RAG PIPELINE FOR INDIC LANGUAGES

Want to build a RAG system for Indic languages like Telugu? Here's how you can do it step-by-step!

WHY RAG FOR INDIC LANGUAGES?

Indic languages like Telugu, Hindi, and Tamil are rich but lack AI resources. RAG (Retrieval-Augmented Generation) helps by:

- Bridging the resource gap.
- Providing context-aware responses.
- Scaling efficiently for large applications.

HOW DOES RAG WORK?

1. **Extract text** (e.g., Telugu dataset).
2. **Chunk** text into smaller pieces.
3. **Embed** chunks into numerical vectors.
4. **Store** embeddings in a vector database (e.g., FAISS).
5. **Retrieve** relevant chunks for a query.
6. **Generate** a response using an LLM (e.g., Groq's Mixtral).

LIBRARIES NEEDED

```
pip install transformers
pip install sentence-transformers
pip install groq
pip install faiss-cpu
pip install datasets
```

WHAT DO THESE LIBRARIES DO?

1. **Transformers:** Work with pre-trained language models.
2. **Sentence-Transformers:** Generate embeddings for text.
3. **Groq:** Fast and scalable LLM inference.
4. **FAISS:** Efficient similarity search for retrieval.
5. **Datasets:** Load and process datasets (e.g., Telugu text).

EXTRACTING TEXT

1. Start by loading a Telugu text dataset (e.g., from Hugging Face's datasets library).

```python
from datasets import load_dataset
# Load a Telugu dataset (example: OSCAR Telugu subset)
dataset = load_dataset("oscar", "unshuffled_deduplicated_te", split="train")
# Extract Telugu text
telugu_texts = [example["text"] for example in dataset]
```

SPLITTING TEXT INTO CHUNKS

1. Split the Telugu text into smaller chunks for efficient retrieval.

```python
from transformers import AutoTokenizer

# Load a Telugu tokenizer (e.g., IndicBERT)
tokenizer = AutoTokenizer.from_pretrained("l3cube-pune/telugu-bert")
# Function to split text into chunks
def split_into_chunks(text, max_tokens=128):
    tokens = tokenizer.encode(text, truncation=False, return_tensors="pt")[0]
    chunks = [tokens[i:i + max_tokens] for i in range(0, len(tokens), max_tokens)]
    return [tokenizer.decode(chunk, skip_special_tokens=True) for chunk in chunks]
# Split all Telugu texts into chunks
telugu_chunks = []
for text in telugu_texts:
    chunks = split_into_chunks(text)
    telugu_chunks.extend(chunks)
```

GENERATING EMBEDDINGS

1. Convert the text chunks into embeddings using a multilingual embedding model.

```python
from sentence_transformers import SentenceTransformer
# Load a multilingual embedding model
embedding_model = SentenceTransformer("paraphrase-multilingual-mpnet-base-v2")
# Generate embeddings for Telugu chunks
chunk_embeddings = embedding_model.encode(telugu_chunks, show_progress_bar=True)
```

STORING EMBEDDINGS IN A VECTOR DATABASE

Use **FAISS** to store and index the embeddings for fast retrieval.

```python
import faiss
import numpy as np
# Convert embeddings to a numpy array
embeddings_array = np.array(chunk_embeddings).astype("float32")
# Create a FAISS index
dimension = embeddings_array.shape[1]
index = faiss.IndexFlatL2(dimension)  # L2 distance for similarity search
index.add(embeddings_array)
```

LOADING THE LLM

Use **Groq's API** to load a fast and powerful LLM (e.g., Mixtral).

```python
from groq import Groq
# Initialize Groq client
client = Groq(api_key="your_api_key")

def query_with_llm(query: str, context: str) -> str:
    # Combine the query and context into a prompt
    prompt = f"ప్రశ్న: {query}\nసందర్భం: {context}"

    # Query the Groq API
    response = client.chat.completions.create(
        model="llama-3.3-70b-versatile",  # Replace with your preferred Groq-supported model
        messages=[
            {"role": "system", "content": "You are a helpful assistant in Telugu language."},
            {"role": "user", "content": prompt}
        ]
    )
    # Return the generated response
    return response.choices[0].message.content.strip()
```

QUERYING THE SYSTEM

1. Convert the user query into an embedding.
2. Retrieve the top-matching chunks from the FAISS index.
3. Pass the chunks to the LLM to generate a response.

```python
def rag_system(query: str, top_k: int = 50) -> str:
    # Step 1: Convert query to embedding
    query_embedding = embedding_model.encode([query], show_progress_bar=False)
    query_embedding = np.array(query_embedding).astype("float32")

    # Step 2: Retrieve top-k matching chunks
    distances, indices = index.search(query_embedding, top_k)
    retrieved_chunks = [telugu_chunks[idx] for idx in indices[0]]

    # Step 3: Combine retrieved chunks into a single context
    context = " ".join(retrieved_chunks)

    # Step 4: Generate a response using Groq's API
    response = query_with_llm(query, context)

    return response
```

RUNNING THE SYSTEM

Run the system with a Telugu query and see the response.

EXAMPLE OUTPUT

```
# Example usage
query = "పీఎం అమెరికా టూర్ కి ఎందుకు వెళ్లారు?"
response = rag_system(query)
print("User Query:", query)
print("System Response:", response)
```

TRY IT YOURSELF!

- Sign up for Groq API access.
- Replace "your_groq_api_key_here" with your API key.
- Run the code and experiment with Telugu queries!

STATFUSION Ai

FOLLOW US ON

/ STATFUSIONAI

LET'S CHAT!

Have questions or need help? Drop a comment below!
Or tag a friend who loves building AI systems for Indic languages!