Task 2 – Optimizing the RAG Model for Business QA Bot

Harsh Indoria

Note : PDF is AI augmented not AI generated. Answers are based on my ideas and formatted by ChatGPT for proper reading.

🧠 Introduction

In Task 1, a Retrieval-Augmented Generation (RAG) system was developed to serve as a question-answering (QA) chatbot for Bay Wheels, a bike-sharing service operated by Lyft. The pipeline combined a vector database (Pinecone) with a language model (initially OpenAI, later switched to Ollama with Mistral) to provide contextual answers based on document retrieval.

While this implementation successfully handles general queries, it can be further optimized for **enterprise usage**, especially in terms of **accuracy, latency, and modularity**. This document proposes two practical and scalable optimization techniques rooted in a data engineering and system design perspective.

⚙️ Optimization Techniques

🔷 Technique 1: Department-wise Most-Asked Q/A Embedding

📌 Overview

One of the most effective ways to improve retrieval precision is by embedding curated question-answer pairs that directly reflect real-world business needs. This technique involves **collecting frequently asked questions (FAQs)** from different departments (e.g., Customer Support, Marketing, Technical Operations), documenting their accurate answers, and embedding them in the vector database as department-specific documents.

🧰 Implementation Steps

1. **Collect Real FAQs**
    - Use internal sources such as chat logs, support tickets, email threads, and onboarding sessions.
    - Ask department heads to provide their top 10–20 most common queries.
2. **Organize by Department**
    - Create structured documents or PDFs such as:
        - `support_faqs.txt`
        - `marketing_faqs.txt`
        - `technical_faqs.txt`
3. **Embed into VectorDB**
    - Use a text loader and embedding model (e.g., HuggingFace) to vectorize and store these documents.
4. **Enhance Retrieval**
    - The RAG model retrieves from a **targeted, high-quality source** when a matching query is detected.

✅ Benefits

- Reduces hallucinations from generic model answers.
- Delivers faster and more accurate responses to business users.
- Easy to update — simply add or revise the source file for each department.

🔷 Technique 2: Department-specific Vector Databases with Rule-Based Routing

📌 Overview

In real-world business settings, different teams often operate semi-independently. Questions asked by users are frequently tied to a particular department (e.g., “What is the subscription price?” vs. “How to fix a bike lock error?”). Instead of using a single, monolithic vector store, this technique proposes building **multiple small vector databases**, one for each department, and routing the query to the relevant store using **simple manual rules or classification logic**.

🧰 Implementation Steps

1. **Segment Documents by Department**
    - Divide existing knowledge into department-specific sets and create separate vector indexes:
        - `support_vector_db`
        - `marketing_vector_db`
        - `tech_vector_db`
2. **Apply Routing Logic**
    - Use keyword-based `if-else` statements or a basic classifier to direct queries:

```python
if "pricing" in query or "subscription" in query:
    use marketing_vector_db
elif "error" in query or "bike lock" in query:
    use support_vector_db
```

3. **Query the Correct Index**
    - Retrieve documents only from the relevant department’s index, reducing irrelevant context.
4. **Fallback Strategy**
    - For ambiguous queries, a general index or merged retrieval can be used as backup.

✅ **Benefits**

- Enhances **retrieval relevance** by reducing noise from unrelated documents.
- Improves **latency**, as each vector DB is smaller and faster to query.
- Ensures **modularity** — departments can update or manage their data independently.

🧩 Conclusion

The proposed techniques are designed to make the RAG system more robust, enterprise-ready, and aligned with how real companies operate. By focusing on **domain-specific QA data** and **departmental modularity**, the system can deliver faster, more accurate, and more maintainable responses. These optimizations bridge the gap between raw LLM power and real-world organizational needs.