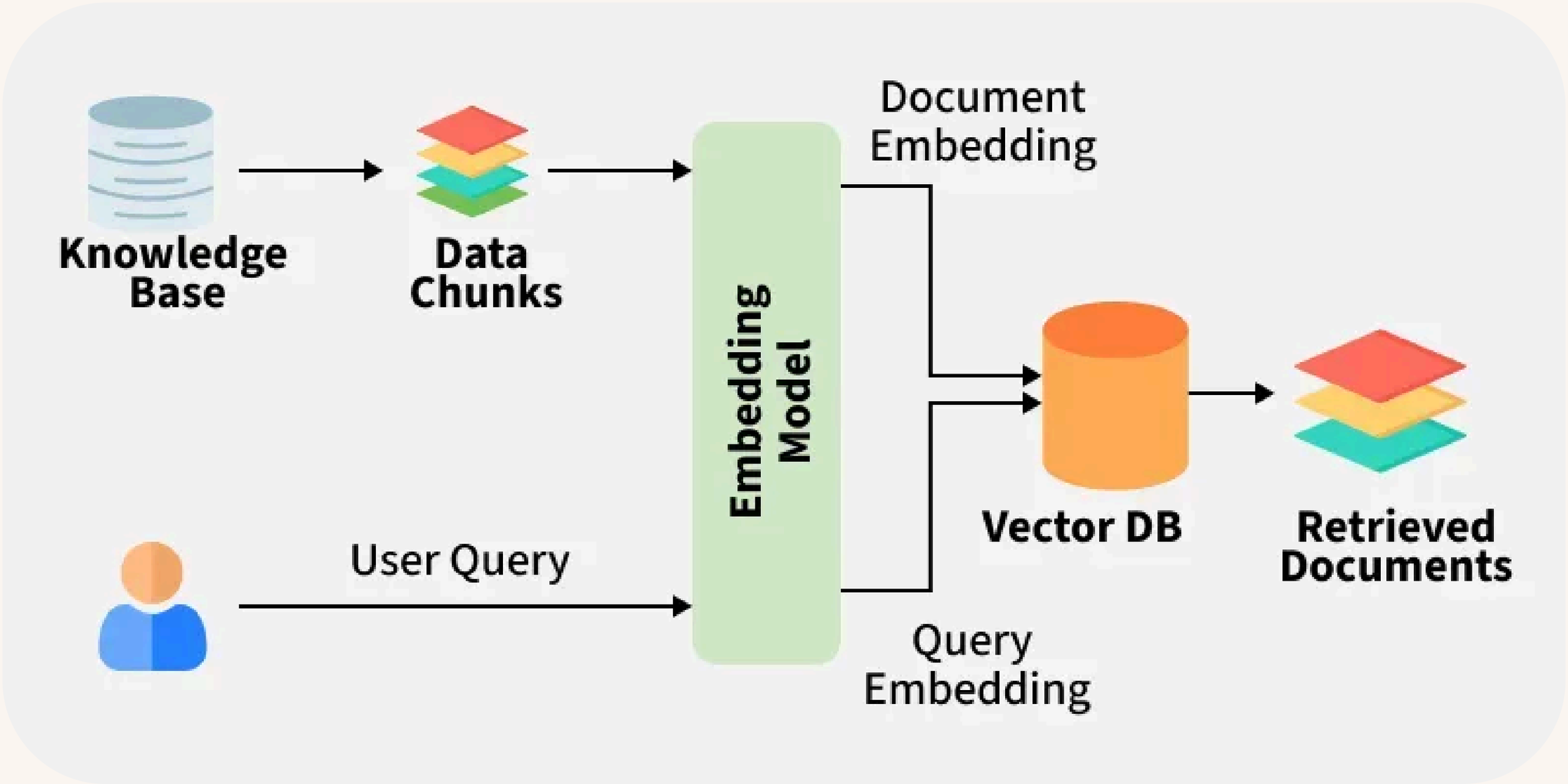
a. **Document ProcessingUpload PDF → Extract text → Split into chunks**

b. **Embedding & StorageConvert chunks to vectors → Store in FAISS database**

c. **Query ProcessingUser question → Convert to vector → Retrieve similar chunks**

d. **Answer GenerationPass context + question to LLM → Generate answer → Return with sources**

RAG ARCHITECTURE



LIBRARIES & TECHNOLOGIES

Core Framework:

- Streamlit: Web interface for easy interaction
- LangChain: Orchestrates RAG pipeline components

NLP & ML:

- Transformers: Hugging Face library for loading pre-trained models
- PyTorch: Deep learning framework powering the models
- Sentence-Transformers: Creates semantic embeddings

Vector Storage:

- FAISS: Facebook AI Similarity Search for efficient vector retrieval

Document Processing:

- PyPDF2: Extracts text from PDF files

MODEL SELECTION

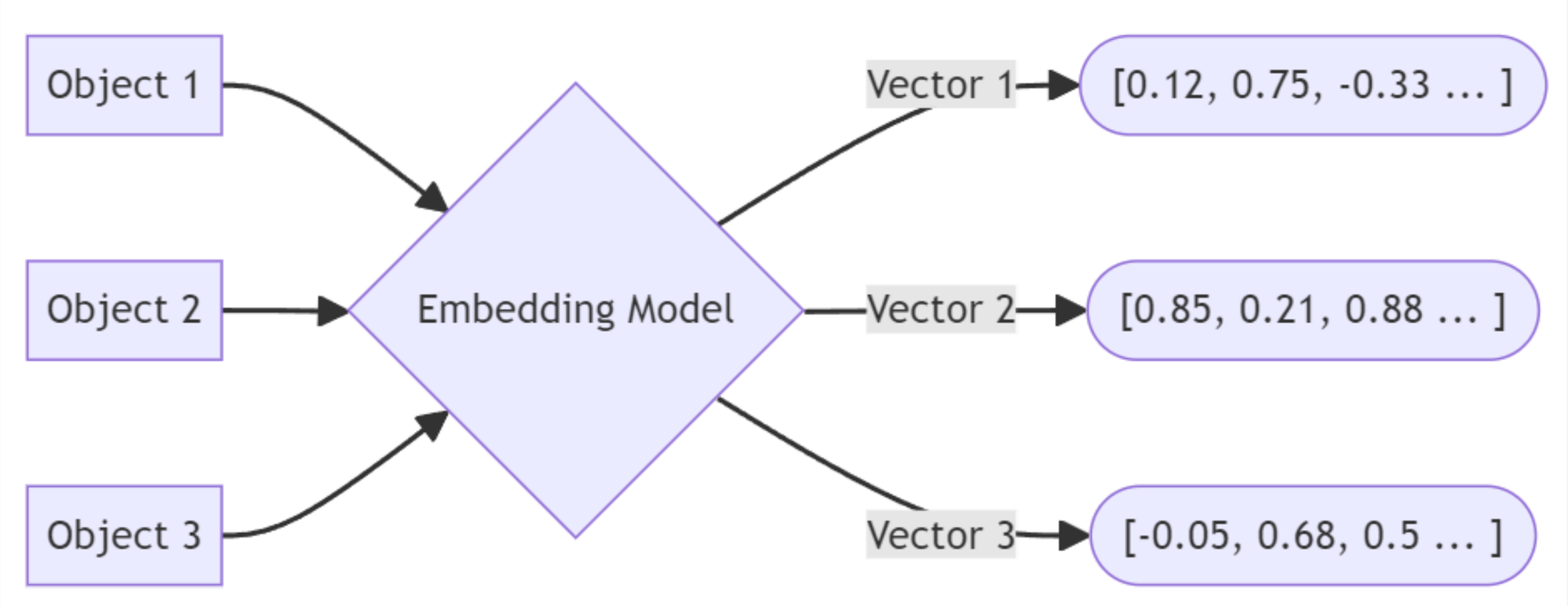
LLM: FLAN-T5-Base

- Google's instruction-tuned T5 model
- 250M parameters
- Excellent question-answering capabilities
- Runs locally without API costs
- ~900MB download size

Embedding Model: all-MiniLM-L6-v2

- Compact and efficient
- 384-dimensional embeddings
- Fast encoding speed
- Good semantic understanding
- ~90MB model size

WHAT ARE EMBEDDINGS?



Vector embeddings are numerical representations of various data types, such as text, images, or audio, that machine learning models can understand. These high-dimensional numerical arrays, or vectors, capture the meaning and relationships within the data, causing similar data points to cluster together in a multidimensional space.

TECHNICAL APPROACH

Step 1: PDF Upload & Text Extraction

- PDF File → PyPDF2 → Raw Text

Step 2: Text Chunking

- Chunk size: 1000 characters
- Overlap: 200 characters (maintains context)
- Recursive splitting preserves sentence boundaries

Step 3: Embedding Generation

- Each chunk → Sentence-Transformer → 384-dim vector
- Captures semantic meaning of text

Step 4: Vector Storage

- Vectors stored in FAISS index
- Enables fast similarity search

LET’S CODE