Project Report: ERP Chatbot (RAG + Generative AI) Module

Ahmed Ayman
November 2025

Abstract

This report details the implementation of an ERP Chatbot module utilizing a **Retrieval-Augmented Generation (RAG)** architecture. The project successfully integrates multi-document ingestion, vector embedding storage via MongoDB Atlas Vector Search, and a sophisticated retrieval and generation pipeline using Google's **Gemini LLM**. A key feature is the ability to query across multiple ERP document collections (Invoices, Purchase Orders, and Approvals), providing grounded and contextually relevant responses. The architecture is designed for future extensibility, supporting features like confidence scoring and structured workflow integrations.

1 Introduction and Project Objective

The primary objective of this module was to design and implement a scalable, intelligent query solution capable of providing accurate, grounded answers based on critical business documents from an Enterprise Resource Planning (ERP) system. The solution uses a RAG pipeline to achieve information retrieval and high-fidelity text generation, effectively turning unstructured ERP documents into a searchable knowledge base.

2 System Architecture and Core Components

The solution comprises a Flask API backend for core RAG functionality and a Streamlit frontend for user interaction, utilizing a modern, containerized deployment strategy.

2.1 Technology Stack

- **Backend Framework:** Python (Flask) for routing and orchestration.
- **Database and Vector Store:** MongoDB Atlas (for document storage and Vector Search indexing).
- **Vector Embeddings:** `SentenceTransformer` (`all-MiniLM-L6-v2`) for generating query and document embeddings.
- **Generative AI:** `ChatGoogleGenerativeAI` (Gemini 2.5 Flash) via LangChain for contextual generation.
- **Deployment:** Containerized using Docker (Gunicorn) to ensure environment consistency.

2.2 Data Ingestion and Vector Store

The system is designed to provide answers grounded in Enterprise Resource Planning (ERP) documentation, categorized into multiple data streams.

1. **Document Ingestion:** Supports and manages data from three distinct ERP collections: `invoices`, `purchase_orders`, and `approvals`.
2. **Preprocessing & Embedding:** Each document's content is encoded into high-dimensional vectors.
3. **Vector Storage:** Embeddings are stored and indexed within the respective MongoDB collections using MongoDB Atlas Vector Search, enabling efficient semantic retrieval.

3 Retrieval-Augmented Generation (RAG) Pipeline

The RAG pipeline is the central intelligence mechanism, ensuring the LLM's answers are based on factual data retrieved directly from the ERP documents.

3.1 The Query Process

1. **Query Embedding:** The user's natural language question is immediately converted into a vector using the `all-MiniLM-L6-v2` model.
2. **Multi-Collection Vector Search (Retrieval):** The backend executes simultaneous MongoDB `$vectorSearch` aggregation pipelines across all specified ERP collections. This ensures maximum recall across all document types.
3. **Context Aggregation:** The top N most relevant documents from the combined search results are compiled into a unified context block. Each document is labeled with its collection type (e.g., `[INVOICE]`), serving as a source citation.
4. **Generation (LLM Invocation):** A final, grounded prompt is constructed, instructing the `gemini-2.5-flash` model to answer *only* based on the provided context, resulting in a factual and traceable response.

4 Addressing Module Requirements and Extensibility

The foundational architecture is designed to support both current requirements and future advanced features.

4.1 Guardrails Implementation

The current RAG implementation directly addresses key guardrail requirements:

- **Citation of Sources:** Implemented by labeling each context chunk with its originating collection (`[INVOICE]`), providing a clear path for traceability.
- **Context Retention:** The API structure facilitates the integration of an external memory store to retain chat history and context across user sessions.
- **Confidence Scoring (Planned):** The architecture supports augmenting the RAG output with calculated relevance scores from the MongoDB vector search to provide a confidence metric.

4.2 Structured Workflows

The architecture is extensible to integrate structured workflows (e.g., automated leave requests) by enabling the LLM to perform:

- **Intent Recognition:** Identifying user requests that map to specific ERP actions rather than just information retrieval.
- **Function Triggering:** Calling specific, non-RAG backend functions or external ERP action APIs based on the recognized intent.

5 Deployment and Access

The application components have been successfully deployed to the Hugging Face Spaces platform, providing public access to both the frontend interface and the backend API service.

- **Streamlit Frontend Application:** Accessible via the link below, providing the user interface for natural language querying of ERP documents: https://huggingface.co/spaces/ahmed-ayman/erp-chatbot
- **Flask Backend API:** The underlying processing service is deployed separately for scalability and decoupled access: https://huggingface.co/spaces/ahmed-ayman/erp_chatbot_api

6 Conclusion

The ERP Chatbot module delivers a robust, multi-document RAG system utilizing modern embedding models and Google's Gemini LLM. The design provides a scalable foundation for accurate, grounded Q&A over ERP data and is ready for the integration of advanced features like workflow automation and improved guardrails.