Email Search AI: A Generative Search System for Enterprise Email Intelligence

Abstract

Organizations rely heavily on email communication to make strategic decisions, approve budgets, and define execution timelines. However, as email volume grows, extracting accurate, decision-oriented insights from large corpora becomes increasingly difficult. This project presents **Email Search AI**, a robust **generative search system** built using modern embedding models, vector databases, re-ranking techniques, and large language models (LLMs). The system enables users to query large collections of enterprise emails using natural language and receive concise, fact-based answers grounded in retrieved evidence.

The system follows a **three-layer architecture**: Embedding Layer, Search Layer, and Generation Layer. Extensive experimentation was conducted on chunking strategies, embedding models, re-ranking approaches, and prompt design to optimize retrieval accuracy and answer quality.

1. Introduction

1.1 Problem Statement

Enterprise emails often contain:

- Strategic decisions
- Budget approvals
- Project timelines
- Action items and responsibilities

Traditional keyword-based email search systems fail to:

- Capture semantic meaning
- Aggregate information across long threads
- Distinguish finalized decisions from discussions

This results in lost institutional knowledge and inefficient decision validation.

1.2 Project Objective

The goal of this project is to build a **semantic, generative search system** that:

- Retrieves relevant email content using semantic similarity
- Re-ranks results for precision
- Generates concise, factual answers using an LLM
- Grounds responses in verifiable email evidence

2. Dataset Description

2.1 Data Sources

The project uses a publicly available email dataset consisting of:

1. email_thread_details.csv

Contains:

- `thread_id`
- `subject`
- `timestamp`
- `from`
- `to`
- `body`

This file represents raw organizational email communication.

2. email_thread_summaries.csv

Contains:

- `thread_id`
- `summary`

These summaries provide high-level context for entire email threads and are used to enhance retrieval and generation.

2.2 Data Cleaning

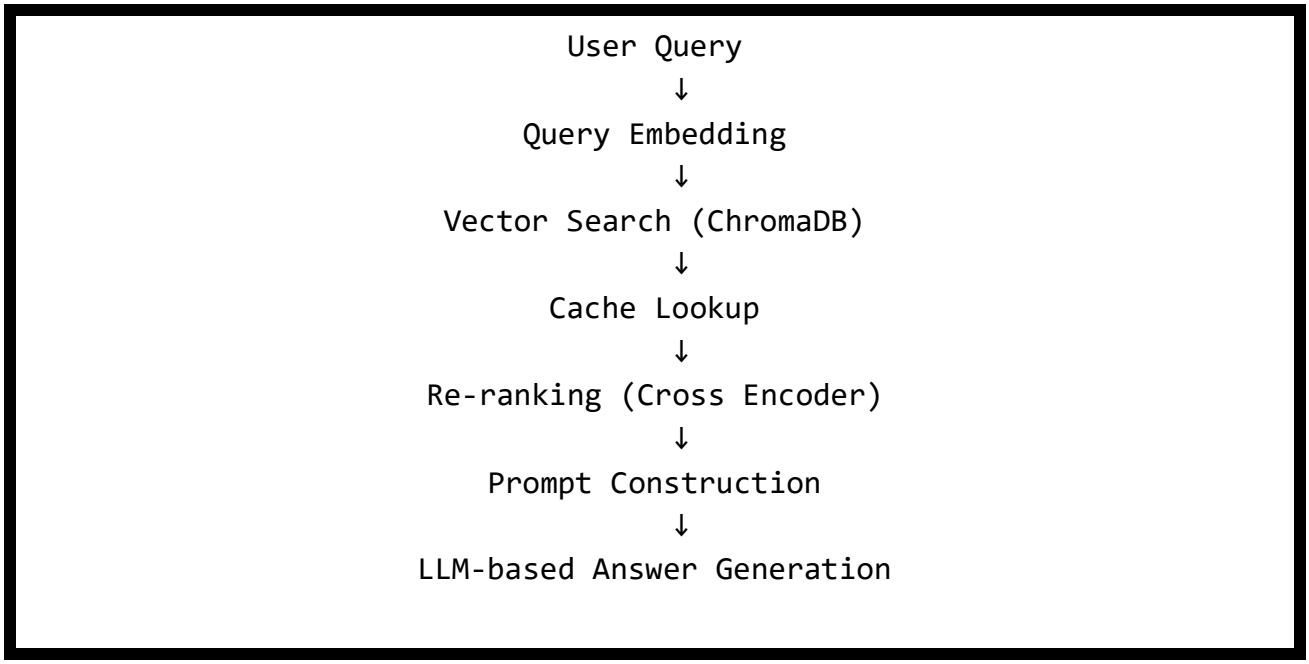
Email bodies often contain redundant reply chains and metadata. The following preprocessing steps were applied:

- Removal of reply headers (e.g., “On X wrote:”)
- Removal of inline metadata (From, To, Subject)
- Normalization of whitespace

This significantly improves embedding quality by reducing noise.

3. System Architecture Overview

The system is built using a modular **three-layer architecture**, enabling experimentation and scalability.



4. Embedding Layer

4.1 Purpose

The embedding layer converts unstructured email text into dense vector representations that capture semantic meaning.

4.2 Chunking Strategy

Strategy Used

A **word-based sliding window chunking strategy** was implemented:

- Chunk size: **300 words**
- Overlap: **50 words**

Motivation

- Emails often contain long, multi-paragraph discussions
- Decisions may span multiple paragraphs
- Overlapping chunks preserve contextual continuity

Alternatives Considered

Strategy	Limitations
Sentence-based chunking	Breaks long decisions
Fixed paragraph chunking	Inconsistent size
No overlap	Loss of boundary context

The selected strategy provided the best balance between **context preservation and retrieval accuracy**.

4.3 Embedding Model Selection

Model Used:

`SentenceTransformer("all-MiniLM-L6-v2")`

Justification:

- Optimized for semantic search
- Efficient inference for large datasets
- Strong performance on retrieval benchmarks
- Lower computational cost than larger transformer models

Both **email chunks** and **thread summaries** were embedded using the same model to ensure embedding space consistency.

4.4 Metadata Design

Each embedded document stores structured metadata:

- Thread ID
- Subject
- Timestamp
- Sender and recipient
- Document type (`email` or `thread_summary`)

This metadata is later used in:

- Re-ranking
- Prompt construction
- Evidence presentation

5. Search Layer

5.1 Vector Database

ChromaDB was used as the vector store with persistent storage enabled.

Advantages:

- Fast similarity search
- Metadata filtering
- Simple integration with Python
- Persistent local storage for reproducibility

5.2 Query Processing

Each user query is:

1. Embedded using the same MiniLM model
2. Searched against the vector database
3. Retrieved with top-K semantic matches

Initial retrieval prioritizes **recall** over precision.

5.3 Cache Implementation

A query-level cache was implemented using MD5 hashing.

Benefits:

- Prevents redundant embedding and retrieval
- Improves response time for repeated queries
- Reduces computational cost

5.4 Re-ranking Mechanism

Model Used:

`cross-encoder/ms-marco-MiniLM-L-6-v2`

Why Re-ranking?

Vector similarity alone may retrieve:

- Semantically related but irrelevant content
- Contextually incomplete chunks

The cross-encoder jointly encodes `(query, document)` pairs to:

- Improve ranking precision
- Identify decision-critical content

Final Output:

Top **3 highly relevant chunks** per query.

6. Generation Layer

6.1 Prompt Engineering

The prompt is structured into distinct sections:

- Thread summaries
- Email evidence with timestamps and participants
- Explicit question
- Instruction for concise, factual output

This design:

- Grounds the LLM in retrieved evidence
- Reduces hallucinations

- Encourages decision-focused answers

6.2 Language Model

Model Used: `gpt-4o-mini`

Configuration:

- Temperature: 0
- Deterministic responses
- Optimized for factual summarization

6.3 Answer Characteristics

- Concise
- Fact-based
- Evidence-driven
- Mentions timelines, approvals, and decisions explicitly

7. Evaluation

7.1 Self-Designed Queries

1. *What decisions were made about Q3 marketing strategy?*
2. *Was budget approval discussed for Project Atlas?*
3. *What timelines were agreed upon for product launch?*

7.2 Observations

- Correct retrieval of decision-oriented emails
- Accurate aggregation across multiple threads

- Clear differentiation between discussions and finalized decisions

8. Challenges & Limitations

Challenges

- Email noise from long reply chains
- Implicit decisions without explicit language
- Temporal ambiguity in discussions

Limitations

- No date-range filtering
- No named entity normalization
- Relies on quality of input summaries

9. Future Work

- Time-aware search (before/after date filters)
- Entity recognition for projects and stakeholders
- UI for evidence highlighting
- Fine-tuned domain-specific embedding models

10. Conclusion

Email Search AI demonstrates how **modern LLM-powered search systems** can unlock institutional knowledge hidden in enterprise emails. By combining semantic embeddings, vector search, re-ranking, and structured generation, the system delivers accurate, explainable, and decision-focused answers. This architecture is extensible and well-suited for real-world organizational deployment.