RAG QnA Bot on Social media Data

Implementation

- build a streaming pipeline that ingests instagram posts and Pinterest data into a vector DB.
- clean, chunk, and embed the captions Data in Csv.
- build a retrieval client to query answers based on the captions
- use a rerank pattern to improve retrieval accuracy
- visualize content retrieved for a given query in a 2D plot.

1. Data Ingestion

I. Data Cleaning

Preprocessing: Before chunking, clean the text to remove unnecessary characters (e.g., emojis, URLs if irrelevant for retrieval).

The following features of the above post are not compatible with embedding models. We'll need to find some way of handling them in our **preprocessing step:** emojis bold, italic text other non-ASCII characters URLs content that exceeds the context window limit of the embedding model

II. Chunking Strategy

Sentence-Based Chunking would allow each chunk to maintain its semantic meaning, making it easier for the retriever to match queries accurately.

How It Works: Split the captions into chunks of a fixed number of tokens (e.g., 100 tokens).
Pros: Ensures that chunks are of a manageable size for the retriever, preventing token overflow.
Cons: May split sentences or ideas inappropriately, potentially reducing the quality of retrieval.
When to Use: Useful when captions vary greatly in length, and you need a more consistent chunk size.

III. Embeddings

I would recommend starting with **Sentence-BERT (SBERT)** because it offers:

- Superior performance on sentence-level tasks like semantic search.
- A balance between computational efficiency and embedding quality.

If you require a more general-purpose solution or if SBERT doesn't meet your needs, **OpenAI's embeddings** are also an excellent choice.

OpenAI Embeddings:

- OpenAI's models, especially the ones based on GPT-3, are general-purpose and have shown excellent performance across a wide range of tasks.
- strong performance on general language tasks.
- Versatile across different types of data and tasks.

Vector Embedded Indexing for retrieval and efficiently storage , enabling fast similarity searches

IV. Database or Retriever:

Vector Database,Based on the data, **start with a vector database** like Pinecone, Weaviate, or Milvus for embedding-based similarity search. This will likely be the most immediate need for your use case.

2. Data Retrieval

Auto-Merging Retriever

- If bot frequently needs to combine information from multiple captions to answer complex queries, this method could be advantageous. However, it's more complex.
- Use Case: Ideal for scenarios where comprehensive responses are more important than simple, quick answers.

Consideration:Sentence-Window Retrieval / Small-to-Large Chunkin

For Maximum Accuracy: Ensemble Retrieval and Re-Ranking

- If we need to ensure the highest accuracy across a variety of queries and are willing to invest in the added complexity, this method could be ideal.

Preferred Retrieval Steps:

- **Prototype with Sentence-Window Retrieval**: Start by implementing a prototype using the Sentence-Window Retrieval method to see how well it handles the data and query requirements.
- **Evaluate and Expand**: If we find that our queries require combining information from multiple captions, consider incorporating the Auto-Merging Retriever. If accuracy across diverse queries becomes a priority, explore Ensemble Retrieval.