Comprehensive Documentation – AI QA Agent for Shared Documents

# 1. Setup & Run Instructions

This project builds an AI-powered QA Agent over sales and forecast data using a Retrieval-Augmented Generation (RAG) pipeline. The system is designed to run entirely on CPU, using free and open-source tools. Below are the setup and run steps.  
  
System Requirements:  
- OS: Windows/Linux/macOS  
- CPU only (no GPU required)  
- Python 3.10+  
  
Prerequisites:  
1. Install Ollama from https://ollama.ai/download  
 - Run Ollama server: ollama serve  
 - Pull model: ollama pull llama3.2:latest  
  
2. Install Python packages:  
 pip install streamlit faiss-cpu pandas numpy sentence-transformers requests  
  
3. Prepare Data:  
 - Ensure enriched merged.parquet file is ready.  
 - Build chunks, embeddings, and FAISS index.  
 - Outputs include chunks\_forecast.parquet, faiss\_forecast.index, meta\_forecast.parquet.  
  
4. Run the Application:  
 - Save the app code as app.py  
 - Run: streamlit run app.py  
 - Access in browser: http://localhost:8501

# 2. Step-wise Project Plan

1. Step 1: Data Preparation – Load Excel files (sales and forecast), clean and enrich them into merged.parquet.
2. Step 2: Chunking – Convert each sales row into plain-text chunks with metadata (product, site, month).
3. Step 2.1: Create month keys (e.g., JAN\_2025) from Date column for alignment.
4. Step 2.2: Build text chunks by flattening rows into human-readable text, including sales and forecast values.
5. Step 2.3: Vectorization – Use MiniLM model (sentence-transformers/all-MiniLM-L6-v2) for embeddings, stored in FAISS.
6. Step 2.4: Retriever – Encode user query into embeddings, search FAISS, and return top-k chunks with metadata.
7. Step 2.5: Prompt Builder – Construct a structured prompt with citations and pass to Ollama LLM.
8. Step 2.6: LLM Integration – Use Ollama with llama3.2:latest to generate answers from retrieved chunks.
9. Step 3: Streamlit UI – Build a user interface for natural language Q&A with answer display and citations.
10. Step 4: Documentation & Demo – Provide thorough documentation and a demonstration video.

# 3. System Architecture

The system follows a Retrieval-Augmented Generation (RAG) pipeline with distinct components working together to answer natural language queries on structured Excel data. Each part is described below.  
  
Data Preparation: Sales and forecast Excel files are read, cleaned, and merged into a unified dataset. Partial linkage was discovered – only 8 forecast products exist, so embeddings are limited to these.  
  
Chunking: Each row of merged data is transformed into a natural-language snippet. For example, a row with Product HISSPL15HPINV becomes:  
 'Product: HISSPL15HPINV; Site: BABU; Date: 2024-01-02; Month: JAN\_2025; Sales Qty: 120; Forecast: 100; Order Type: Accepted'.  
This chunking ensures structured tabular data is interpretable by an LLM.  
  
Vectorization: SentenceTransformer model all-MiniLM-L6-v2 is used to generate 384-dimensional embeddings. These embeddings capture semantic similarity, enabling FAISS vector database to retrieve relevant rows. Embeddings are normalized and indexed with FAISS IndexFlatIP for cosine similarity search.  
  
Retriever: At query time, the user’s question is embedded using MiniLM, searched against FAISS, and top-k similar chunks are returned with scores and metadata.  
  
Prompt Builder: Retrieved chunks are concatenated into a prompt with citation tags [CIT1], [CIT2] etc., ensuring traceability. The question is appended, and instructions force the model to respond only from context.  
  
Generator (LLM): Ollama serves llama3.2:latest locally. The prompt is sent via REST API to Ollama, which generates a natural language answer. If forecast is unavailable, the model is guided to answer 'Not available'.  
  
UI Layer: Streamlit app provides sidebar controls (top-k, temperature, token limits) and a main area for input and answers. Retrieved context is displayed as a table, and answers include citations. Debug expander shows the raw prompt.

# 4. Codebase Walkthrough

Core Functions:  
  
row\_to\_chunk(): Converts each DataFrame row into a chunk of text. Adds metadata like item\_code, site, month\_key, and order type. Aligns sales month with forecast columns.  
  
retrieve(query,k): Encodes the user’s query using MiniLM, searches the FAISS index, and returns top-k rows. Adds scores and citation tags.  
  
build\_prompt(question, rows): Constructs a structured prompt with system instructions, context (retrieved chunks with [CITx]), and the question. Ensures grounding and citations.  
  
ask\_ollama(prompt): Sends prompt to Ollama REST API. Parameters like temperature (0.2) and max tokens (250) are configurable. Returns the model’s text response.  
  
Streamlit UI (app.py): Provides the front-end. Sidebar allows adjusting retrieval parameters, question input box is in main area. Retrieved context is shown in a dataframe, answer is displayed with citations, and raw prompt is available in an expander.

# 5. Challenges & Solutions

1. Partial Linkage (Forecast ⊂ Sales): Only 8 products were forecasted. Solution: Restrict embeddings to these SKUs, return 'Not available' for others.  
  
2. Long Embedding Runtime: Full dataset embeddings took ~2 hours on CPU. Solution: Limit to forecast SKUs, deduplicate, increase batch size to 512.  
  
3. Month-Year Alignment: Forecast months lacked years. Solution: Renamed rightmost months as 2025, and earlier months backfilled as 2024, 2023.  
  
4. Ollama 500 Errors: Caused by oversized prompts. Solution: Limit top-k chunks, truncate text to 800 chars, cap prompt size.  
  
5. Parquet Type Errors: Object columns had mixed types. Solution: Cast all object columns to string before saving.