AI Personal Email Assistant Documentation

Overview:

The Personal Email Assistant is a fully automated AI-powered tool designed to help users manage their Gmail inbox more intelligently. The assistant is capable of reading emails, summarizing content using a **large language model** (LLM), classifying the email type, identifying actionable items (e.g., scheduling), responding automatically, and pushing important notifications to Slack. It also integrates web search for information retrieval and calendar services to schedule events based on natural language content in emails. The goal is to build a smart, modular assistant that acts like a human virtual assistant with minimal user intervention.

Email Fetching and Storage:

- **Fetching Mechanism**: The system **uses OAuth 2.0 credentials** with appropriate **Gmail API** scopes(https://www.googleapis.com/auth/gmail.readonly, https://www.googleapis.com/auth/gmail.modify, https://www.googleapis.com/auth/gmail.send) to securely fetch emails.
- **Message Parsing**: It extracts sender, recipient, subject, timestamp, body, and checks for attachments.
- **Thread Linking**: The email thread Id is stored to maintain conversation context and allow threading of replies.
- **Database Schema**: Emails are stored in a MySQL database using a schema that supports threading, classification labels, summaries, and flags for whether replies have been sent.
- **Design Decision**: Initially, older emails were deleted before new ones were fetched to avoid duplication, but this behavior was modified to retain prior context and preserve full conversation history for better assistant context understanding.

LLM-Powered Summarization:

- **Model Used**: **facebook/bart-large-cnn** from Hugging Face is employed for abstractive summarization of email bodies.
- **Prompting Strategy**: Both the subject and body are combined to provide enough context for the LLM to generate coherent summaries.
- **Challenge**: Some emails exceeded the 1024 token limit of the model. This was mitigated by truncating long bodies or using text segmentation.

- **Output Handling**: The summary is cleaned and stored in the database to be reused in replies and Slack notifications.

Email Classification:

- **Technique**: A rule-based or ML model (can be upgraded) classifies emails into categories like Finance, Work, Personal, Promotions, etc.
- **Use Case**: The classification is used to determine the tone of replies and whether to push the notification to Slack.

Search Integration:

- **Trigger Detection**: Keywords or phrasing patterns are matched using a utility function should_trigger_search() to identify if a search is needed.
- **API Used**: Google Custom Search API is used to fetch relevant URLs in response to questions or ambiguous email queries.
- **Fallback Handling**: If the LLM summary lacks clear actionable info but the intent suggests a query (e.g., “what is…”), a search is triggered automatically.

Calendar Integration:

- **Functionality:** If the summary or body contains scheduling information, it is parsed using date/time recognition and integrated into Google Calendar.
- **Challenge:** Parsing vague time expressions (e.g., “next Friday at noon”) was tricky, but using libraries like date parser helped extract temporal data reliably.
- **Fallback:** If extraction fails, a follow-up message is sent asking the sender to confirm time/date.

Automated Replies:

- **Reply Generation:** The summary or search result is wrapped into a polite message using a reply template (generate_reply()).
- **Sending Mechanism:** Gmail API's users.messages.send endpoint is used to send the message using the original threadId for continuity.
- **Error Handling:** Errors like 404 Requested entity not found were caused by expired threadIds or deleted drafts. Added error handling with retry logic and logging.

Slack Notifications:

- **Trigger:** Only important emails (e.g., containing "urgent", from known senders, or labeled Work/Finance) are sent to Slack.

- **Message Format:** Uses Markdown formatting for subject, sender, and category.
- **API Used:** Slack Web API's chat.postMessage with bot tokens.

Flow Orchestration:

- **Entry Point:** Personal_email_assistant() in Main.py orchestrates the end-to-end process.
- **Steps:**
    1. Fetch new emails.
    2. Summarize content.
    3. Classify the email.
    4. Decide on search, calendar, or simple summary-based reply.
    5. Send the reply.
    6. Push important updates to Slack.
- **Counters:** Track total emails processed, replies attempted, sent, skipped.
- **Design Choice:** Centralized orchestration with stateless modules to allow easy testing and upgrades.

Challenges and Solutions:

Challenge	Solution
OAuth token refresh issues	Implemented token storage and auto-refresh logic
Gmail API 404 errors	Improved validation and ensured correct thread Id usage
Long emails breaking LLM	Truncated or chunked email body input
Misclassification	Future plan: Train fine-tuned transformer or use semantic classification
Calendar extraction failures	Used fallback questions and fuzzy datetime parsing