RAG System Documentation

Kisan Call Centre Chatbot

1.Data Description

The dataset used in this system originates from data.gov.in and contains interaction logs from the Kisan Call Centre, where farmers call to ask agricultural queries and receive guidance from trained experts.

The dataset includes two primary columns:

- **questions:** Queries asked by farmers
- **answers:** Responses provided by agricultural experts

The initial dataset consists of 178,939 rows and 2 columns. However, for optimal performance and relevance, a robust data preprocessing strategy is applied to refine the dataset prior to ingestion into the RAG pipeline.

2.Data Preprocessing

2.1 Cleaning and Filtering

- **Handling Missing Values:** All rows with null or missing entries in the questions or answers fields were removed to maintain dataset consistency.
- **Filtering Non-informative Entries:** Rows containing generic phrases, administrative notes (e.g., “call transferred to expert”, “Test Call”), numeric-only content, or placeholders (e.g., “--”) were excluded.
- **Whitespace & Casing Normalization:** All text was lowercased and stripped of unnecessary spacing.

2.2 Semantic Deduplication

Traditional string-based deduplication approaches (e.g., `pandas.drop_duplicates()`) were found insufficient, as they failed to detect semantically similar but textually different entries. Therefore, semantic deduplication was implemented using sentence embeddings.

Method:

- Each question and answer were concatenated into a single field: `qa_combined`.
- Embeddings were generated using the `all-MiniLM-L6-v2` model via the sentence-transformers library.
- Cosine similarity was computed pairwise between all question-answer pairs.
- Any pair of rows with a similarity score exceeding 0.95 were deemed semantically redundant and one of them was removed.

Outcome:
This approach significantly reduced redundancy, resulting in a more informative and diverse dataset, suitable for semantic search-based retrieval tasks.

3.Chunking Strategy

Final Strategy: *Row-Based Q&A Chunking*
Each row in the dataset—representing a single question-answer pair—is treated as a standalone document. This method offered the best balance between granularity, semantic coherence, and alignment with the intended use case.

Why Not Semantic or Agentic Chunking?

- **Semantic Chunking:** Often merged answer parts with unrelated questions, diluting retrieval quality.
- **Agentic Chunking:** Risked overly large documents, sometimes exceeding model input limits.

Benefits of Row-based Chunking:

- Ensures high semantic fidelity by associating a query directly with its answer.
- Aligns with LangChain's CSVLoader, which naturally loads documents row-wise.
- Guarantees that retrieval focuses on complete, contextually relevant Q&A blocks.

4.Vector Embedding Generation

To support semantic retrieval, each document (Q&A pair) is transformed into a dense vector using a transformer model.

Embedding Engine:
Embeddings are generated using `nomic-embed-text:latest`, hosted locally via Ollama.

A custom embedding interface (`OllamaEmbeddings`) manages all interactions with the local embedding server.

Embedding Workflow:

- Each document (row) is passed to the embedding client.
- Embeddings are generated and batched efficiently.
- All document vectors are stored persistently in a local vector database.

Advantages:

- **Privacy:** No third-party APIs involved.
- **Speed:** Local inference using GPU or CPU.
- **Flexibility:** Future fine-tuned or custom models can be integrated easily.

5.Vector Database Storage

The vector database is responsible for storing document embeddings and enabling high-speed semantic similarity searches.

Technology:
ChromaDB is used as the vector store for its performance, persistence support, and compatibility with LangChain.

Storage Structure:

- **Page content:** Combined Q&A string
- **Embedding:** Generated via Ollama
- **Metadata:** Includes row number, and optionally crop/category

Persistent Storage:
ChromaDB is initialized with a specified directory to ensure that embeddings are saved between sessions. This avoids the need to recompute embeddings on system restart.

6.Semantic Retrieval Mechanism

At inference time, the RAG pipeline retrieves the most semantically relevant Q&A pairs to serve as context for the LLM.

Retrieval Workflow:

- **Query Embedding:**
  The user's question is embedded using the same `nomic-embed-text:latest` model for consistency.
- **Initial Retrieval (k=10):**
  ChromaDB performs a similarity search to retrieve the top 10 semantically similar chunks.
- **Cosine Reranking:**
  Retrieved documents are reranked using cosine similarity between the query embedding and the document embedding.
- **Final Context Selection (Top 5):**
  The top 5 most relevant documents post-reranking are selected for the final prompt context.
- **Filtering Threshold:**
  A similarity threshold (e.g., 0.5) is applied to discard irrelevant matches.

Benefits:

- Balances recall and precision by combining wide net retrieval and fine-grained reranking.
- Prevents overfitting to irrelevant phrases or answer fragments.
- Ensures that only semantically closest documents are passed to the LLM.

7.Prompt Construction & LLM Response Generation

Once the relevant context is retrieved, it is combined with the user's query and inserted into a structured prompt for the LLM to process.

LLM Engine:
The chosen model is `gemma3:27b`, hosted locally via Ollama. All LLM interaction is performed through an OpenAI-compatible API interface.

Prompt Design:

```
 You are an AI assistant for agricultural advisory...

Context:
{top_5_documents_combined}

Question:
{user_query}

Answer:
```

- Encourages factual synthesis from context
- Instructs the model to avoid hallucinations
- Includes a fallback response:
  "I don't have enough information to answer that."

Special Handling:
If the model outputs a `<think>...</think>` reasoning block, only the content after `</think>` is rendered in the UI.

8.FastAPI Web Application Interface

A user-friendly interface is served using FastAPI + Jinja2.

Features:

- Query submission form
- Clean UI with CSS-styled markdown rendering
- Responses displayed in bullet or paragraph format
- Logs both markdown and HTML output to terminal for debugging

UI Enhancements:

- HTML output rendered from markdown using the `markdown` module
- Responsive design
- Structured presentation of diseases, symptoms, and treatments