Project Documentation: AI-Powered Chatbot for Supplier Information

Prepared by: Diya Bhattacharjee

Date: September 3, 2025

# 1. Executive Summary

This document outlines a strategic approach to develop a highly scalable and responsive AI-powered chatbot capable of answering queries based on a dataset of 50 million supplier records. The primary challenge is to overcome the high latency associated with using large language models (LLMs) like Llama directly against a massive database.  
  
Our research indicates that the current direct-query method is not scalable for real-time applications. The proposed solution is to implement an industry-standard Retrieval-Augmented Generation (RAG) architecture. This approach separates the slow task of searching from the smart task of answering, which will dramatically reduce response times and ensure the system can handle future growth.

# 2. The Problem Statement

The core task is to build a chatbot that can provide accurate, conversational answers to questions about a database containing 50 million supplier records.  
  
Initial Challenge: The initial implementation, which involved sending user queries directly to a large language model (like Llama or Deepseek), resulted in unacceptably high response times. This is because LLMs are not designed to be high-speed search engines for large, structured databases.  
  
Key Requirement: The system must be scalable (able to handle growth in data and users) and provide low-latency (near-instant) responses to maintain a good user experience.

# 3. The Database Schema

The available data is highly structured and detailed, which is ideal for this project. The schema includes the following key fields, which provide rich context for each shipment:  
  
Parties Involved: supplier\_name, supplier\_address, importer\_name, importer\_address, carrier\_name, etc.  
  
Product Details: product\_desc, hs6\_name, quantity, weight, quantity\_unit, etc.  
  
Logistics: destination\_port, departure\_place, vessel\_name, etc.  
  
Timelines: estimated\_arrival\_date, actual\_arrival\_date, lag\_days  
  
(Images of the full schema are attached for reference in the project's source files.)

# 4. Research and Proposed Approach: Retrieval-Augmented Generation (RAG)

After researching the best practices for building scalable AI query systems, the recommended solution is the Retrieval-Augmented Generation (RAG) model.  
  
Why are we using this approach?  
  
The RAG model is the industry-standard solution because it solves the core problem of speed and scalability by dividing the work into two distinct phases:  
  
A fast, offline indexing phase that prepares the data for rapid searching.  
  
A smart, real-time query phase that finds relevant information first and then uses the LLM to generate a human-like answer.  
  
This is far superior to the direct-query approach because it plays to the strengths of different technologies: a fast search engine for searching, and a powerful language model for talking.  
  
Detailed Approach  
Phase 1: Building the "Smart Index" (Offline Preparation)  
  
This is the foundational step that enables the chatbot's speed. It is a one-time process that is run in the background.  
  
Create Summaries: We will write a script to iterate through all 50 million database records. For each record, it will create a clean, human-readable text summary based on the schema.  
  
Example Summary: "Supplier is [supplier\_name] from [supplier\_address]. They shipped [product\_desc] to [importer\_name] via the carrier [carrier\_name] on the vessel [vessel\_name]."  
  
Generate Embeddings: These 50 million text summaries will be fed into a small, fast AI model called an embedding model. This model will convert the meaning of each summary into a numerical format (a "vector").  
  
Store in a Vector Database: All 50 million of these vectors will be loaded into a specialized vector database. This type of database is designed for incredibly fast similarity searches, allowing us to find the most relevant summaries in milliseconds.  
  
Phase 2: The Real-Time Chatbot Workflow  
  
This is what happens instantly every time a user asks a question.  
  
User Asks a Question: The user types a query (e.g., "Find me suppliers from China who shipped 'automotive parts'").  
  
Retrieve Relevant Information:  
  
The system first converts the user's question into a vector using the same embedding model.  
  
It then searches the vector database to instantly find the top 5-10 most relevant supplier summaries. This is the "retrieval" step.  
  
Augment the Prompt:  
  
The system creates a new, detailed prompt for our large language model (Llama). This prompt includes the user's original question and the handful of relevant summaries it just retrieved.  
  
Generate the Answer:  
  
This new prompt is sent to Llama. Llama's job is now very easy: it reads the small amount of provided context and generates a high-quality, conversational answer. This is the "generation" step.  
  
By adopting this RAG approach, we can build a chatbot that is not only intelligent but also extremely fast and scalable, capable of handling our massive supplier dataset with ease.

# 5. Implementation Plan

This section outlines the technical steps and tools required to build a proof-of-concept for the proposed RAG system.  
  
Technology Stack:  
  
Programming Language: Python  
  
Core AI Libraries: Hugging Face transformers, sentence-transformers, torch  
  
Vector Database: faiss-gpu (for an efficient, local-first implementation)  
  
LLM for Generation: Llama (or a similar instruction-tuned model like Deepseek)  
  
API Framework: FastAPI (for creating a web-accessible service)  
  
Phase 1 Implementation: Building the Smart Index  
  
Data Ingestion: A Python script will connect to the source database. To handle the large volume of data, records will be processed in batches (e.g., 100,000 records at a time).  
  
Text Summarization: For each record in a batch, a function will generate the clean text summary as defined in the approach.  
  
Embedding Generation:  
  
Load a pre-trained sentence-transformer model (e.g., all-MiniLM-L6-v2).  
  
Use this model to encode the batch of text summaries into a list of vectors.  
  
Vector Storage:  
  
Initialize a FAISS index on a server with sufficient RAM and GPU memory.  
  
Add the generated vectors from each batch to the FAISS index.  
  
Once all records are processed, save the final index to disk for persistence.  
  
Phase 2 Implementation: Building the Chatbot API  
  
API Setup: A FastAPI application will be created with a single primary endpoint (e.g., /query). This endpoint will accept a user's question as a JSON object.  
  
Workflow within the API: Upon receiving a request at /query:  
  
Load Models: The API will load the pre-trained embedding model and the saved FAISS index into memory.  
  
Encode Query: The incoming user question will be converted into a vector using the embedding model.  
  
Retrieve Context: The API will search the FAISS index with the query vector to get the indices of the top 'k' (e.g., 5) most relevant supplier records.  
  
Fetch Summaries: The original text summaries corresponding to these indices will be retrieved.  
  
Augment Prompt: A detailed prompt will be constructed, combining the original question with the retrieved summaries.  
  
Generate Response: This prompt will be sent to the Llama/Deepseek model to generate the final, human-readable answer.  
  
Return Answer: The generated answer will be returned to the user as a JSON response.  
  
Deployment: The FastAPI application will be deployed on a server (ideally with a GPU to run the LLM) to make it accessible to front-end applications or other internal services.