**Project Report**

# Intelligent Email Parsing and Information Retrieval with RAGs

---

## Introduction

The exponential growth of digital communication has necessitated efficient solutions for processing and extracting insights from vast email repositories. This project aimed to develop an end-to-end system leveraging Retrieval-Augmented Generation (RAG) techniques to process email data, generate embeddings for efficient retrieval, and automate context-based responses using advanced language models like GPT-4, Gemini, and Llama. The system was designed to address real-world challenges in email management, such as identifying critical information, tracking spending, and answering specific queries based on email content. By integrating cutting-edge technologies, the project sought to deliver a robust, scalable, and user-friendly solution.

---

## Initial Approach and Challenges

At the outset, the project was envisioned as a cloud-based solution leveraging **Google Cloud Platform (GCP)** to process emails in real-time. The proposed system included a marketplace application where users could connect their email accounts and receive answers to their queries directly via the platform. The idea aimed to provide a seamless, on-the-go experience by integrating advanced cloud capabilities for data processing and retrieval.

### a. Design of the Initial Approach

1. **Real-Time Email Processing**: Emails would be synced and processed in real-time using GCP services such as Cloud Pub/Sub for message ingestion and Cloud Functions for processing.
2. **Cloud Storage and Analysis**: Extracted data from emails would be stored in **BigQuery** for analysis and retrieval, and embeddings would be generated using GCP's AI tools.
3. **Marketplace Application**: Users would interact with the system via a web or mobile application, querying their email data for insights and receiving instant responses.

### b. Challenges Encountered

Despite its ambitious scope, the implementation of this cloud-based solution faced several critical challenges:

1. **Complexity of Real-Time Processing**: Setting up real-time email processing pipelines required extensive integration with Gmail APIs and precise handling of authentication, data streaming, and error recovery.
2. **Infrastructure Management**: Managing and maintaining cloud infrastructure, especially for large-scale data processing and retrieval, became increasingly time-intensive.
3. **Resource Constraints**: The cloud-based approach involved significant costs for storage, processing, and AI model usage, making it less feasible within the project’s constraints.

4. **Time Limitations**: The complexity of setting up the cloud environment and building a fully functional marketplace application extended the project timeline beyond acceptable limits.

### c. Decision to Pivot

Given the increasing difficulty and resource demands of the initial approach, the project was reoriented toward a **local development solution**. This pivot offered the following advantages:

- **Simplified Workflow**: Local processing of emails eliminated the need for real-time data streaming and reduced infrastructure dependencies.
- **Cost Efficiency**: Operating on local machines significantly cut down costs associated with cloud services.
- **Focused Scope**: Developing a local Retrieval-Augmented Generation (RAG) pipeline enabled me to focus on core functionalities without being burdened by ancillary challenges.

The decision to pivot not only streamlined the project but also allowed for the successful implementation of a robust, functional system within the given constraints.

---

## Implementation

### a. Data Acquisition

The foundation of this project was built on a dataset generated from **Google Takeout**, a service that allows users to export their data from various Google applications. The exported dataset included an `.mbox` file containing all emails from the user's Gmail account. This `.mbox` file served as the primary data source. The file was processed to extract essential email details such as subject, sender, recipient, date, and content, enabling downstream processing.

### b. Email Data Extraction

The `Extract_email_script.py/Extract_email_script.ipynb` module was developed to handle the `.mbox` file efficiently. The script extracted key metadata (e.g., subject, sender, recipient, date) and email bodies while removing unnecessary HTML elements. Cleaned email content was saved as `.txt` files, with each file representing an individual email. This structured data was critical for embedding generation and subsequent steps.

### c. Embedding Generation and Data Storage

In this phase, the extracted email text was transformed into high-dimensional embeddings using OpenAI's **text-embedding-ada-002** model. The decision to use this model was driven by its ability to generate **high-quality embeddings** with superior contextual understanding. Prior to selecting this model, an experimentation phase was conducted with alternative embedding techniques, including **all-MiniLM-L6-v2**, a popular open-source embedding model. While all-MiniLM-L6-v2 was efficient and lightweight, the embeddings it produced lacked the depth and nuance required for capturing complex semantic relationships in email data. OpenAI's text-embedding-ada-002 consistently outperformed in terms of **accuracy** and **relevance**, making it the ideal choice for this project.

To ensure manageable processing and efficient retrieval, the dataset was limited to **5000 emails dated after September 2024**. This decision reduced computational overhead and allowed for a focused analysis of recent and relevant emails, optimizing the embedding and retrieval processes.

Each email was processed to generate an embedding that encapsulated its semantic content. To enhance **retrievability**, metadata such as the subject, sender, recipient, and date was paired with the embeddings. This ensured that both the context and the metadata could be efficiently queried.

The generated embeddings and their associated metadata were stored in **ChromaDB**, a persistent vector database tailored for managing high-dimensional data. Several factors influenced the choice of ChromaDB as the storage solution: its scalability for large datasets, efficient similarity-based retrieval, seamless integration with Python workflows, and reliable persistence of embeddings across sessions.

The code for embedding generation, metadata pairing, and ChromaDB storage is implemented in the `RAG_Embeddings_Generate.py` file. This script forms a core component of the project, ensuring that all data is prepared for efficient retrieval and query processing.

### d. Retrieval-Augmented Generation

The **`RAG_Generation.py`** module implemented a Retrieval-Augmented Generation (RAG) pipeline to enable context-driven query answering. When a user posed a query, the system utilized OpenAI's **text-embedding-ada-002** model to convert the query into a high-dimensional embedding.

The generated query embedding was compared against stored email embeddings in **ChromaDB** using its built-in similarity search functions. These functions retrieved the most relevant email contexts based on the query, leveraging ChromaDB's optimized indexing mechanisms for high-speed retrieval. This ensured that the retrieved contexts were both accurate and semantically aligned with the query.

Once the relevant contexts were retrieved, they were passed to large language models (LLMs) — **GPT-4, Gemini, and Llama** — for response generation. Each model processed the retrieved context and query to produce coherent, contextually relevant answers. The responses were stored separately for further evaluation and comparison across models.

By combining the precision of **text-embedding-ada-002** for embedding generation with the efficient document retrieval capabilities of ChromaDB, the RAG pipeline provided a robust framework for answering diverse user queries with high accuracy and contextual relevance.

---

## Results

A total of **30+ different prompts** were given to each LLM—GPT-4, Gemini, and Llama—to generate responses based on the same context retrieved by ChromaDB. The responses for each model were saved in separate text files for analysis. The following prompt was used to guide the LLMs in generating responses:

**"You are an intelligent assistant that can answer questions based on provided context. The context is from emails, be sure to give a satisfactory response."**

This prompt was carefully designed after testing various variations and consistently produced the most accurate and contextually relevant results. By explicitly emphasizing the email-based context and the need for satisfactory responses, it helped the models focus on the provided information and minimize generic outputs.

### Key Findings

**1. Model Performance Overview**

- **GPT-4**: Delivered responses that were precise, contextually relevant, and highly fluent. It effectively generated detailed answers for queries requiring explanation (e.g., "Explain quantum computing in simple terms") but struggled to provide specific insights when the context lacked the required information.
- **Gemini**: Produced accurate and concise responses for straightforward queries. However, it occasionally failed to extrapolate broader answers from the context or identify subtle details, leading to incomplete responses.
- **Llama**: While Llama provided reasonable answers for simpler queries, its performance was inconsistent. Responses often lacked depth, and some contained incomplete information, especially for queries requiring nuanced understanding (e.g., "How does the stock market work?").

**2. Response Accuracy and Relevance**

- Across models, queries with clear and well-defined context (e.g., "Are there any overdue bills or payments in my emails?") yielded accurate responses.
- For ambiguous or broad prompts (e.g., "How much have I spent on electronics till now?"), all models struggled due to insufficient context in the emails, highlighting the need for additional data preprocessing or more sophisticated context enrichment.

**3. Analysis of Strengths**

- **GPT-4**: Consistently provided coherent and well-structured responses with a higher degree of fluency. Its explanation of "Quantum computing in simple terms" was the most detailed and accurate among the models.
- **Gemini**: Excelled in identifying specific details, such as the most frequent discount percentage offered in promotional emails (e.g., "40% off"). It also handled straightforward prompts efficiently.
- **Llama**: Performed well for direct prompts with sufficient context, such as "Do I have any emails with gift card codes or promo codes that I haven't used yet?"

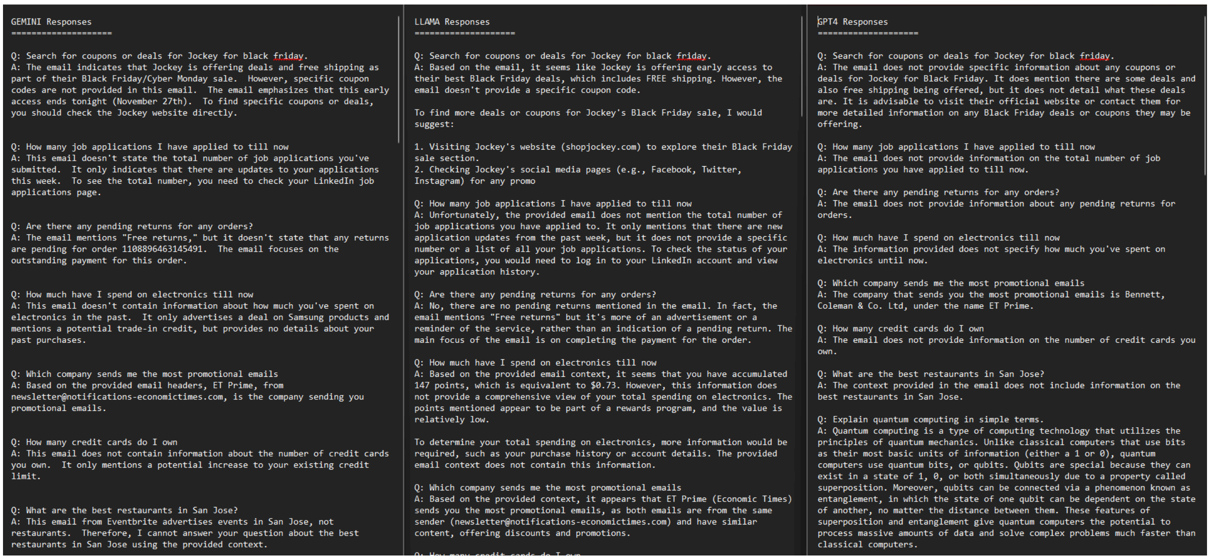
**4. Limitations Observed**

- **Context Gaps**: For queries relying on incomplete or implicit email data (e.g., "How many credit cards do I own?"), all models struggled to provide relevant answers.

- **Failure to Aggregate**: Queries requiring aggregation of information across multiple emails (e.g., “How many job applications have I applied to till now?”) could not be answered due to the lack of cumulative data processing.

**5. Insights and Observations**

1. **Prompt Effectiveness**: The chosen prompt ensured the models stayed focused on the email context while generating coherent and relevant answers. Variations of this prompt were tested, and this version consistently delivered superior results.
2. **Content-Specific Limitations**: The responses heavily relied on the specificity of the email content retrieved. When the context lacked detailed information, all models provided generic or incomplete answers.




*Responses generated*

---

# Improvements

The project demonstrated significant success in leveraging embeddings and retrieval for generating context-aware responses, but there remain areas for improvement, particularly in document retrieval, as it critically influences the quality of responses.

## 1. Enhancing Embeddings

- **Fine-Tuning Models**: Fine-tune text-embedding-ada-002 on email-specific data for improved contextual relevance.
- **Exploring Alternatives**: Evaluate models like Sentence Transformers or Multi-Query Dense Retrievers for faster, domain-aligned embeddings.

## 2. Improving Document Retrieval

- **Advanced Similarity Metrics**: Experiment with learned similarity functions or contextual matching for precise retrieval.
- **Metadata Weighting**: Use weighted metadata (e.g., sender, date) to prioritize relevant documents.

- **Cross-Document Analysis**: Enable simultaneous retrieval and aggregation of multiple documents for better query responses.

### 3. Retrieval Techniques

- **Pre-Filtering**: Filter by metadata (e.g., date range, categories) to improve relevance.
- **Cluster-Based Retrieval**: Group documents semantically and retrieve cluster representatives for reduced noise.
- **Dynamic Context Expansion**: Enrich retrieved documents with summaries or external data for richer input.

---

## Conclusion

This project successfully demonstrated the integration of advanced embedding models, efficient vector databases, and state-of-the-art language models to create a robust Retrieval-Augmented Generation (RAG) system tailored for email-based queries. By leveraging OpenAI's text-embedding-ada-002 and ChromaDB, the system effectively retrieved and processed relevant email contexts, enabling coherent and contextually accurate responses across diverse query types.

The analysis highlighted the strengths of GPT-4, Gemini, and Llama, showcasing their capabilities in generating high-quality responses. However, the project also identified areas for improvement, particularly in embedding optimization and document retrieval strategies. Enhancing these components will not only refine the precision and depth of responses but also unlock greater scalability and versatility for real-world applications.

This work serves as a foundation for further innovations in retrieval-augmented systems, with potential applications in personal email management, enterprise solutions, and customer service automation. The insights gained from this project pave the way for creating smarter, more efficient AI systems that bridge the gap between raw data and actionable insights.