**DESIGN AND IMPLEMENTATION OF A RETRIEVAL – AUGMENTED GENERATION (RAG) BASED CHATBOT**

Team Members: Vinay Kumar Mannava, Siri Ravalli

**TABLE OF CONTENTS**

**1. Objective ------------------------------------------------------------3**

**2. Problem Statement ------------------------------------------------3**

**3. Solution Overview------------------------------------------------- 4**

**4. Functional Workflow--------------------------------------------- 4**

**5.** **Architecture Diagram---------------------------------------------5**

**6.** **Component Exploration------------------------------------------5**

* Input Interface----------------------------------------------------**5**
* Prompt Creation------------------------------------------------- **5**
* Embedding Model-----------------------------------------------**6**
* Chunking Strategy-----------------------------------------------**6**
* Vector Store------------------------------------------------------**6**
* Retriever----------------------------------------------------------**7**
* Language Model (LLM)----------------------------------------**7**
* Response Generation & Display-------------------------------**8**

**7.** **Conclusion-----------------------------------------------------------8**

**8.** **References-----------------------------------------------------------8**

**Objective**

The goal of this project is to build a smart chatbot that helps consultants at SMA TECH LLC easily find and prepare for interview questions based on past interviews. The chatbot searches through documents stored in Odoo and gives relevant questions and answers based on what the user asks.

This saves time, avoids the need to manually go through multiple files, and makes it easier for consultants to get ready for interviews by showing accurate and useful information quickly.

**Problem Statement**

SMA TECH LLC is a technology consulting company that manages a wide range of consultants skilled in domains such as .NET Full Stack Development, AI/ML Engineering, Data Engineering, and Python Development. These consultants are actively engaged in client interviews across various industries. As part of the internal knowledge-sharing process, each consultant is required to document their interview experiences—this includes the job description (JD), the number of interview rounds, technical questions asked, topics covered, and the result of each round. These documents are uploaded and organized in the Odoo platform, forming a valuable interview knowledge base over time. However, as the volume of consultants and client interactions increases, the number of interview records also grows rapidly, leading to several challenges:

Consultants often spend excessive time navigating through dozens or even hundreds of documents to find relevant interview questions and answers for a specific role or client.

Due to varied formats and writing styles, the information in these documents is not always uniformly structured, making it harder to extract meaningful insights quickly.

Questions on the same topic (e.g., API integration in .NET or model evaluation in ML) are scattered across multiple documents, which makes it hard to see recurring patterns or focus areas.

Many valuable insights from past interviews remain buried in documents, limiting their reuse and reducing the overall preparedness of consultants.

These challenges make it difficult for both new and experienced consultants to prepare effectively for upcoming interviews, reducing efficiency and increasing the likelihood of missed opportunities.

**SOLUTION OVERVIEW**

To address the inefficiencies in accessing historical interview documentation, SMA TECH LLC AI/ML Engineers is implementing a **Retrieval-Augmented Generation (RAG)-based chatbot**. This solution combines semantic search with generative AI to deliver accurate and context-aware answers to consultants' questions based on past interview data stored in the Odoo system.

The architecture includes the following major components:

* User Interface: A chatbot interface where consultants can type queries in natural language.
* Embedding Layer: User queries and interview documents are transformed into vector embeddings using pre-trained models.
* Vector Database (Retriever): The system stores embedded documents in a vector store (e.g., FAISS or Pinecone) and retrieves the most relevant content using similarity search.
* Prompt Generator: Retrieved documents are injected into a prompt structure to provide background context to the language model.
* Language Model (LLM): A pre-trained generative model processes the prompt and generates accurate, grounded responses.
* Response Delivery: The chatbot delivers the final response to the user in a clean, readable format, optionally linking to relevant document sources.

**Functional Workflow**

* The consultant enters a query such as “Show me round2 .NET Full Stack interview questions.
* The chatbot converts the query into an embedding.
* This embedding is used to retrieve the most relevant interview documents from the vector store.
* The top-k document snippets are combined with the query and sent as input to the LLM.
* The LLM generates a natural language answer based on the provided context.
* The final output is displayed to the consultant via the chatbot interface.

**ARCHITECTURE DIAGRAM**

A diagram of a chatbot

AI-generated content may be incorrect.

**COMPONENT EXPLORATION**

**Input Interface:** We implemented a **Streamlit-based web UI** as the chatbot’s input layer. It allows users to upload documents and submit queries in natural language. Streamlit was chosen for its simplicity, quick deployment, and ability to handle interactive responses along with document previews. The interface also displays source content alongside the generated responses for transparency.

**Prompt Creation:**

Several **prompting techniques** were explored, including:

* **Zero-shot prompting** – Simple, no-context prompt
* **Few-shot prompting** – Adds examples for better grounding
* **Chain-of-Thought (CoT)** prompting – Adds reasoning steps
* **Context-injected prompting** – Injects retrieved document chunks into the prompt to ground the response

**Final choice:** We used **context-injected prompting** via StuffDocumentsChain from Lang Chain. This technique ensures that the chatbot’s answers are grounded in the uploaded documents, significantly reducing hallucinations and improving relevance.

**Embedding Model:**

OpenAI text-embedding-ada-002: Fast and effective but requires external API calls and cost per token.

all-MiniLM-L6-v2 (Sentence Transformers): Free and local, but lower semantic understanding.

Google’s models/embedding-001: Optimized for Gemini-compatible architecture with improved contextual embeddings.

**Final Choice:** Google models/embedding-001  
We selected Google’s embedding model for its deep semantic matching, lower noise, and compatibility with Gemini-based LLM stacks. It showed improved precision when retrieving technical content such as interview questions.

**Chunking Strategy:**

**Fixed Window Chunking**: Easy to implement but may break context mid-sentence or mid-concept.

**SentenceSplitter**: Preserves semantics but results in uneven chunk sizes.

**RecursiveCharacterTextSplitter**: Breaks content at paragraph > sentence > word level, maintaining a good balance between size and coherence.

Final Choice: RecursiveCharacterTextSplitter  
This approach offers structural awareness and maintains semantic flow across chunks, essential for accurate document retrieval and grounding the LLM's response. The overlap prevents knowledge loss at chunk boundaries.

**Vector Store:**

Pinecone: Managed, scalable, but requires setup and incurs cost.

ChromaDB: Easy to use, good for local projects, lacks scalability.

FAISS (Facebook AI Similarity Search): Fast, open-source, highly customizable for both RAM and disk-backed retrieval.

Final Choice: FAISS  
We selected FAISS for its high-speed retrieval, offline compatibility, and easy local deployment. It's suitable for our in-house document base and integrates seamlessly with LangChain for plug-and-play vector indexing.

**Retriever:**

Explored Techniques:

* BM25 (Sparse Search): Keyword-based, not ideal for semantic understanding.
* Dense Vector Retrieval with Cosine Similarity: Uses embeddings to find semantically similar chunks.
* Hybrid Search (BM25 + Embeddings): Powerful but requires tuning and introduces additional complexity.

Final Choice: Dense Vector Search with Cosine Similarity  
Since our documents are semi-structured and domain-specific, dense retrieval using cosine similarity over vector embeddings delivered the best trade-off between relevance and speed. It's also natively supported by FAISS.

**Language Model (LLM):**

OpenAI GPT-3.5 Turbo: Accurate, cost-efficient, great instruction following.

Claude 1/2 (Anthropic): High-quality responses, but latency and access limitations.

Google Gemini Pro: Competitive performance, but limited documentation at time of use.

Open source (LLaMA2, Mistral): Free but complex to serve and scale securely.

Final Choice: OpenAI GPT-3.5 Turbo  
GPT-3.5 Turbo was chosen for its low latency, high instruction-following accuracy, robustness in document-grounded Q&A, and reasonable cost per token, making it ideal for an enterprise-level RAG chatbot.

**Response Generation & Display:**

Once the response is generated by the LLM, it is rendered in the Streamlit chatbot UI, optionally accompanied by source context chunks. Highlighted with metadata such as client name, role, round number, or technology tagThis provides consultants with both a direct answer and traceability, improving trust and comprehension.

**Conclusion**

In this project, we built a smart chatbot using Retrieval-Augmented Generation (RAG) to help consultants at SMA TECH LLC easily find interview questions from past experiences. The chatbot uses a mix of document search and AI to give accurate, helpful answers based on what users ask.By using techniques like context-injected prompting, the chatbot gives responses that are based on real documents instead of making things up. This saves time, improves preparation, and makes it easier for consultants to get ready for interviews.This solution helps organize knowledge better and makes it easy to reuse across different roles like .NET Developer, Data Engineer, and AI/ML Engineer

**References**

* <https://docs.langchain.com>
* <https://github.com/facebookresearch/faiss>
* <https://platform.openai.com>
* <https://developers.generativeai.google>
* https://streamlit.io