**Optimizing the RAG Model in finalapp.py**

This document outlines two innovative techniques to enhance the performance and scalability of the Retrieval-Augmented Generation (RAG) model implemented in the finalapp.py script. The proposed optimizations focus on improving text splitting and vector storage efficiency.

**1. Dynamic Chunking with Semantic Awareness**

**Problem**

The current implementation employs RecursiveCharacterTextSplitter to divide documents into chunks of fixed sizes (700 characters with 50-character overlap). While functional, this method:

* Splits text arbitrarily, potentially breaking sentences or concepts.
* Reduces the semantic cohesion of chunks, affecting retrieval quality.

**Proposed Solution**

Implement **semantic chunking** using a pretrained language model or sentence boundary detection. By splitting documents at meaningful points, such as sentence or paragraph boundaries:

* Each chunk retains contextual integrity.
* Retrieval accuracy improves as the model processes cohesive chunks.

**Implementation**

Replace RecursiveCharacterTextSplitter with a semantic-based text splitter:

from langchain.text\_splitter import SemanticTextSplitter

st.session\_state.text\_splitter = SemanticTextSplitter(max\_chunk\_size=700)

st.session\_state.final\_documents = st.session\_state.text\_splitter.split\_documents(st.session\_state.docs[:30])

**Benefits**

* Enhanced relevance in document retrieval.
* Improved user experience through precise and contextually rich answers.

**2. Efficient Index Updating with Incremental Vector Store**

**Problem**

The script recreates the FAISS vector store each time vector\_embedding() is executed. This approach:

* Is computationally expensive, especially for large datasets.
* Redundantly processes unchanged documents.

**Proposed Solution**

Utilize **incremental updates** to append new embeddings to the existing FAISS index. By processing only new documents:

* Resource usage is minimized.
* The application scales effectively with growing datasets.

**Implementation**

Modify the FAISS handling logic to support incremental updates:

if "vectors" not in st.session\_state:

st.session\_state.vectors = FAISS.from\_documents(st.session\_state.final\_documents, st.session\_state.embeddings)

else:

new\_docs = st.session\_state.loader.load\_new\_docs() # Hypothetical function for new documents

new\_chunks = st.session\_state.text\_splitter.split\_documents(new\_docs)

st.session\_state.vectors.add\_documents(new\_chunks)

**Benefits**

* Faster embedding updates for newly added documents.
* Avoids redundant computations, ensuring efficient resource utilization.

**Conclusion**

By adopting these techniques:

1. **Dynamic Chunking with Semantic Awareness** improves retrieval relevance by preserving contextual integrity in document chunks.
2. **Efficient Index Updating** reduces computational overhead and enhances scalability.

These optimizations ensure a more robust and scalable RAG pipeline, meeting performance demands for real-world applications.