AI-Powered Communication Assistant – Architecture & Approach

Date: 2025-09-06

1) Problem

Organizations receive many support emails. Goal: fetch, filter, prioritize, extract info, and auto-draft responses with context (RAG).

2) Architecture

- Backend (Flask)
  - Endpoints: /init, /fetch_emails, /emails, /respond, /send, /analytics
  - Modules:
    - db_utils.py: SQLite schema, CRUD, analytics
    - email_utils.py: IMAP fetch, keyword filtering, parsing
    - ai_utils.py: sentiment, priority, RAG retrieval, reply drafting (OpenAI fallback to heuristic)
- Storage: SQLite
- Knowledge Base: docs/kb/* (md/txt/json)
- Frontend: Streamlit dashboard
  - Controls: init DB, fetch emails
  - Inbox view: priority queue, details panel
  - Draft editor + send
  - Analytics: counts + simple charts

3) Data Flow

IMAP → filter by subject → extract details → sentiment/priority → store in SQLite → dashboard lists (priority-first) → generate draft (RAG + LLM) → edit → send (simulate) → mark responded → analytics updates.

4) AI Techniques

- Sentiment: GPT mini classification or keyword heuristic
- Priority: keyword heuristic (urgent phrases)
- RAG: light retrieval (term overlap) over knowledge base documents; merged into prompt