Project Documentation: Retrieval-Augmented Generation (RAG) for Indian Legal Applications

---

1. Project Overview

Objective

This project aims to implement a **Retrieval-Augmented Generation (RAG)** system to answer legal questions based on Indian law. The system integrates semantic retrieval with natural language generation (NLG) to deliver accurate and context-sensitive answers.

Key Features

1. **Preprocessing:** Data cleaning, deduplication, and standardization.
2. **Semantic Search:** FAISS-based retrieval of legal context.
3. **Answer Generation:** Leverages the **Flan-T5-base** model for generating human-like, grounded responses.

Why Retrieval-Augmented Generation (RAG)?

Definition

RAG combines:

1. **Retrieval:** Fetches relevant context from a knowledge base.
2. **Generation:** Uses a language model to create coherent responses based on the context and query.

Advantages

- **Grounded Responses:** Anchored in real data, reducing hallucinations.
- **Dynamic Updates:** Knowledge base updates do not require retraining.
- **Improved Accuracy:** Combines domain-specific retrieval with powerful language models.

---

2. Dataset Description

Datasets Used

1. **Indian Constitution QA:** Fundamental rights, governance, duties.
2. **CrPC QA:** Procedural law (arrests, bail, judicial processes).
3. **IPC QA:** Substantive law (offenses and punishments).

Structure

Each dataset is in JSON format:

- **Question:** A legal query.
- **Answer:** Corresponding explanation or legal section.

---

3. Code Workflow

Step 1: Install Required Libraries

Purpose:

Install libraries like **Transformers**, **FAISS**, **Sentence-Transformers**, and **Torch** for model usage, embedding creation, and similarity searches.

Step 2: Load and Combine Datasets

Purpose:

Load JSON files and merge all datasets into a unified list (combined_data).

Step 3: Data Cleaning and Preprocessing

Steps:

- Convert text to lowercase.
- Remove extra spaces for consistent formatting.

Step 4: Remove Duplicate Questions

Functionality:

- Uses a dictionary to filter unique questions.
- Outputs a cleaned list (cleaned_data).

Step 5: Save the Cleaned Data

Purpose:

Save processed data as cleaned_legal_data.json to avoid redundant preprocessing in future runs.

Step 6: Embedding Creation

Details:

- **Model:** all-mpnet-base-v2 (Sentence-BERT).
- **Output:** Dense 768-dimensional embeddings for semantic similarity.
- **Usage:** Saved as .npy for FAISS indexing.

Step 7: Build FAISS Index

Purpose:

Create a FAISS index using embeddings for fast retrieval.

- **IndexFlatL2:** Uses Euclidean distance for similarity search.

Step 8: Define the Retrieval Function

Process:

1. Convert query into an embedding.
2. Use FAISS to find the most similar entries in the dataset.

3. Fetch top-k question-answer pairs based on similarity.

Step 9: Load Flan-T5-Base Model

Functionality:

- Load **Flan-T5-base** model and tokenizer for NLG tasks.
- Utilize GPU acceleration (if available) for faster computations.

Step 10: Generate Answers

Workflow:

1. Retrieve top-k context using FAISS.
2. Combine query and context into a format understood by Flan-T5.
3. Use the model to generate a coherent, human-like answer.
4. Output: A concise response tailored to the query and context.

---

4. Key Technical Concepts

1. Retrieval-Augmented Generation (RAG)

Definition:

RAG combines retrieval and generation to provide accurate, data-driven responses.

Advantages:

- **Grounded Responses:** Tied to real data.
- **Dynamic Updates:** Easy knowledge base updates.
- **Accuracy:** Combines retrieval precision with generative fluency.

Example:

- **Query:** "What are fundamental rights?"
- **Retrieved Context:** Relevant sections from the Indian Constitution dataset.
- **Generated Answer:** "Fundamental rights include equality, freedom, and protection from discrimination."

---

2. Natural Language Generation (NLG)

Model:

- **Flan-T5-base:** Instruction-tuned for question-answering tasks.
- **Outcome:** Contextually relevant, human-readable responses.

---

3. FAISS (Facebook AI Similarity Search)

Purpose:

Efficiently search embeddings for semantic similarity.

- **IndexFlatL2:** Measures Euclidean distance between embeddings.

---

4. Sentence-BERT Model (all-mpnet-base-v2)

Details:

- Generates compact embeddings (768 dimensions) for semantic matching.
- Captures deeper meanings beyond lexical similarities.

---

5. Future Improvements

1. **Multilingual Support:** Extend to regional Indian languages.
2. **Knowledge Base Expansion:** Include case laws and recent amendments.
3. **Interactive Features:** Enable live feedback and query refinement.