Project Report: Hypothetical Question Answering Chatbot using Vertex AI

---

1. Introduction

This project implements a Hypothetical Question Answering Chatbot based on the Forex Market Screener document. The chatbot answers user questions by identifying the most relevant passage from the document using vector similarity search and then generating a fluent response using a large language model (LLM).

The system is modeled on the Retrieval-Augmented Generation (RAG) framework and closely aligns with the Hypothetical Question Retriever reference architecture described by GraphRAG.

---

2. Technologies Used

- **Google Cloud Storage (GCS)**: To store raw PDF documents.
- **Vertex AI Embeddings API**: To convert user queries into dense embeddings.
- **Vertex AI Matching Engine**: To store and query chunks using Approximate Nearest Neighbor (ANN) search.
- **Vertex AI Generative Model (Gemini 2.5 Pro)**: To generate natural language responses.
- **Python with Vertex AI SDK**: To integrate all components programmatically.
- **PyMuPDF (fitz)**: For PDF text extraction.
- **RecursiveCharacterTextSplitter (LangChain)**: For layout-aware document chunking.
- **Streamlit**: For front-end chatbot UI.

---

3. Architecture Overview

1. **Document Ingestion**

2. PDF documents are uploaded to **Google Cloud Storage**.

3. Text is extracted using **PyMuPDF** and split using a **Recursive Character Text Splitter** (chunk size 512, overlap 25).

4. **Vector Indexing**

5. Each chunk is embedded using `text-embedding-005`.

6. Embeddings are uploaded to **Vertex AI Matching Engine** with associated metadata.

7. **Query Flow**

8. User input is embedded using `TextEmbeddingModel`.

9. A **nearest-neighbor query** is sent to the Matching Engine.
10. The most relevant chunk ID is retrieved.
11. **Answer Generation**
12. A prompt is constructed using the retrieved chunk ID and user question.
13. This is passed to the **Gemini 2.5 Pro** model to generate a contextual answer.

---

4. Code Flow Summary

Step 1: Embed the User Query

```
embedding_model = TextEmbeddingModel.from_pretrained("text-embedding-005")
embedding_response =
embedding_model.get_embeddings([TextEmbeddingInput(text=user_query)])
query_vector = embedding_response[0].values
```

Step 2: Initialize Vertex AI & Setup Client

```
aiplatform.init(project="hypo-question-retriever", location="us-central1")
client_options = {"api_endpoint": "1391263978.us-
central1-80137264349.vdb.vertexai.goog"}
vector_search_client =
aiplatform_v1.MatchServiceClient(client_options=client_options)
```

Step 3: Prepare and Send Nearest Neighbor Request

```
datapoint = aiplatform_v1.IndexDatapoint(feature_vector=query_vector)
query = aiplatform_v1.FindNeighborsRequest.Query(datapoint=datapoint,
neighbor_count=1)
request = aiplatform_v1.FindNeighborsRequest(
    index_endpoint="projects/.../indexEndpoints/...",
    deployed_index_id="hypo_question_index_deployment",
    queries=[query],
    return_full_datapoint=False
)
response = vector_search_client.find_neighbors(request)
```

Step 4: Generate Answer using Gemini

```
nearest_chunk_id =
response.nearest_neighbors[0].neighbors[0].datapoint.datapoint_id
generative_model = GenerativeModel("gemini-2.5-pro")
prompt = f"Answer this question based on chunk {nearest_chunk_id}: {user_query}"
final_answer = generative_model.generate_content(prompt)
```

---

5. Google Cloud Services Enabled

Service	Purpose
Vertex AI API	For embeddings, generative model, and project init
Vertex AI Matching Engine API	For vector indexing and ANN search
Cloud Storage API	To store documents for processing
Cloud Resource Manager API	For project-level metadata and access management

---

6. Benefits of This Approach

- **Scalable RAG**: Retrieves only relevant chunks, reducing LLM token usage.
- **Semantic Search**: Embedding-based retrieval outperforms keyword search.
- **Serverless & Managed**: All infrastructure is managed by Google Cloud.
- **Extendable**: Can easily support multiple documents, chunk metadata, and fine-tuned prompts.

---

7. Screenshots and Diagram

![GraphRAG Reference Screenshot](sandbox:/mnt/data/Prototyping_ GraphRAG Hypothetical Question Retrieve.pdf)

This figure (from the GraphRAG reference PDF) summarizes:

- The RAG pipeline components used
- Tools and technologies involved
- Behaviors like greetings, summary, fallback response

---

8. Conclusion

This chatbot implements a robust Retrieval-Augmented Generation architecture using fully managed services on Google Cloud. By integrating document chunking, semantic vector search, and large language models, it demonstrates an enterprise-grade solution for document-aware Q&A bots.

Future improvements may include chunk content resolution, document summaries, or dynamic fallback strategies based on chunk confidence.