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

**Sourikta Nag**

**1. Project Overview**

**Objective**

This project aims to create a **Retrieval-Augmented Generation (RAG)** system for answering legal questions based on Indian law. This system combines **semantic retrieval** with **natural language generation (NLG)** to provide accurate, context-aware answers.

**Key Features**

* **Preprocessing**: Data cleaning, deduplication, and standardization.
* **Semantic Search**: FAISS-based retrieval of relevant legal context.
* **Answer Generation**: Using the pre-trained Flan-T5-base model for generating human-like responses.

**Why Retrieval-Augmented Generation (RAG)?**

**Definition**:  
RAG is an AI approach that combines:

1. **Retrieval**: Fetches relevant context from a knowledge base.
2. **Generation**: Uses a language model to generate a coherent response based on the retrieved context and the user’s query.

**Advantages**:

* **Grounded Responses**: Ensures answers are based on real data.
* **Dynamic Knowledge Base**: Retrieval allows easy data updates without retraining the model.
* **Improved Accuracy**: Reduces "hallucination" (errors) common in standalone models.

**2. Dataset Description**

[**Datasets Used**](https://www.kaggle.com/datasets/akshatgupta7/llm-fine-tuning-dataset-of-indian-legal-texts)

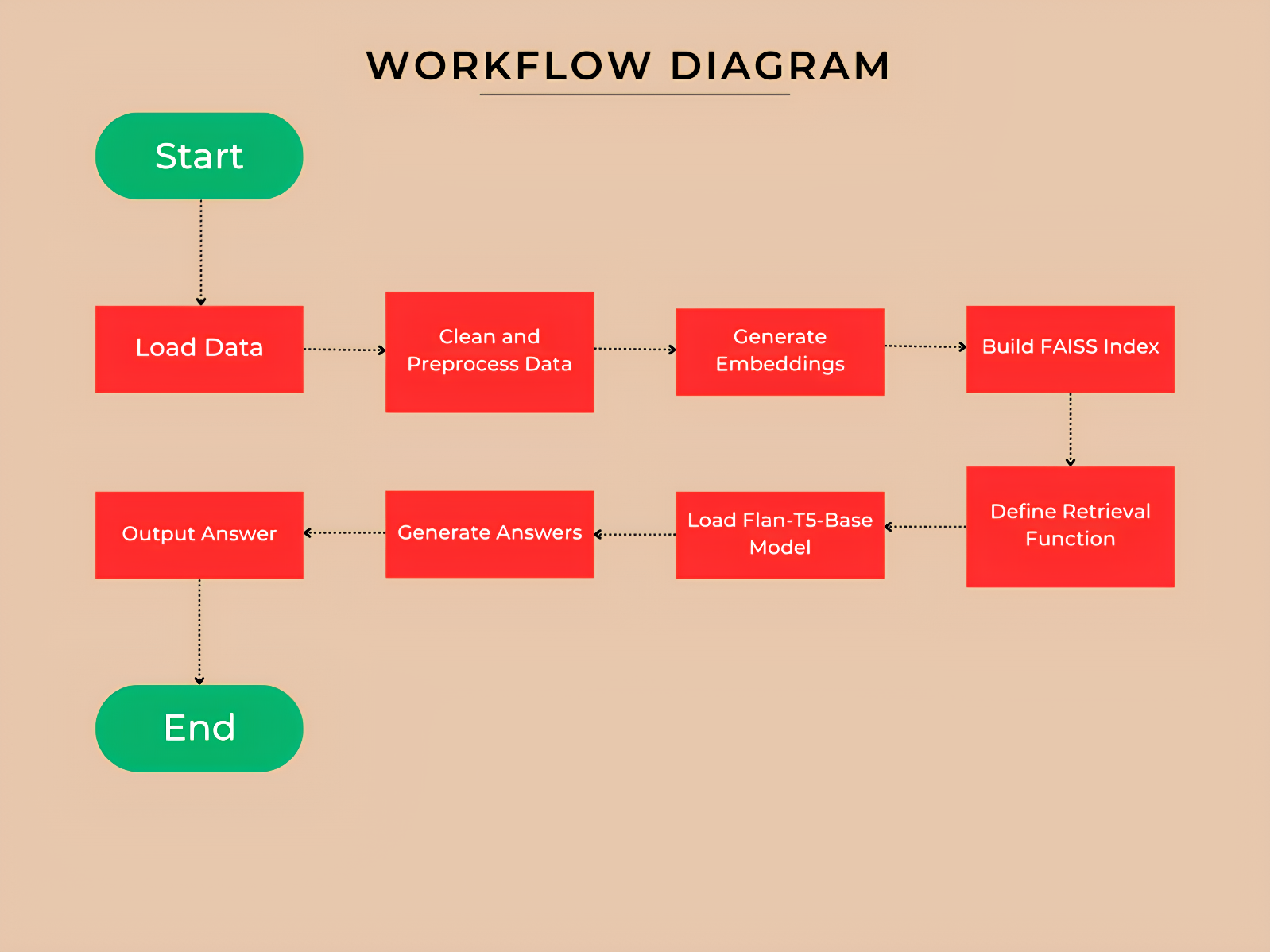
Three datasets were used:

1. **Indian Constitution QA**: Covers topics like fundamental rights, governance, and duties.
2. **CrPC QA**: Focuses on procedural aspects like arrests, bail, and judicial processes.
3. **IPC QA**: Covers substantive criminal law, including offenses and their punishments.

**Structure**

Each dataset is a JSON file with:

* **Question**: Legal queries.
* **Answer**: Explanations or sections of the law.



**3. Step-by-Step Explanation of Code**

**Step 1: Install Required Libraries**

**Explanation:**

* Installs necessary libraries:
  + **Transformers**: These are used for using pre-trained models like Flan-T5-base.
  + **FAISS-CPU**: For efficient similarity search.
  + **Sentence-Transformers**: For embedding generation.
  + **Torch**: For computations related to deep learning models.

**Step 2: Load and Combine the Dataset**

**Explanation:**

* **Purpose**: Loads JSON datasets into Python and combines them into a single list, combined\_data.
* **Functionality**: Merges legal QA pairs from the Constitution, CrPC, and IPC datasets.

**Step 3: Data Cleaning and Preprocessing**

**Explanation:**

* **Purpose**: Ensures consistent formatting for better retrieval accuracy.
* **Steps**:
  + Converts text to lowercase to avoid case-sensitive mismatches.
  + Removes extra spaces to clean up formatting.

**Step 4: Remove Duplicate Questions**

**Explanation:**

* **Purpose**: Removes duplicate entries to avoid redundancy in retrieval.
* **Functionality**:
  + Uses a dictionary (unique\_entries) to store unique questions.
  + Converts the dictionary back to a list (cleaned\_data).

**Step 5: Save the Cleaned Data**

**Explanation:**

* **Purpose**: Saves the cleaned dataset as cleaned\_legal\_data.json for reuse in future runs.
* **Significance**: Eliminates the need to reprocess the raw data each time.

**Step 6: Embedding Creation**

**Explanation:**

* **What are embeddings?** Numerical representations of text capturing semantic meaning.
* **Model Used**: all-mpnet-base-v2, a Sentence-BERT model optimized for semantic similarity.
* **Functionality**:
  + Generates embeddings for all QA pairs.
  + Saves the embeddings as a .npy file for use in FAISS.

**Step 7: Build a FAISS Index**

The **FAISS** (**Facebook AI Similarity Search**) library is used to create an index of embeddings. These embeddings represent the semantic meaning of the question-answer pairs in the cleaned dataset. The FAISS index allows the system to efficiently find the top-k most relevant question-answer pairs for a user query based on similarity.

**Detailed Steps:**

1. **Embedding Dimensions:**
   * Each embedding generated by the all-mpnet-base-v2 model has a fixed number of dimensions (768 for this model). The dimensionality (embedding\_dim) is necessary to initialize the FAISS index.
2. **FAISS Index Initialization:**
   * IndexFlatL2 creates a FAISS index that uses Euclidean distance to measure similarity. This means that the closer two embeddings are in the embedding space, the more semantically similar they are.
   * Why Euclidean Distance?
     + It is simple and effective for semantic similarity tasks when embeddings are normalized.
3. **Adding Embeddings to the Index:**
   * All embeddings from the cleaned dataset are added to the FAISS index.
   * These embeddings act as references that the system will compare user query embeddings against during retrieval.

**Purpose:**

* Build an index of embeddings for fast retrieval.

**Outcome:**

* A fully initialized FAISS index containing all embeddings from the dataset. This index enables fast and efficient similarity searches when a query is submitted.

**Step 8: Define the Retrieval Function**

**What is Happening?**

The retrieval function, retrieve\_top\_k, uses the FAISS index to find the most relevant question-answer pairs for a given query. The function compares the query's embedding to the embeddings stored in the FAISS index and returns the top-k closest matches.

**Detailed Steps:**

1. **Query Embedding Generation:**
   * The user’s query (e.g., *"What are the fundamental rights?"*) is passed through the SentenceTransformer model to generate its embedding. This embedding captures the semantic meaning of the query.
2. **FAISS Similarity Search:**
   * The query embedding is compared against all embeddings in the FAISS index.
   * The index.search function returns:
     + **Distances:** How far each matched embedding is from the query embedding (lower is better).
     + **Indices:** The positions of the closest embeddings in the dataset.
3. **Extracting Relevant Entries:**
   * Using the indices returned by FAISS, the corresponding question-answer pairs are fetched from the cleaned dataset.

**Outcome:**

* A list of the top-k most relevant question-answer pairs is returned, providing context for the query.

**Step 9: Load the Flan-T5-Base Model**

**What is Happening?**

The **Flan-T5-base** model and its tokenizer are loaded. These components are essential for generating human-like answers based on the retrieved context and query.

**Detailed Steps:**

1. **Model Loading:**
   * The T5ForConditionalGeneration class from the Hugging Face Transformers library loads the pre-trained Flan-T5-base model.
   * This model is optimized for instruction-following tasks, making it ideal for question-answering.
2. **Tokenizer Initialization:**
   * The tokenizer, T5Tokenizer, is used to:
     + Convert input text into tokens (numerical IDs) for the model.
     + Convert the model’s output tokens back into readable text.
3. **CUDA Utilization:**
   * The model and tokenized inputs are moved to the GPU ("cuda") for faster computations if a GPU is available.

**Why Used?**

* Handles complex legal queries effectively.
* Generates contextually relevant, human-like responses.

**Why Used?**

* Handles complex legal queries effectively.
* Generates contextually relevant, human-like responses.

**Outcome:**

* The Flan-T5-base model and tokenizer are ready to generate answers.
* The model is loaded onto the GPU or CPU, depending on available resources.

**Step 10: Generate Answers**

**What is Happening?**

The generate\_answer function combines retrieval and generation to produce an answer. It retrieves relevant context using FAISS and passes it to Flan-T5-base along with the query to generate a response.

Detailed Steps:

1. **Retrieve Context:**
   * The retrieve\_top\_k function fetches the top-k most relevant question-answer pairs for the given query.
2. **Prepare Input for the Model:**
   * The query and the retrieved context are combined into a single input string in a format Flan-T5 can understand.
   * For example:
     + Input: "question: What is bail? context: Bail is a process..."
3. **Generate Response:**
   * The generate method of Flan-T5-base predicts the next words (tokens) step-by-step, producing a complete answer.
   * The max\_length=200 ensures the generated response does not exceed 200 tokens.
4. **Decode the Output:**
   * The generated tokens are converted back into human-readable text using the tokenizer.

**Outcome:**

* The function returns a concise, human-readable answer based on the query and retrieved legal context.
* Example:
  + Query: *"What are fundamental rights?"*
  + Retrieved Context: Text from the Indian Constitution dataset.
  + Generated Answer: *"Fundamental rights include the right to equality, freedom, and protection from discrimination."*

**4. Key Technical Concepts**

This section explains the key technical concepts used in the project, providing detailed insights into their functionality and relevance.

**1. Retrieval-Augmented Generation (RAG)**

**Definition**

Retrieval-Augmented Generation (RAG) is a hybrid AI framework that combines the retrieval of relevant data from a knowledge base with the **generation** of responses using a language model. Instead of relying entirely on a pre-trained language model’s memory, RAG enhances accuracy by grounding responses in real data fetched at query time.

**How It Works:**

1. **Retrieval**:
   * Given a query, a semantic search mechanism (like FAISS) fetches the most relevant pieces of text from a knowledge base.
   * Example: For the query *"What are the rights of an arrested person?"*, the system retrieves relevant sections of the CrPC dataset.
2. **Generation**:
   * The retrieved information is combined with the query and passed to a language model like Flan-T5-base.
   * The language model generates a human-like response informed by the retrieved context.

**Advantages:**

* **Grounded Responses**: Ensures that answers are based on real, retrieved data, reducing inaccuracies (hallucinations) common in standalone models.
* **Dynamic Knowledge Base**: Retrieval allows updating the knowledge base without retraining the language model.
* **Domain Adaptability**: The system can answer domain-specific queries (e.g., legal) by retrieving contextually relevant data.

**Example in This Project:**

For the question:

* **Query**: *"What is the process for granting bail under CrPC?"*
* **Retrieved Context**: Sections from CrPC explaining bail procedures.
* **Generated Answer**: *"The process involves filing an application before a magistrate and demonstrating sufficient grounds for bail."*

**2. Natural Language Generation (NLG)**

**Definition**

Natural Language Generation (NLG) is the capability of AI to produce human-like text based on input data. It is a key NLP (Natural Language Processing) component that focuses on generating fluent, coherent, and grammatically correct sentences.

**How It Works:**

1. **Input Understanding**:
   * The model processes the input query and, optionally, additional context.
2. **Text Generation**:
   * Using a decoder mechanism, the model predicts the next word (or token) in a sequence until it completes the response.

**NLG Model in This Project:**

* **Flan-T5-base**:
  + A pre-trained, instruction-tuned model from Google.
  + Generates answers to legal questions by combining the query with retrieved context.
  + Example:
    - **Input**: *"What are fundamental rights?"* with retrieved context.
    - **Output**: *"Fundamental rights include the right to equality, freedom, and protection against discrimination."*

**Applications of NLG:**

* Chatbots and virtual assistants.
* Summarization of lengthy documents.
* Automated report and content generation.

**Why NLG?**

In this project, NLG ensures that the answers are accurate, human-readable, and contextually appropriate.

**3. FAISS (Facebook AI Similarity Search)**

**Definition**

FAISS (Facebook AI Similarity Search) is a library for fast and efficient similarity searches in large datasets. It is commonly used to match user queries with relevant data points based on vector embeddings.

**How FAISS Works:**

1. **Indexing**:
   * Embeddings (numerical representations of text) are indexed for fast retrieval.
2. **Query Matching**:
   * When a query embedding is submitted, FAISS identifies the closest matches by calculating the distance (e.g., Euclidean) between the query and indexed embeddings.

**Why Use FAISS?**

* **Efficiency**: Handles large datasets with millions of embeddings efficiently.
* **Scalability**: Easily updates with new embeddings or data points.
* **Accuracy**: Retrieves the most semantically similar entries for a given query.

**Example in This Project:**

* **Input Query**: *"What is sedition under IPC?"*
* **Retrieved Context**: Relevant sections about sedition from the IPC dataset.

**4. all-mpnet-base-v2 (Embedding Model)**

**Definition**

**all-mpnet-base-v2** is a pre-trained Sentence-BERT model used to generate embeddings for text. These embeddings are dense vector representations that capture the semantic meaning of the text, making it possible to compare sentences based on their meaning rather than exact words.

**Why all-mpnet-base-v2?**

* **State-of-the-Art Accuracy**: Achieves high accuracy in semantic similarity tasks.
* **Compact Embeddings**: Generates 768-dimensional embeddings, which are efficient for large datasets.
* **Domain Independence**: Works well across various text types without fine-tuning.

**How It Works:**

1. **Input Processing**:
   * Combines the question and answer into a single input, e.g., *"What is bail? Bail is a security to ensure compliance with the law."*
2. **Embedding Creation**:
   * Encodes the input text into a dense vector that captures its meaning.
3. **Similarity Search**:
   * These embeddings are used in FAISS to find the most semantically relevant entries.

**Applications:**

* Text similarity detection.
* Semantic search.
* Question-answering systems.

**Example in This Project:**

For the question *"What is bail?"*, the model generates an embedding that allows it to retrieve entries explaining bail procedures from the CrPC dataset.

**5. Flan-T5-Base**

**Definition**

Flan-T5 is a pre-trained language model developed by Google that treats all NLP tasks as text-to-text problems. The **Flan-T5-base** variant used in this project is optimized for instruction-based tasks like question-answering.

**Why Flan-T5-Base?**

* **Open Source**: Freely available for use.
* **Instruction-Tuned**: Fine-tuned on datasets with task instructions, making it effective for generating task-specific responses.
* **Powerful Yet Efficient**: While larger than Flan-T5-small, it balances accuracy and computational efficiency.

**How It Works:**

1. **Input**: A combination of the user query and retrieved context.
2. **Tokenization**: Converts text into numerical format for processing.
3. **Generation**:
   * Predicts one token at a time to construct a fluent response.
4. **Output**: Decodes the numerical output back into readable text.

**Example in This Project:**

* **Input**: *"Question: What are fundamental rights? Context: Fundamental rights are guaranteed by Part III of the Constitution..."*
* **Generated Output**: *"Fundamental rights include the right to equality, freedom, and protection against discrimination."*