Of course. Based on our discussion and the limitations we identified in the papers, here is a breakdown of *how* you can implement the key features for your project, followed by a consolidated list of the required methodologies, tech stack, and datasets.
Yes, I have the full context of your project: an **AI-Powered Email Voice Assistant** for composing, summarizing, and managing tasks from your email.

How to Implement the Recommended Features

Here are the practical steps to build the core modules of your project, turning the limitations of the research papers into features.

1. Implement an Integrated, Multi-Functional System

Your main goal is to overcome the fragmented, single-purpose nature of systems like GlassMail (composition only) [1]and

ShortMail (summarization only)[2].

How to do it:

- **Step 1: Authenticate and Connect to Email:**
  - Use the **Google Gmail API** or **Microsoft Graph API**. You will need to set up OAuth 2.0, which is the standard, secure way to ask users for permission to access their email. This will allow your app to read and send emails on their behalf.
- **Step 2: Build an Intent Router:**
  - After converting the user's voice command to text, your first task is to understand what they want. You can start with a simple rule-based system (e.g., if the command contains "read my inbox," trigger the summarization function).
  - For a more advanced approach, use a single API call to your LLM (Gemini) with a prompt like: "Classify the user's intent from this command: '[user's command]'. Choose one of: Compose, Summarize, Reply, or Find Tasks."
- **Step 3: Create the Summarization Module:**
  - Use the email API to fetch the 2-3 most recent unread emails.
  - For each email, send its content to the Gemini API with a clear prompt: "Summarize this email from [Sender] about [Subject] in under 30 words for a voice assistant to read aloud. Focus on the main question or key information."
  - Use a Text-to-Speech service to read the summaries back to the user.
- **Step 4: Create the Reply Module:**
  - After an email is summarized, your assistant can ask, "Would you like to reply?"
  - If the user says yes, capture their dictated response and use the Gemini API with a prompt that includes context:

"Draft a reply to the following email: '[Original Email Body]'. The user wants to say: '[User's Dictated Response]'. Make it a complete and polite email." This directly addresses a key limitation of GlassMail[3].

2. Implement Proactive Task Management and Scheduling

This feature directly addresses the limited interactions found in the Microsoft reminder system [4]and the temporal reasoning errors seen in

TellTime[55].

How to do it:

- **Step 1: Build the Task Extraction Module:**
    - As your system processes new emails, send the email body to the Gemini API with a task-focused prompt: "Scan this email for actionable tasks, events, or meeting requests. Extract the task description, date, and time. If none are found, respond with 'None'. Format the output as a JSON object."
- **Step 2: Implement a Confirmation Dialogue (Crucial Step):**
    - The
      TellTime paper showed that AIs often make errors with dates and times[6666]. To prevent this,

      do not schedule tasks automatically.
    - Instead, have your assistant confirm with the user first: "I found a potential meeting request in the email from [Sender] for [Date] at [Time]. Should I add this to your calendar?" This human-in-the-loop approach makes your system much more reliable.
- **Step 3: Connect to a Calendar API:**
    - Use the **Google Calendar API** or **Microsoft Graph API**. When the user confirms the task, your backend server will make an API call to create the event in their calendar.

3. Implement Enhanced Personalization

This solves the "robotic tone" problem identified in
GlassMail [7]and

Emails by LLMs [8]by using a technique suggested in the

LaMPost paper[99].

How to do it:

- **Step 1: Collect User Writing Samples:**
  - During the initial setup of the app, ask the user for permission to analyze 3-5 emails from their "Sent" folder. This is a one-time step.
  - Store these email bodies in your database, linked to the user's profile.
- **Step 2: Use Few-Shot Prompting for Composition:**
  - When the user wants to draft a new email, dynamically construct a prompt that includes their samples as examples. This is a powerful technique called "few-shot learning."
  - Your prompt to the Gemini API would look like this:
    ```
    You are an AI assistant helping a user write an email in their specific style. Here are examples of how they write:
    ---
    Example 1: [Content of user's sample email #1]
    ---
    Example 2: [Content of user's sample email #2]
    ---
    Now, matching that style, tone, and vocabulary, write a new email based on this instruction: "Tell my manager I've finished the report and will send it over shortly."
    ```
  - This guides the AI to generate text that sounds much more like the user and less like a generic template.

Methodologies, Tech Stack, and Datasets

Here is the consolidated list for your project report and presentation.

Methodologies

- **Human-in-the-Loop Design:** Key actions like sending an email or scheduling a task are always confirmed by the user. This improves system reliability and user trust, mitigating AI errors noted in papers like
  TellTime[10].

- **Few-Shot Learning:** The system is personalized by providing the LLM with a few examples of the user's own writing in the prompt. This avoids the need for model retraining and addresses the personalization challenges highlighted in
  GlassMail and LaMPost[11111111].

- **API-Driven Architecture:** The project will be built by integrating several powerful, specialized cloud services (for speech, language, email, and calendar) rather than building each component from scratch.

Tech Stack

- **Mobile App (Frontend):** React Native or Flutter (for building a cross-platform app for both iOS and Android).
- **Backend Server:** Python with FastAPI or Flask (for handling logic and API requests).
- **Core AI Services:**
    - **Large Language Model:** Gemini API (for composition, summarization, and task extraction).
    - **Speech-to-Text:** Google Cloud Speech-to-Text API.
    - **Text-to-Speech:** Native platform APIs (iOS/Android).
- **API Integrations:**
    - Google Workspace APIs (for Gmail and Google Calendar).
    - Microsoft Graph API (for Outlook and Outlook Calendar).
- **Database:** PostgreSQL or MongoDB (for storing user profiles, preferences, and personalization samples).

Datasets

For a project of this nature, you will primarily use live data via APIs, not static datasets for training.

- **Primary Data Source:** The system will operate on the **user's own live email and calendar data**, accessed securely through the official APIs.
- **Personalization "Dataset":** This is not a public dataset. For each user, the dataset consists of **3-5 sample emails collected from their own "Sent" folder** during setup. This data is used exclusively for personalizing the tone and style of generated emails.
- **Model Training:** You will **not** be training or fine-tuning a large language model. Your project's "intelligence" comes from leveraging powerful pre-trained models (like Gemini) through sophisticated prompting and a well-designed system architecture.