OPTIMIZING QA RESPONSES: INSIGHTS INTO BERT AND RAG-BASED MODELS

1. Background and Objectives

Background:

Initially, started with a basic BERT model, deepset/bert-base-cased-squad2, to handle question-answering (QA) tasks using documents in various formats, including DOCX, PDF, and Excel. This approach provided a simple solution for QA but was limited in handling complex queries and large datasets.

Objective:

To enhance the QA system by integrating a more sophisticated Retrieval-Augmented Generation (RAG) model. The goal is to improve the system's ability to handle complex queries by combining document retrieval with generative language models, thus providing more accurate and contextually relevant answers.

2. Design Decisions and Approach

A. Data Processing and Integration:

1. **Document Processing:**
    - **Initial Approach:**
        - Used libraries like pandas, docx, and pdfminer to extract text from Excel, DOCX, and PDF files.
    - **Enhanced Approach:**
        - Switched to more specialized libraries (PyPDF2, pptx) for better handling of various document formats.
        - Used langchain components for more advanced document handling and embedding.
2. **Switch to RAG Model:**
    - **Initial Model:**
        - Basic BERT model (deepset/bert-base-cased-squad2) was used for QA.
    - **RAG Model:**
        - Integrated Retrieval-Augmented Generation (RAG) to improve performance. The RAG model combines document retrieval with generative language modeling to provide more accurate answers.
        - Utilized FAISS for efficient similarity search and document retrieval.

        - Employed HuggingFaceEmbeddings for high-quality embeddings and Cohere for generating answers.

B. RAG Model Integration:

1. **Document Embedding and Indexing:**
    - **Embedding:**
        - Used HuggingFaceEmbeddings with the sentence-transformers/all-MiniLM-L6-v2 model to convert text chunks into embeddings.
    - **Indexing:**
        - Indexed the embedded text chunks using FAISS for fast similarity search.
2. **Retriever Setup:**
    - **Retrieval:**
        - Set up a retriever using FAISS to retrieve the most relevant documents based on similarity scores.
3. **Prompt Template:**
    - **Design:**
        - Created a prompt template to structure the interaction with the language model, ensuring that it answers the question based on the provided context.
        - Prompt Template

          Answer the question as precise as possible using the provided context. If the answer is not contained in the context, say "answer not available in context" \n\n

          Context: \n {context}?\n

          Question: \n {question} \n

          Answer:
4. **Answer Generation:**
    - **Chain Construction:**
        - Constructed a RAG chain using Cohere for generating answers.
        - Utilized format_docs function to concatenate document contents into a single string.
        - Defined generate_answer function to invoke the RAG chain with the question and context.

C. API Integration with FastAPI:

1. **API Design:**
    - **Endpoints:**
        - Created endpoints to handle QA requests and evaluate latency.
        - POST /answer/: Takes a question, retrieves relevant documents, generates an answer, and returns the result along with latency.
        - GET /evaluate_latency/: Evaluates the average latency of answering a set of sample questions from the SQuAD dataset.
2. **Error Handling:**
    - Implemented error handling to manage exceptions and provide informative error messages.
3. **Latency Measurement:**
    - Measured latency for each QA request to assess performance and efficiency.

3.Technical Details

1. **Libraries and Frameworks:**
    - **HuggingFaceEmbeddings**: For creating embeddings of text.
    - **FAISS**: For efficient similarity search.
    - **Cohere**: For generating answers using a generative model.
    - **langchain**: For chaining components and handling prompts and outputs.
    - **FastAPI**: For building the web API to interact with the model.
2. **Code Implementation:**
    - **Document Processing:**
        - Handled various document types and extracted text.
    - **RAG Chain:**
        - Constructed a pipeline combining retrieval and generative models.
    - **API Endpoint Implementation:**
        - Exposed endpoints for answering questions and evaluating performance.

4.Results and Performance Evaluation

1. Performance Metrics:

- **Latency:** Measured the time taken to generate answers.

- **Answer Quality:** Assessed the relevance and accuracy of generated answers.

2. Challenges and Solutions:

- **Handling Large Datasets:** Optimized retrieval and embedding processes to manage large volumes of text.
- **Integration Issues:** Ensured smooth integration of various components (e.g., embeddings, retriever, generator).

3. Future Improvements:

- **Model Enhancement:** Explore other generative models or fine-tune the existing ones for better performance.
- **Scalability:** Optimize the system for handling larger datasets and more complex queries.

5.Showcase: Validating QnA Responses, Preventing Hallucination, and Metrics

1. Validation of QnA Responses:

A. Response Validation:

- **Manual Validation:**
  - Randomly sampled responses from the QA system and reviewed them manually for accuracy and relevance.
- **Cross-Validation:**
  - Compared responses against a held-out validation set (e.g., SQuAD dataset) to ensure consistency and correctness.
- **Benchmarking:**
  - Compared model outputs against known answers and evaluated their alignment with the expected results.

B. Automated Validation:

- **Metrics-Based Evaluation:**
  - Utilized similarity metrics like cosine similarity and embedding similarity to gauge how close the generated responses are to known answers.
- **Custom Evaluation Scripts:**
  - Developed scripts to automatically evaluate response quality based on predefined criteria and similarity thresholds.

2. Preventing Hallucination:

A. Contextual Awareness:

- **Enhanced Retrieval:**
    - Used FAISS to retrieve the most relevant documents, ensuring that the generative model works with high-quality, contextually relevant information.
- **Prompt Engineering:**
    - Designed prompts to explicitly instruct the model to base answers on the provided context and to indicate when the answer is not available.

B. Generative Model Controls:

- **Temperature Setting:**
    - Adjusted the temperature parameter in the generative model (Cohere) to balance creativity and accuracy. Lower temperatures help reduce the likelihood of hallucination by making the model's responses more deterministic.
- **Answer Constraints:**
    - Incorporated constraints in the prompt to limit the model's scope to the context and to directly address scenarios where information is missing.

6.Summary and Recommendations:

- **Performance Insights:**
    - The initial BERT model provided faster responses with reasonable accuracy and F1 scores but lacked the ability to handle complex queries and larger contexts.
    - The RAG-based model offers a more sophisticated approach by combining retrieval with generation, though it currently suffers from higher latency and lower similarity scores.
- **Recommendations for Improvement:**
    - **Optimization:** Further optimize the retrieval and generative processes to reduce latency.
    - **Enhancement:** Fine-tune the generative model or explore alternative models to improve response relevance and similarity.
    - **Evaluation:** Continuously evaluate the system using additional metrics and real-world queries to ensure robustness and reliability.

7. Conclusion

This approach combines advanced document processing techniques with a powerful RAG model to enhance QA performance. The integration of FastAPI allows for efficient

deployment and testing of the model, providing a robust solution for real-world QA applications.