**N Dinesh Reddy**

**AI-Powered Communication Assistant – Documentation**

**1. Introduction**

The **AI-Powered Communication Assistant** is a smart email management and response system designed to help users efficiently process, analyze, and respond to incoming emails. The platform provides:

* Automated extraction of key email information (sender, phone number, alternate email, subject, body, priority, sentiment).
* Context-aware reply generation using a Retrieval-Augmented Generation (RAG) approach.
* Analytical insights on email traffic, priority distribution, and sentiment trends.

This system is particularly useful for support teams, interns, or anyone who needs to manage high volumes of customer or internal emails efficiently.

**2. Architecture Overview**

**a. Frontend**

The frontend is built using **Streamlit**, providing a simple and interactive web interface. The main components include:

1. **Inbox Tab**
   * Displays all filtered emails in a table with essential metadata.
   * Allows selection of an email to view full details and body.
   * Generates a draft reply using AI, considering email content, sentiment, and urgency.
2. **Analytics Tab**
   * Shows metrics such as total emails, emails received in the last 24 hours, urgent emails, and sentiment breakdown.
   * Provides charts for better visualization of email trends.

**b. Backend / Modules**

1. **preprocess.py**
   * Normalizes email column headers (handles variations like sender, from, content, body).
   * Extracts metadata:
     + **Priority** (heuristic-based on keywords like “urgent”, “reminder”).
     + **Sentiment** (positive, negative, neutral).
     + **Phone numbers** and **sender names**.
     + **Requirement type** (e.g., account access, billing, technical issue).
     + **Preview snippet** for quick reading.
2. **rag.py**
   * Implements the **Retrieval-Augmented Generation** (RAG) pipeline.
   * Retrieves relevant knowledge base (KB) documents using embeddings and/or keyword search.
   * Builds a context-aware prompt for the LLM to generate professional and empathetic email replies.
3. **reply\_generator.py**
   * Interface to generate replies using the RAG class.
   * Handles fallback mechanisms in case of KB retrieval or LLM errors.
4. **analytics.py**
   * Computes metrics on emails (e.g., sentiment counts, urgent vs. normal emails).
   * Generates bar charts and other visualizations for the dashboard.
5. **Optional DB Layer**
   * Emails and draft replies can be stored in **SQLite** or **MongoDB** to maintain persistence beyond the current session.

**3. Data Flow & Processing Steps**

1. **Load Email Dataset**
   * Emails are loaded from a CSV file (intern\_emails.csv) or optionally from a database.
   * Column headers are normalized to a consistent format.
2. **Preprocessing**
   * Extracts metadata: sender email, phone number, alternate email, priority, sentiment, requirement type, and snippet preview.
   * Filters relevant support/query emails for display.
3. **Inbox Display & Reply Generation**
   * Users select an email from the inbox table to view full details.
   * Clicking “Generate Reply” triggers the RAG-based AI model to provide a context-aware draft response.
   * Draft replies are stored in session state to maintain continuity.
4. **Analytics**
   * Computes statistics such as total emails, emails in the last 24 hours, sentiment breakdown, and urgent emails.
   * Generates visual charts for quick insight.
5. **Optional Database Integration**
   * Emails and replies can be stored in a database for persistent storage and retrieval.

**4. Key Design Decisions**

* **Heuristic Preprocessing**
  + Quick and efficient extraction of metadata without relying on heavy NLP models for common fields.
* **RAG-Based Reply Generation**
  + Ensures AI replies are **context-aware**, **empathetic**, and **concise**.
  + Uses knowledge base for domain-specific guidance.
* **Session-State Draft Management**
  + Prevents losing replies when switching between emails.
* **Analytics for Monitoring**
  + Provides actionable insights into email volume, priority, and sentiment trends.
* **Optional Database**
  + Adds persistence and scalability if the application is deployed in a multi-user environment.

**5. Tools & Libraries Used**

* **Python 3.x**
* **Streamlit** – Frontend UI
* **Pandas, NumPy, re** – Data handling and preprocessing
* **OpenAI / Groq API** – LLM for reply generation
* **sentence-transformers** – Embedding model for RAG context retrieval
* **SQLite / MongoDB** – Optional persistent storage
* **Matplotlib / Streamlit charts** – Data visualization

**6. How to Run**

1. Clone the repository.
2. Install dependencies:
3. pip install -r requirements.txt
4. Set up your Groq API key in secrets.toml.
5. Place email CSV (intern\_emails.csv) and knowledge base (faq.txt) in the data/ folder.
6. Run the Streamlit app:
7. streamlit run app.py
8. Open the browser and interact with the **Inbox** and **Analytics** tabs.