**\*\* Workflow for Chatbot with FAISS and LangChain\*\***

**### \*\*Step 1: Crawled Data Storage\*\***

**- \*\*Input\*\*: Crawled data in JSON format (containing text and metadata).**

**- \*\*Process\*\*:**

**1. Load the JSON file containing the crawled data.**

**2. Save the text data into MongoDB for centralized storage and future reference.**

**3. Ensure deduplication by checking for existing entries.**

**- \*\*Output\*\*: MongoDB collection with all crawled text data.**

**---**

**### \*\*Step 2: Embedding Generation and FAISS Storage\*\***

**- \*\*Input\*\*: Text data retrieved from MongoDB.**

**- \*\*Process\*\*:**

**1. Fetch all text data from MongoDB.**

**2. Generate embeddings for each text document using SentenceTransformers.**

**3. Store the embeddings along with the original text in a FAISS index for efficient vector-based retrieval.**

**- \*\*Output\*\*: A FAISS index containing text embeddings and their corresponding documents.**

**---**

**### \*\*Step 3: Query Embedding and Context Retrieval\*\***

**- \*\*Input\*\*: User query in Urdu.**

**- \*\*Process\*\*:**

**1. Generate an embedding for the query using SentenceTransformers.**

**2. Perform similarity search in the FAISS index to identify the most relevant documents.**

**3. Retrieve the top k documents (based on cosine similarity) as context for the query.**

**- \*\*Output\*\*: The top k relevant documents as context for the query.**

**---**

**### \*\*Step 4: Response Generation with Flan-T5\*\***

**- \*\*Input\*\*:**

**1. User query.**

**2. Retrieved context (text from FAISS).**

**- \*\*Process\*\*:**

**1. Combine the user query and retrieved context.**

**2. Use Flan-T5 to generate a conversational response.**

**3. If no relevant documents are found, return a polite fallback response (e.g., apologizing for out-of-scope queries).**

**- \*\*Output\*\*: A contextually relevant or fallback response for the user.**

**---**

**### \*\*Step 5: Conversational Memory with LangChain\*\***

**- \*\*Input\*\*: User interactions across multiple turns.**

**- \*\*Process\*\*:**

**1. Use LangChain’s ConversationBufferMemory to store and manage the history of user queries and responses.**

**2. Append the conversation history to the current query and context.**

**3. Pass the updated context to Flan-T5 for generating a response that is aware of prior interactions.**

**- \*\*Output\*\*: Context-aware responses that maintain conversational continuity.**

**---**

**### \*\*Step 6: User Interaction and Deployment\*\***

**- \*\*Input\*\*: User interacts with the chatbot through a user interface (Gradio).**

**- \*\*Process\*\*:**

**1. The user submits a query via the UI.**

**2. The query is processed through the retrieval (FAISS) and generation (Flan-T5) pipeline.**

**3. The chatbot’s response is displayed back to the user.**

**- \*\*Output\*\*: A functional chatbot interface that provides accurate, conversational responses.**

**---**

**### \*\*Summary of Tools and Components\*\***

**1. \*\*MongoDB\*\*:**

**- Stores the raw crawled data.**

**- Acts as a fallback reference for metadata if needed.**

**2. \*\*FAISS\*\*:**

**- Handles vector-based similarity search for efficient document retrieval.**

**- Stores precomputed embeddings for all text data.**

**3. \*\*LangChain\*\*:**

**- Manages the RAG (Retrieval-Augmented Generation) pipeline.**

**- Provides memory for multi-turn conversations.**

**4. \*\*Flan-T5\*\*:**

**- Generates responses using the retrieved context and user query.**

**- Ensures responses are coherent and relevant to the query.**

**5. \*\*Gradio\*\*:**

**- Provides an interactive interface for user interaction with the chatbot.**

**---**

**### \*\*Example Flow\*\***

**1. \*\*User Query\*\*: "کریکٹر سرٹیفیکیٹ کے لئے کیا ضروری ہے؟"**

**2. \*\*Process\*\*:**

**- Generate embedding for the query.**

**- Search FAISS for the most similar documents.**

**- Retrieve relevant context (e.g., "کریکٹر سرٹیفیکیٹ حاصل کرنے کے لئے شناختی کارڈ اور درخواست فارم کی ضرورت ہے۔").**

**- Pass context and query to Flan-T5.**

**3. \*\*Output\*\*: "کریکٹر سرٹیفیکیٹ حاصل کرنے کے لئے شناختی کارڈ اور درخواست فارم کی ضرورت ہے۔"**

**4. \*\*Fallback\*\*: If no relevant documents are found, return: "معذرت، یہ سوال ہمارے ڈیٹا کی حد سے باہر ہے۔"**

**Summary of What We Have Done**

Here's a detailed overview of the steps and progress we have made:

**1. Workflow Implementation**

We defined and implemented a **Retrieval-Augmented Generation (RAG)** workflow for the chatbot:

1. **Data Storage**:
   * Crawled data was stored in MongoDB for centralized management.
   * Embeddings of the data were computed using SentenceTransformers and stored in MongoDB and FAISS for efficient vector-based similarity search.
2. **Context Retrieval**:
   * Queries are embedded and matched against the FAISS index to retrieve the most relevant context from the stored documents.
3. **Conversational Memory**:
   * LangChain’s ConversationBufferMemory was used to maintain multi-turn conversation history.
4. **Response Generation**:
   * Flan-T5 was integrated to generate responses based on the retrieved context and user query.

**2. Specific Steps and Challenges Addressed**

**A. MongoDB and FAISS Integration**

* **What We Did**:
  + Saved crawled data into MongoDB.
  + Precomputed embeddings using SentenceTransformers and saved them to FAISS for fast similarity searches.
* **Challenges**:
  + Ensured embeddings were stored in both MongoDB and FAISS in a JSON-compatible format.
  + Debugged issues with FAISS initialization and fixed parameter mismatches.

**B. Context Retrieval**

* **What We Did**:
  + Implemented a retrieve\_context function to fetch the top k similar documents using FAISS.
  + Debugged edge cases where FAISS returned no results, ensuring fallback mechanisms handle such cases.

**C. Response Generation with Flan-T5**

* **What We Did**:
  + Integrated Flan-T5 for Urdu-based response generation.
  + Refined the input prompt for better compatibility with Flan-T5.
  + Addressed decoding issues by adjusting generation parameters like num\_beams, temperature, and do\_sample.
* **Challenges**:
  + Debugged empty responses caused by improper input formatting.
  + Added a fallback response for queries outside the scope of the dataset.

**D. Conversational Memory**

* **What We Did**:
  + Used LangChain’s ConversationBufferMemory to store user and assistant messages.
  + Debugged issues with memory updates and ensured messages were appended correctly in the desired format.

**3. Code Adjustments**

* **Input Formatting**:
  + Transitioned from vague input prompts to structured, instruction-based prompts like:

vbnet

Copy code

Answer the following question based on the given context.

Context: ...

Question: ...

Answer:

* **Parameter Tuning**:
  + Increased num\_beams and adjusted temperature for diverse and accurate responses.
* **Fallback Mechanisms**:
  + Ensured meaningful fallback responses for cases with no relevant context or empty outputs from the model.

**4. Debugging and Testing**

* Added debug statements to monitor the flow of execution:
  + Verified query reception, context retrieval, input tokenization, and response generation.
  + Analyzed tokenized inputs and outputs for troubleshooting.

**5. Current Output**

* The chatbot is functioning but falls back to a default response for certain queries due to:
  1. Insufficient context quality.
  2. Possible limitations of Flan-T5 for Urdu-language tasks without fine-tuning.

**Next Steps**

1. **Improve Data Quality**:
   * Enhance the stored documents in MongoDB with more comprehensive and domain-specific data.
2. **Fine-Tune Flan-T5**:
   * Fine-tune the Flan-T5 model on Urdu-language datasets for better performance.
3. **Enhance Context Retrieval**:
   * Refine the FAISS index or consider hybrid retrieval approaches (e.g., combining FAISS with MongoDB queries).

This summary captures our progress and challenges addressed. Let me know if you need further refinements or clarifications!