VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

Natural Language Processing Application

Final Project Report

A12 - Intelligent Email Management System

Instructors: Nguyễn Hồng Bửu Long
Lương An Vinh

Students:

No.	ID	Name
1	21127448	Nguyễn Minh Thuận
2	21127327	Nguyễn Trần Trung Kiên
3	21127412	Hồ Bạch Như Quỳnh
4	22127081	Huỳnh Lê Hải Dương

Ho Chi Minh City, 2025

TABLE OF CONTENT

1. Team information

1.1. Team members list

No.	ID	Name	Role
1	21127448	Nguyễn Minh Thuận	Team lead
2	21127327	Nguyễn Trần Trung Kiên	Member
3	21127412	Hồ Bạch Như Quỳnh	Member
4	22127081	Huỳnh Lê Hải Dương	Member

1.2. Task distribution

No.	Task	Executor	Result
1	Literature Review	Hải Dương	100%
2	UI design	Như Quỳnh, Hải Dương	100%
3	UI implements	Như Quỳnh	100%
4	Set up and implement database (Firestore)	Như Quỳnh	100%
5	Testing Data generation	Hải Dương	100%
6	System design	Minh Thuận	100%
7	Backend implements	Như Quỳnh, Minh Thuận, Hải Dương	100%
8	Scheduling API usage	Minh Thuận	100%
9	System Testing	Minh Thuận, Hải Dương	100%
10	Build prompts and features	Trung Kiên	100%
11	Set up and implement Grafana for monitor	Trung Kiên	100%
12	Evaluate classification feature	Trung Kiên	100%

2. Project Overview

This project is smart mail helper made to aid users in order handling and replying to many types of mails. The system does incoming mail sorting into four groups - Work, Commercial, Fraud, and Others - and gives more understanding like importance scoring, short summarizing, and related reply ideas. The aim is to lower the thinking weight on users help them pay attention to key notes and make sure on time right answers.

It has an simple interface done with Streamlit, connects to Gmail for live email fetching, and uses Google Gemini's advanced NLP capabilities for crucial features. To make things work well and be cheap, handled emails are kept in Firebase; the setup has good watching using Prometheus and Grafana.

2.1. Purpose and Motivation

The drive of this project is to tackle the increasing problem of email overload faced by people who use Gmail for different things - like work, school, news, and events. With the growing number and complexity of emails people often have a hard time:

- Picking out important messages in a sea of promotions notifications and possible scams.
- Ordering which ones to answer so that urgent or high-impact emails get addressed quickly.
- Grasp the main point of long or complicated emails without reading every word.
- Write suitable responses promptly and tactfully, particularly for work-related or delicate matters.

By automating email sorting, highlighting which need attention first, summarizing key points, and suggesting replies, this project helps users take better control of their inboxes - saving time and making sure important messages don't get overlooked or fraudulent messages don't trick them. The addition of watchfulness tools also makes sure of openness and trustworthiness in how the API is used and the system's performance.

2.2. Intended Users and Beneficiaries

- This project is made for **Gmail users who get a large lot of emails** for many reasons, like students, workers, and people who deal with news and event messages. The system helps a lot with folks who:

- Having trouble staying up with high email flow and want to find key notes fast.
- Wish to skip missing important or top-level mails.
- Need aid telling real, ad, and fake mails.
- Likes automated suggestions for replying to mails in a professional and contextual manner. Values short summaries and prioritization to make their mail workflow easy.

- By giving category, priority, summary, and reply suggestions the system helps users save time reduce stress and better manage their mails.

2.3. Technologies Used

- The project leverages a modern, hybrid technology stack to deliver intelligent email management:

- **Gmail API (IMAP & OAuth):** Used to securely retrieve email data directly from the user's Gmail account, supporting both App Password and OAuth authentication methods.
- **Streamlit:** Provides a fast, interactive, and easy-to-maintain web-based user interface for email browsing, categorization, and reply.
- **Firebase (Firestore):** Stores processed email data (such as categories, summaries, and priorities) to minimize redundant processing and reduce API costs.
- **Hybrid local processing:** Most data processing (classification, summarization, etc.) runs locally on the user's machine, ensuring privacy and responsiveness.
- **Google Gemini API:** Utilizes Gemini's advanced natural language understanding for four core tasks:
- **Categorization:** Classifies emails as Work, Commercial, Fraud, or Others.
- **Prioritization:** Assigns a priority score based on variety inputs, including content and context.
- **Summarization:** Generates concise summaries for each email.
- **Reply Suggestion:** Provides context-aware, polite reply drafts.
- **Prompt Engineering:** Carefully crafted prompts guide Gemini to perform each task accurately and consistently.

- **Prometheus & Grafana:** Monitors API usage, error rates, and latency in real time, providing transparency and operational insights through dashboards.

- This combination of technologies ensures the system is robust, scalable, user-friendly, and capable of delivering high-quality, AI-powered email management.

3. System Architecture and Diagrams

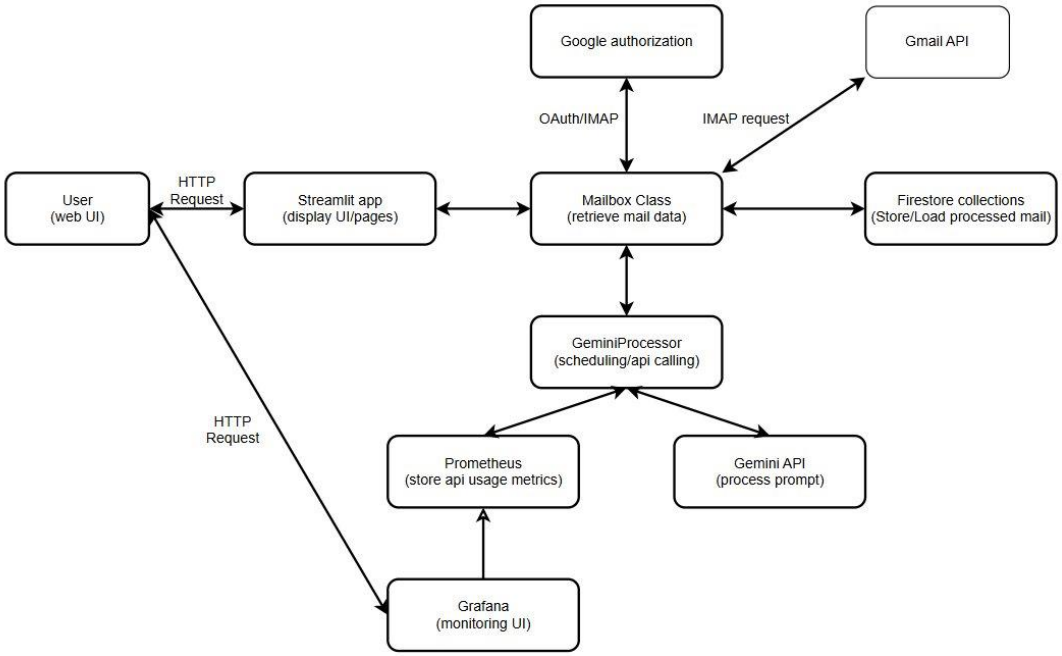
3.1. System Overview

The system starts with retrieving emails via API then performs classifying and prioritizing for each email. After that, the category and priority score of each email will be stored on Firestore to reduce the amount of data processing on next run. With each email, the system can also summarize the content and provide reply suggestion to that email. With that, the main features include classifying, prioritizing, summarizing and providing reply suggestion, these would be handled by using Gemini API throughout prompting.

The Gemini API usage will be monitored with Grafana, a side-service that should be running along with our system. Grafana will count and visualize data such as total number of API calls, error API request, average time for each request...

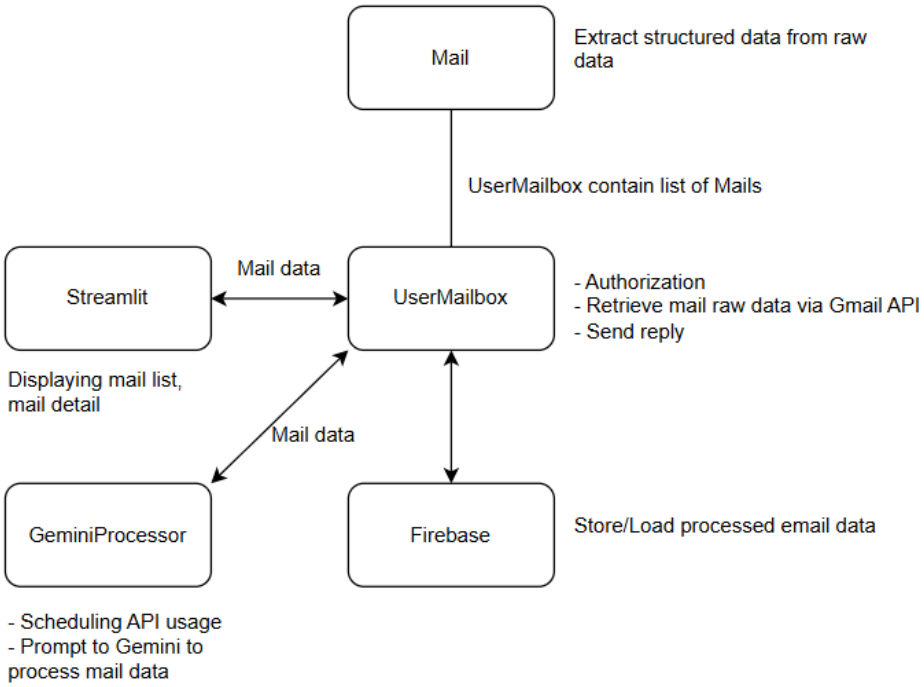
3.2. Technical Diagrams

a) System Architecture Diagram



Our application is a hybrid system that runs on the user side. The application itself calls Api to third party service for authorization (Google authorization), storing (Firestore) and processing data (Gemini). For monitoring Api usage, users need to run Prometheus and Grafana on their end along with our app.

b) Component Diagram



c) Database Schema

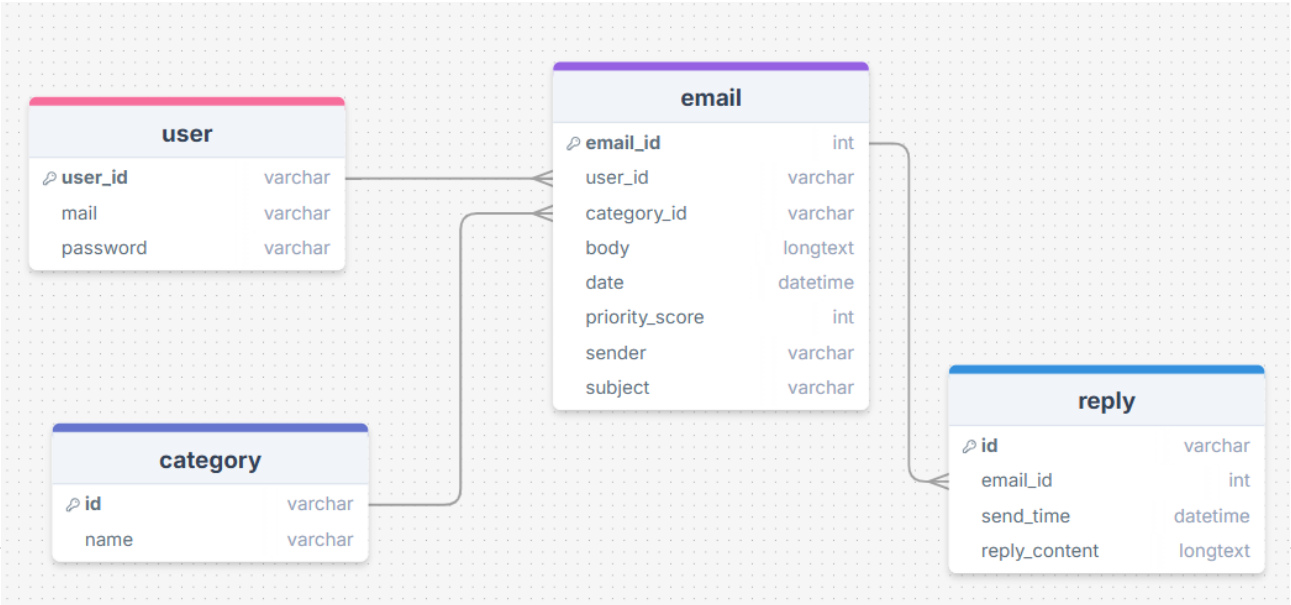
- The **user** table stores basic user account information (email and password).

- Each **user** can receive **multiple emails**, represented by a one-to-many relationship from user to email.

- The **email** table contains individual emails with metadata such as subject, body, send date, and a priority score. Each email belongs to one user and one category.

- The **category** table defines different types of email categories (e.g., work, commercial), forming a one-to-many relationship with email.

- The **reply** table stores responses to emails. Each reply is linked to exactly one email, and optionally, can be associated with a user if you include the responder's user_id.



4. Project Features

- Automatic email classification:

- Classify email to a list of pre-defined categories (Work, Commercial, Fraud, Others). The email's category will be an essential element for prioritizing the email later
- Based on the subject and body of an email, we first define each category then adding rules to guide Gemini for the task
- Our prompt:

```
def classify_email(self, subject: str, body: str) -> str:
    prompt = f"""
    You are an intelligent email classification system.

    Your task is to categorize an email into one of the following categories:
    Work,Commercial,Fraud,Others.

    Definitions:
    - Work: Professional communication such as meeting invites, reports, or project discussions. Typically comes from colleagues, clients, or managers.
    - Commercial: Marketing or promotional emails like newsletters, discounts, sales campaigns. They DO NOT request passwords or financial information.
    - Fraud: Emails pretending to be from trusted entities (e.g., banks), containing suspicious links, requests for login credentials, or urgent action.
    - Others: Emails that are personal, social, or do not clearly match the above.

    Rules for classification:
    - If an email includes both promotional content and a suspicious request or link, classify it as "Fraud".
    - If an email references "invoice", "billing", "payment" but comes from unknown or suspicious domains → classify as "Fraud".
    - If the subject contains business-related terms like "meeting", "project", "report", "update" and the tone is formal → classify as "Work".
    - If the email contains greetings like "Dear customer", "Congratulations", "Limited time offer" and includes a product or deal → classify as "Commercial".
    - If the message urges you to "click immediately", "verify your account", or "your account is suspended" → classify as "Fraud".

    Email to classify:
    - Subject: {subject}
    - Body: {body}

    Respond with ONLY ONE WORD from this list: Work, Commercial, Fraud, Others.
    DO NOT explain your reasoning.
    DO NOT include "Category:" or any other text.
    """.strip()
```

- Prioritizing emails to reply:

- Help the user to know which email they should prioritize to read or reply without having to read every single email.
- For this task, we created our rules for scoring the importance of each email based on its' subject, body, category and the date received.
- Our prompt:

```
def prioritize_email(self, subject: str, body: str, category: str, date: str) -> str:
    prompt = f"""
    You are an intelligent email assistant that evaluates the priority of emails.
    Start with a **base score of 6**. Adjust this score based on the following rules and context to return a final score between **0 and 10**.
    ---
    *Scoring Rules*:
    1. **Based on Category**:
    - "Work": Add 2 points
    - "Commercial": Subtract 2 points
    - "Fraud": Subtract 3 points
    - "Others": No change – score is adjusted only based on subject, body, and date
    2. **Subject & Body Content**:
    - Add points if email contains urgency, deadlines, meetings, approvals, requests for actions, reports
    - Subtract points for promotional language like "discount", "subscribe", "limited offer", "congratulations"
    - Add 1 point for financial or transaction-related content (e.g., invoice, payment notice) IF it's from a trusted source
    3. **Date Received**:
    - Add 1 point if the email was received today or yesterday AND has time-sensitive content
    - Subtract 1 point if the email is older than 7 days and contains no urgency
    ---
    *Email to evaluate*:
    - Category: {category}
    - Subject: {subject}
    - Body: {body}
    - Date received: {date}
    ---
    Respond with ONLY a number between **0 and 10**, representing the final priority score.
    Do NOT explain your reasoning.
    Do NOT include any additional text or formatting.
    """.strip()
```

- Summarizing email for core insight:

- Summarize the content of the email for better understanding without the need of reading the entire email
- In this task, we simply prompt for Gemini to summarize the content to 1 or 2 sentences with the same language as the email based on the category, subject and body
- Our prompt:

```
def summarize_email(self, subject: str, body: str, category: str) -> str:
    prompt = f"""
    You are an AI assistant. Please summarize the following email based on its content and its category: "{category}".
    Email:
    - Subject: {subject}
    - Body: {body}
    Return a short summary (1-2 sentences), capturing the key message. Please note that the summary should be concise and to the point and should be the same language as the email.
    Do NOT include any additional text or formatting.
    """.strip()
```

- Suggesting a useful reply to the email:

- Help the user to come up with a relevant and polite reply to the email
- We have Gemini to generate suggestion that follow some constrains involve in the tone, format based on the email category, subject and body
- Our prompt:

```
def suggest_reply(self, subject: str, body: str, category: str) -> str:
    prompt = f"""
    You are a highly competent AI assistant specialized in drafting thoughtful and professional email replies.

    Based on the given email's content, subject, and assigned category ("{category}"), generate a **complete, relevant, and polite reply**.
    Ensure the response:
    - Appropriately addresses the context and intent of the original message
    - Maintains a professional and natural tone
    - Provides necessary details, clarifications, or next steps if applicable
    - Is coherent and easy to read
    - Can be longer than 3 sentences if the context requires
    Do NOT repeat the subject or header. Your reply should be formatted as the body of an actual email.
    ---
    📧 Email:
    - Subject: {subject}
    - Category: {category}
    - Body: {body}
    ---
    ✉️ Write your reply below:
    """.strip()
```

- Monitoring API usage and respond time:

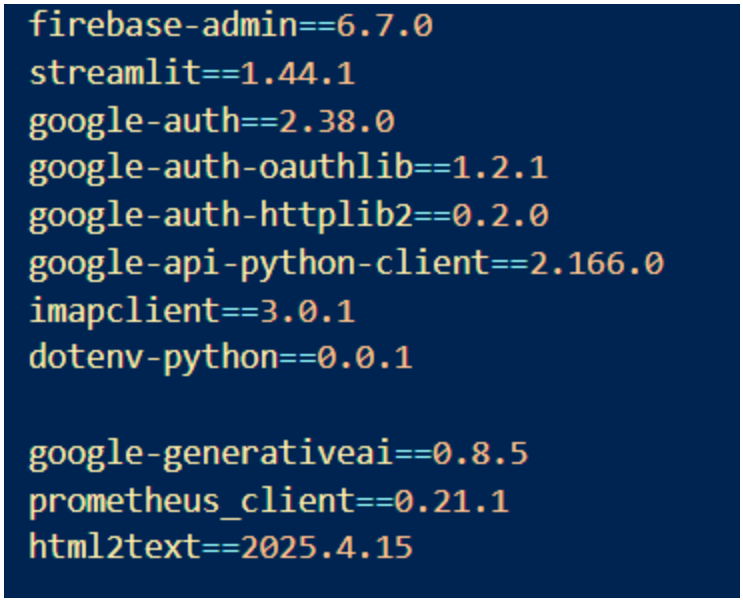
- With API usage and respond time in checked, user can have better understand of the limitation of the system and use it properly
- To monitor API usage and respond time, user can run Prometheus alongside with the system and have the data display on Grafana dashboard on localhost:3000

- Regular API Testing: Set up an automated scheduler to periodically send health-check requests to the email categorization API (e.g., every 10 minutes). Prometheus will record results such as success rates, errors, and latency.

5. Installation Guidelines

5.1. Prerequisites

- Python 3.10
- pip 25.0
- Python libraries required:



5.2. Setup Steps

1. Clone the repository from GitHub: https://github.com/haiduonghuynhle/NLPA-FinalProject
2. Install backend dependencies:

   ```
   pip install -r requirements.txt
   ```

3. Install, setup and run Prometheus, Grafana if needed (Follow 5.3 for more detail)
4. Configure IMAP for Gmail account to use App password (if desire to use app password, follow 5.4 for more detail)
5. Run the program:

   ```
   streamlit run main.py
   ```

5.3. Install Prometheus and Grafana

a) Install Prometheus on Windows

- Step 1: Download Prometheus

- Visit: https://prometheus.io/download
- Download the Windows version (prometheus-x.x.x.windows-amd64.zip)
- Extract it to a folder, e.g., C:\prometheus

- Step 2: Configure Prometheus

- Create a prometheus.yml file in the Prometheus directory with the following content:

```
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: "python_app"
```

```
  static_configs:
    - targets: ["localhost:8000"]
```

- *Step 3: Run Prometheus*

- Open Command Prompt, navigate to the Prometheus folder, and run:
  `prometheus.exe --config.file=prometheus.yml`
- Access Prometheus via your browser at: http://localhost:9090

b) Install Grafana on Windows

- *Step 1: Download Grafana*

- Visit: https://grafana.com/grafana/download
- Choose the Windows installer (.exe) or .zip archive

- *Step 2: Install or Run Grafana*

- If using .exe: install it like a regular Windows application
- If using .zip: extract it and run bin\grafana-server.exe

- *Step 3: Access Grafana*

- Open a browser and go to: http://localhost:3000
- Default login credentials:
  - Username: admin
  - Password: admin (you will be prompted to change it)

c) Connect Prometheus to Grafana

- In Grafana, go to Configuration → Data Sources → Add data source

- Choose Prometheus

- Enter URL: http://localhost:9090

- Click Save & Test to verify the connection

d) Export Metrics from a Python Application

- *Step 1: Install the Prometheus client library*

```
pip install prometheus_client
```

- *Step 2: Create a sample Python metrics exporter*

```
from prometheus_client import start_http_server, Summary, Counter
import time, random

REQUEST_TIME = Summary('request_processing_seconds', 'Time spent processing
request')
REQUEST_COUNT = Counter('my_app_requests_total', 'Total number of requests')

@REQUEST_TIME.time()
def process_request():
    time.sleep(random.random())
```

```
        REQUEST_COUNT.inc()

    if __name__ == '__main__':
        start_http_server(8000)
        print("Serving metrics on http://localhost:8000/metrics")
        while True:
            process_request()
```

- *Step 3: Ensure Prometheus scrapes the Python app*

- In your prometheus.yml:

```
static_configs:
  - targets: ["localhost:8000"]
```

e) Create a Dashboard in Grafana

- Go to Dashboard → New

- Add a panel using the following metrics:

- my_app_requests_total
- request_processing_seconds

f) Import Dashboard from JSON File

- In Grafana, go to Dashboards → Import

- Click Upload JSON file and select your dashboard .json

- Select Prometheus as the data source if prompted

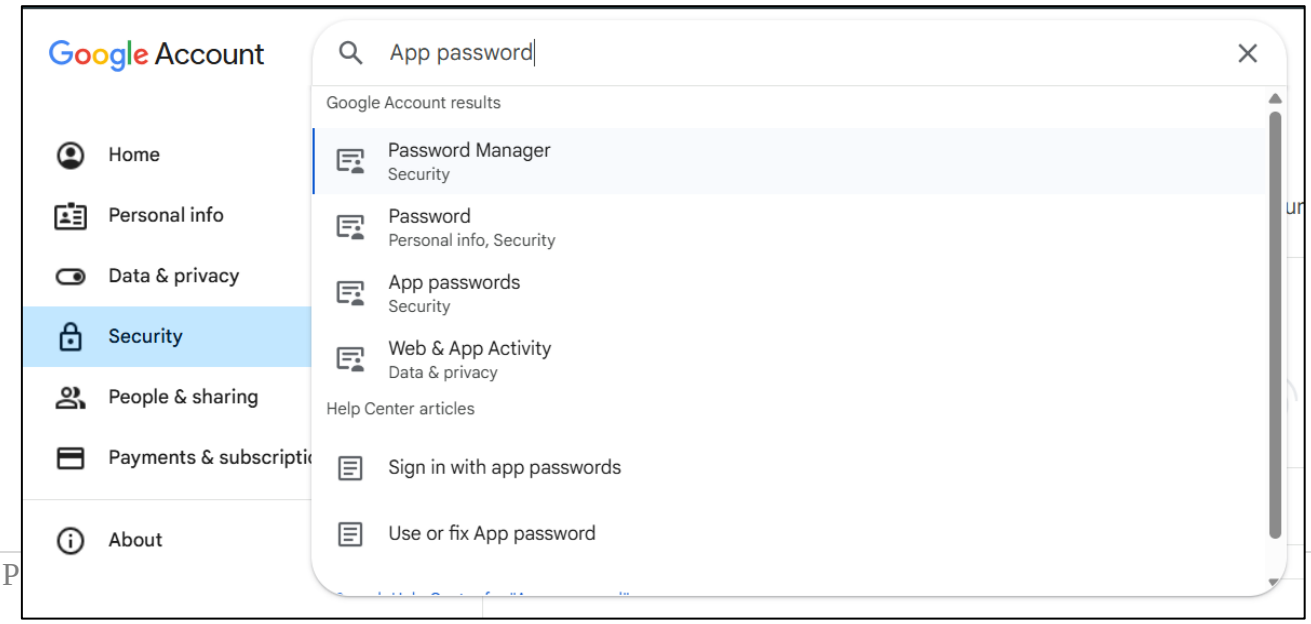
- Click Import to finish

5.4. Setup Gmail account to use app password

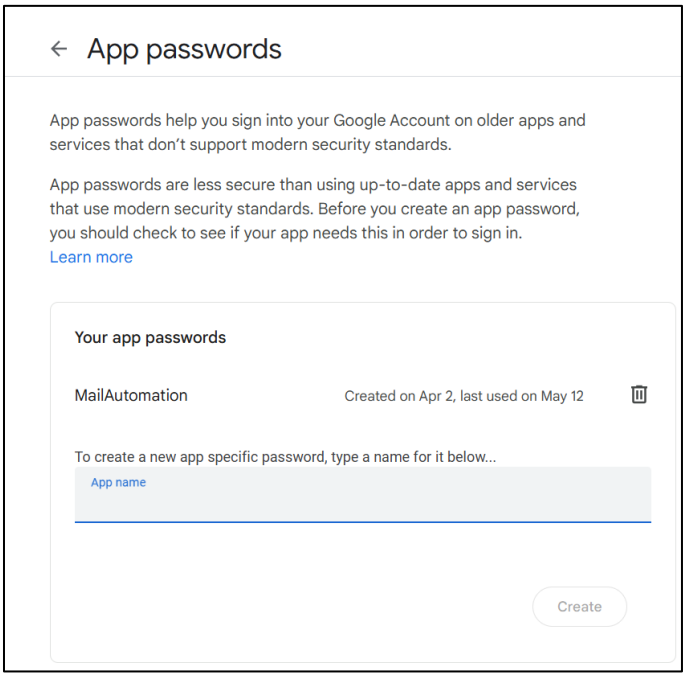
- Go to your Google account management page

- Select Security and search for App password on Search bar

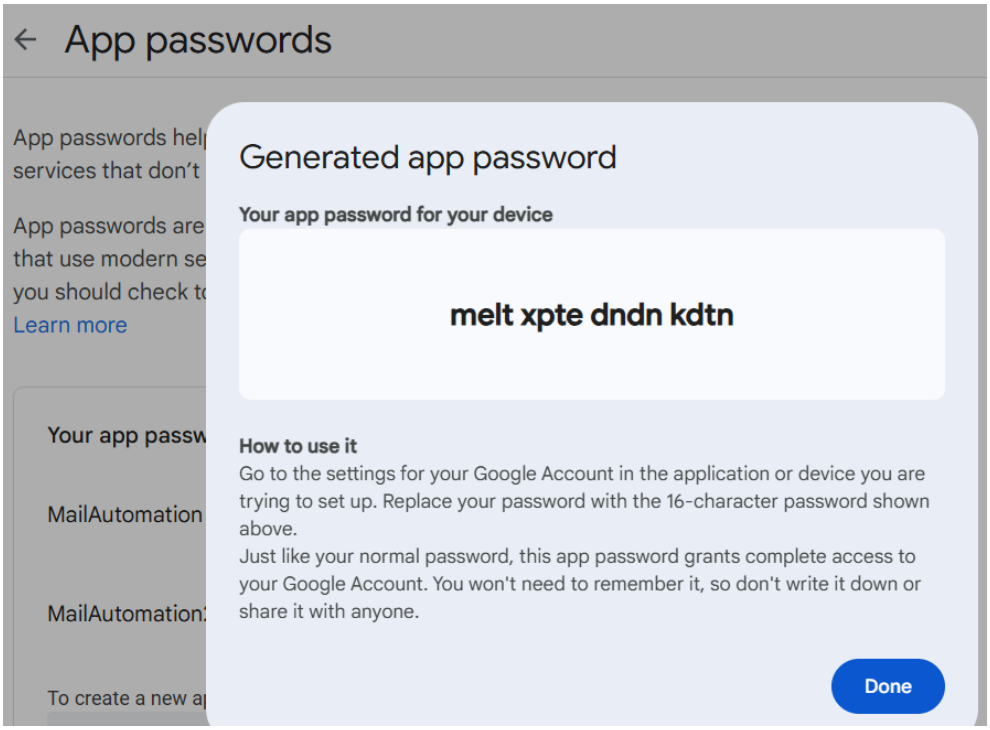
- Choose 'App passwords'



- Enter new app name and select ‘Create’



- Google will create a new password with 16 characters that can be used for our system and Select ‘Done’



6. Usage Guidelines

6.1. Launching the Application

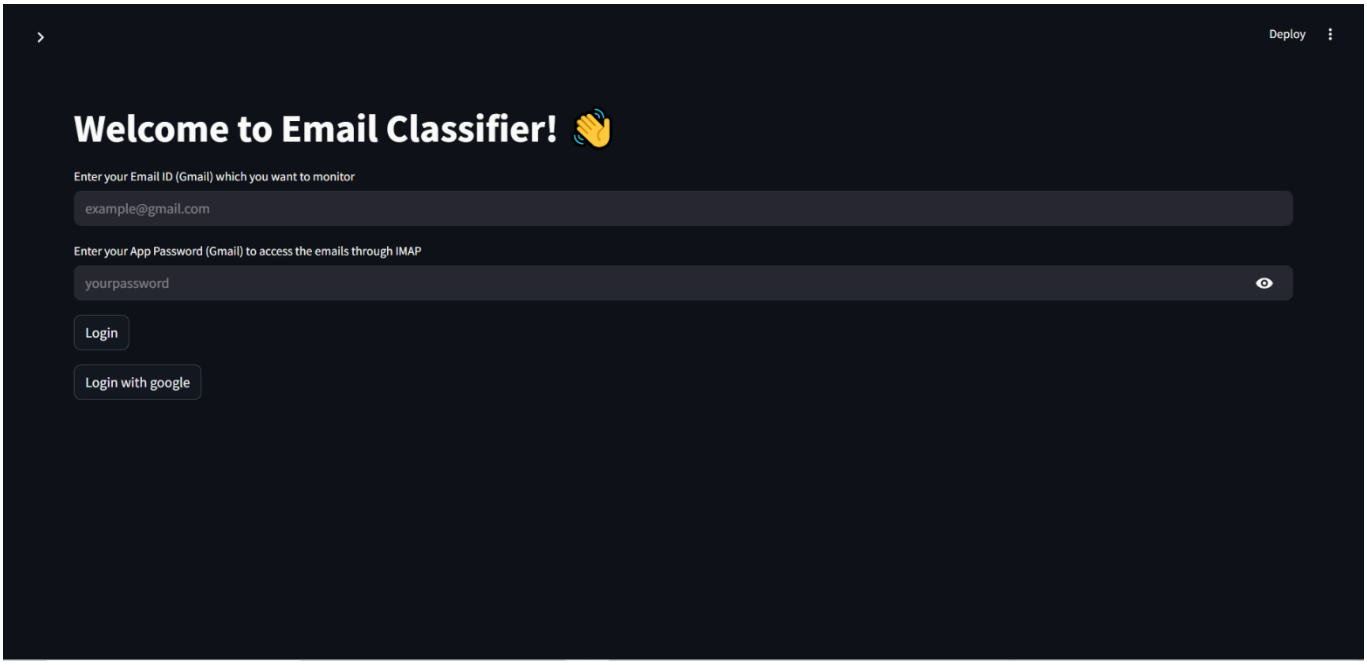
- First, to ensure all required modules are installed, run the following command to install the dependencies listed in requirements.txt (make sure you are in the folder containing the requirement.txt file):

```
pip install -r requirement.txt
```

- To launch the Streamlit application, use the following command:

```
streamlit run main.py
```

- Once the application is successfully launched, the login page (homepage) will appear.



6.2. Using the System's core function

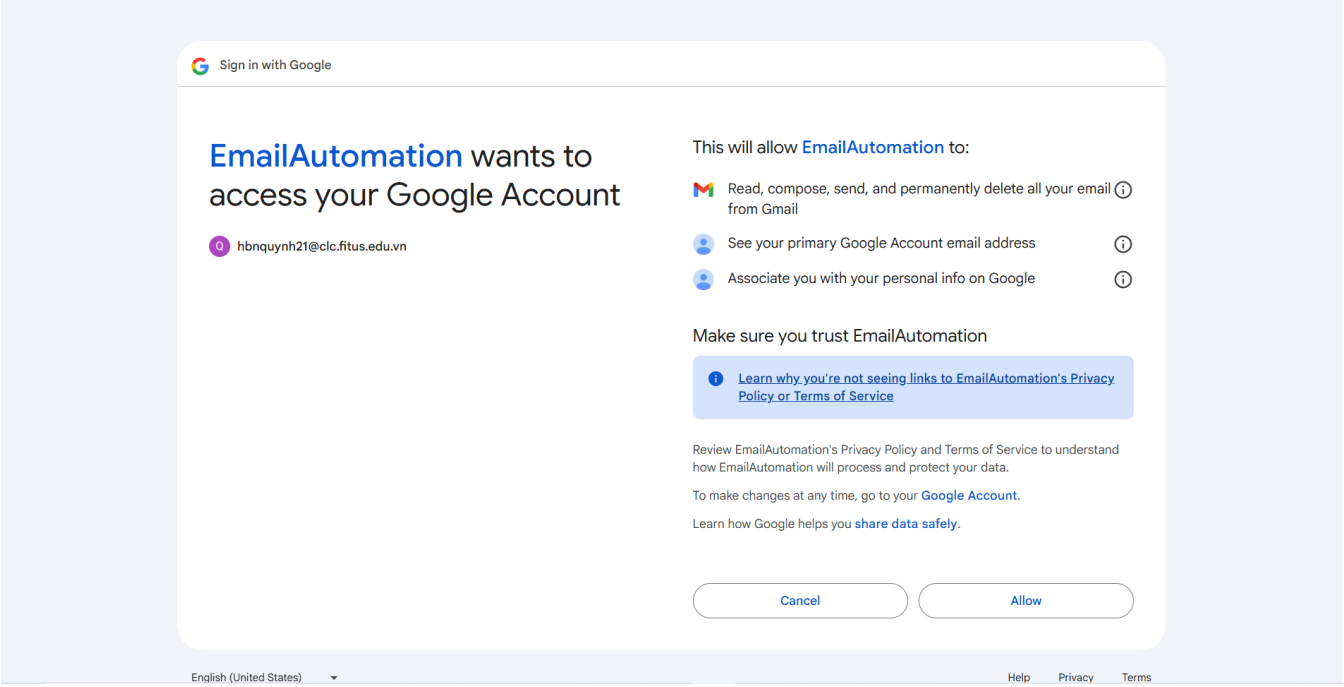
a) Login

- Option 1: Login using Gmail and an App Password *(Note: This method is less convenient as users need to generate an app password manually.)*

- Enter your Gmail address and app password.
- Click the **'Login'** button.

- Option 2: Login with Google

- Click the **'Login with Google'** button. You will be redirected to a Google authentication page.
- Click **'Allow'** to complete the authentication.

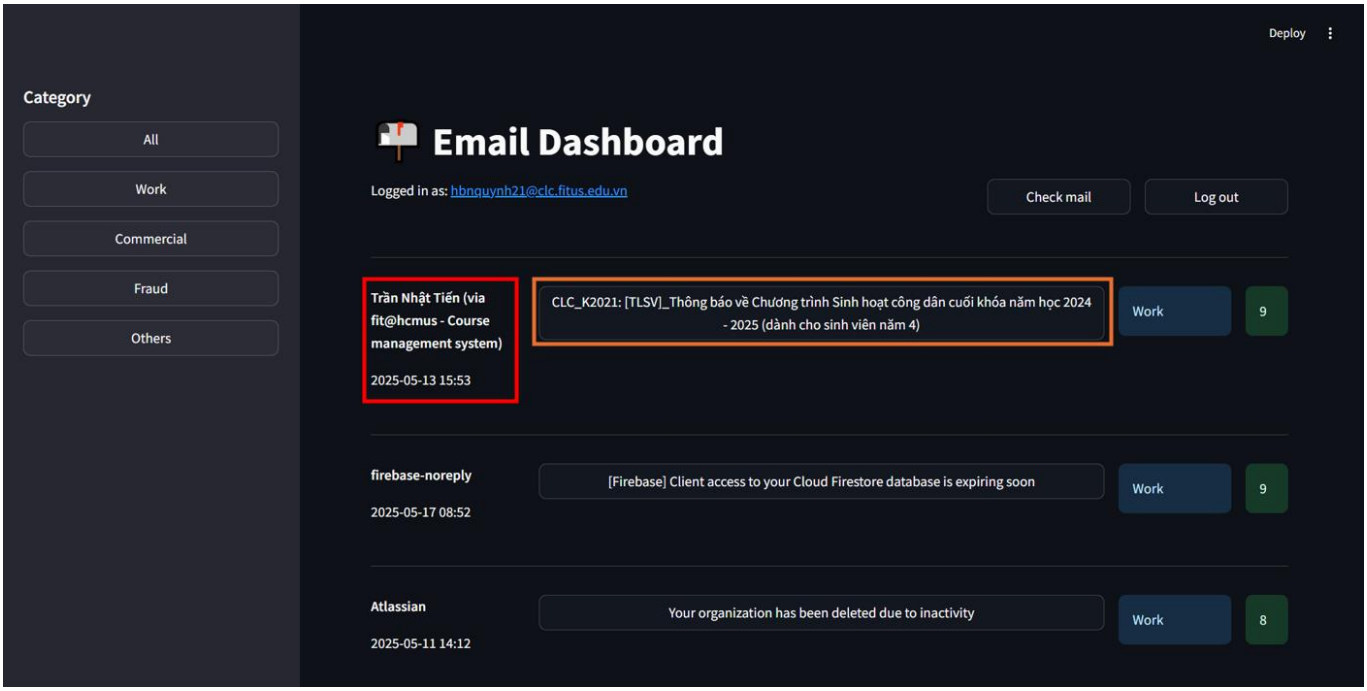


- Once logged in, you can close the authentication tab and return to the application tab to use the features.

b) Auto Email classification and Email prioritization

- These two features are automatically activated after a successful login.

- Once logged in, you will be redirected to the **email dashboard**.



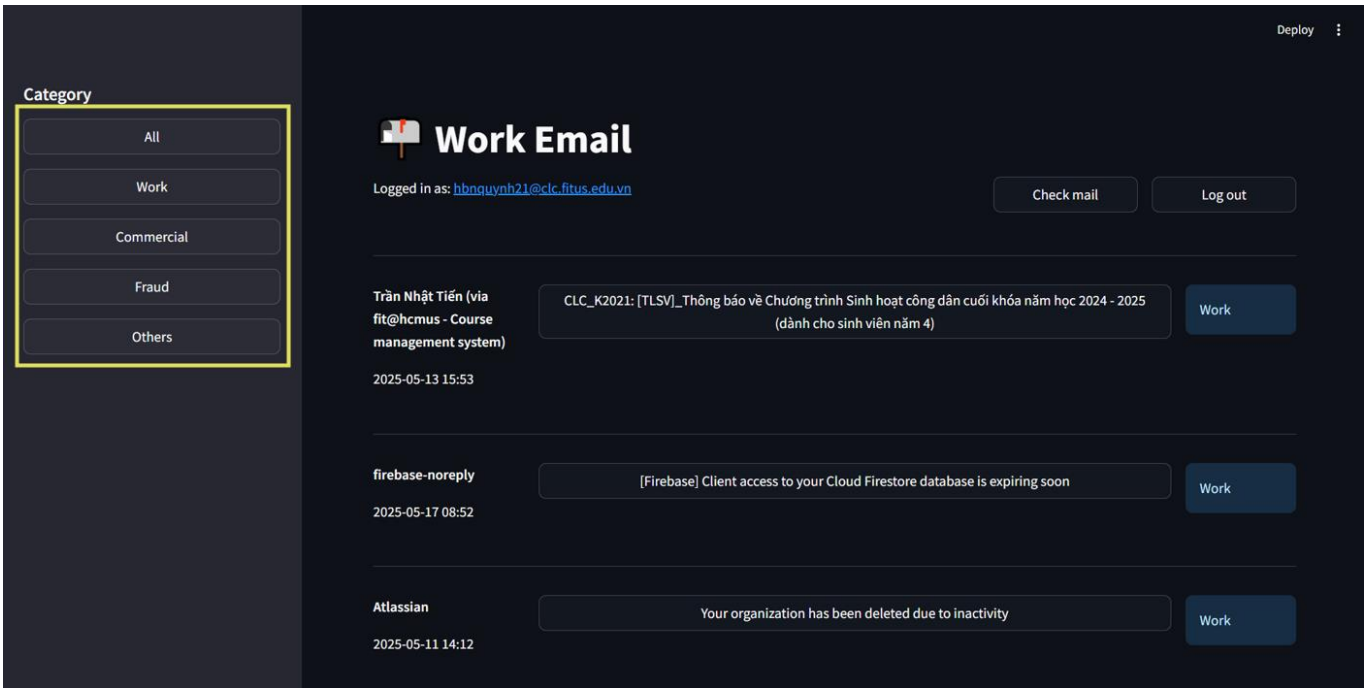
- The dashboard includes the following components:

- **Logged in as**: Your Gmail address
- **'Check Mail' button**: The app will only parse new emails when this button is clicked

- **'Log out' button**
- **Sender and timestamp** (red box)
- **Email subject** (orange box): Click here to view email details
- **Category label** (blue background): The email category automatically classified by Gemini
- **Priority label** (green background): The importance level of the email. Higher-priority emails are displayed at the top

c) View categorized emails

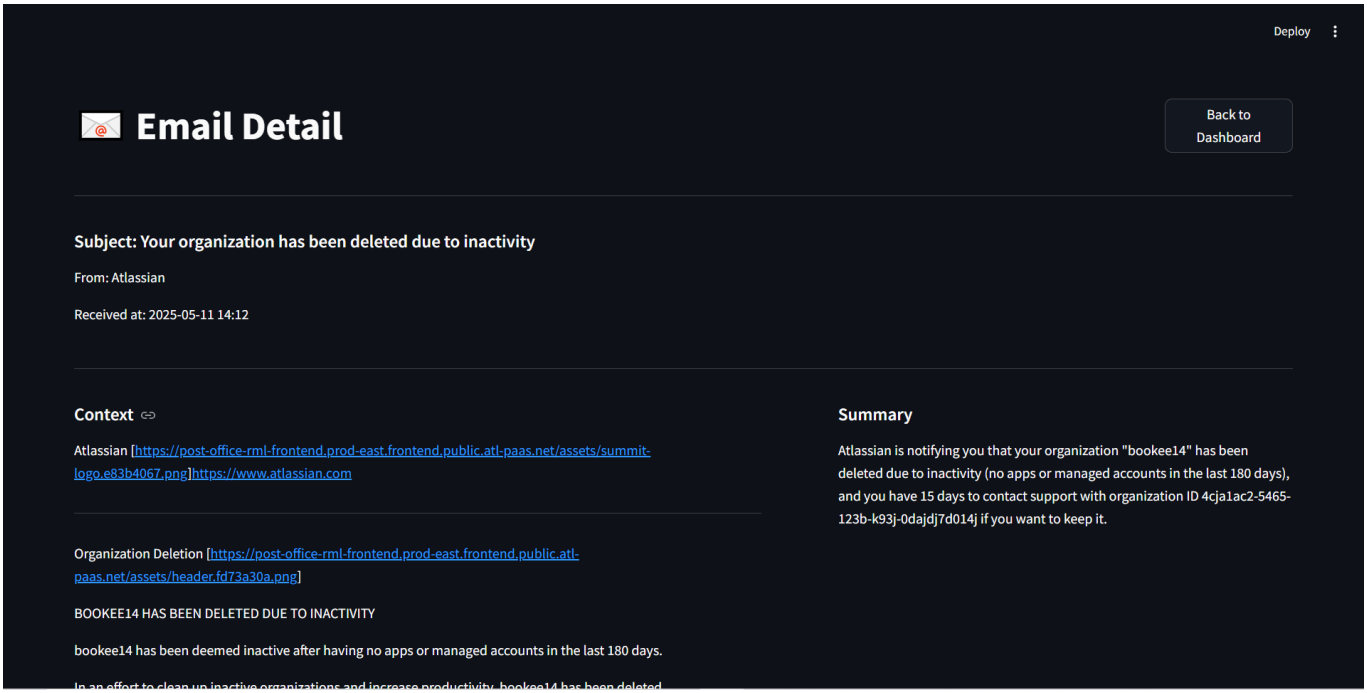
- To filter and view emails by a specific category, click the corresponding label in the **sidebar** (yellow box). Example below shows **Work** emails:



d) Auto summarize email context

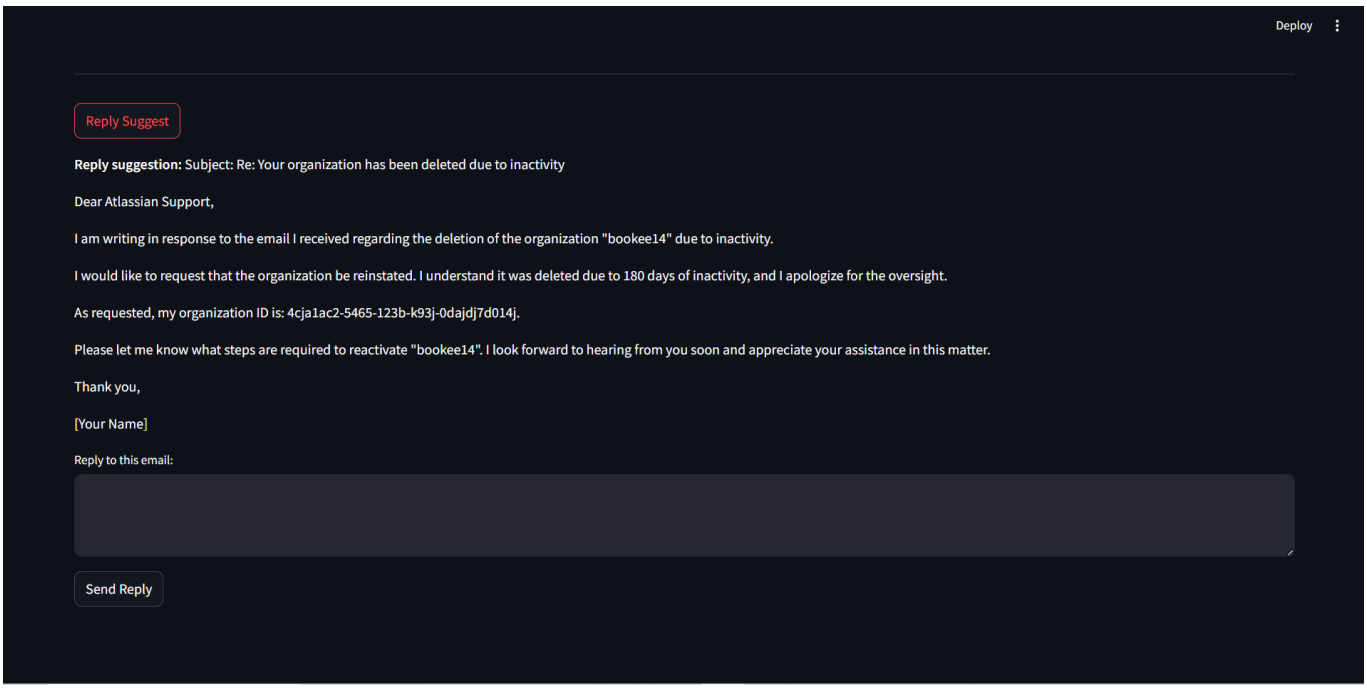
- When clicking on an email subject, you will be taken to the **email detail page**.

- On this page, the AI will summarize the content and display it alongside the original email body.



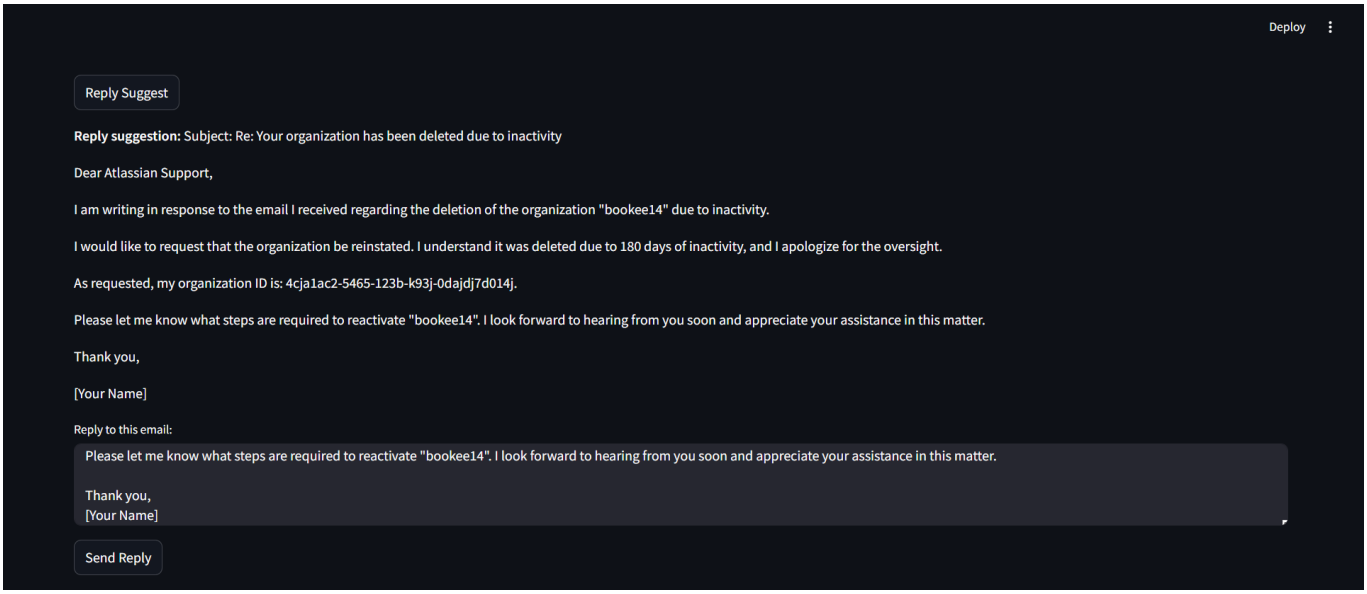
e) AI-based reply suggestion

- When you click the **'Reply Suggest'** button, the AI will generate and display a suggested reply below the email content.



f) Reply email

- Type your reply in the input box
- Click **'Send Reply'** to send your response



7. Limitations

7.1. Known Issues

- Email data for our use case aren't available or do not qualify for our need

- The definition of prioritizing and classifying categories for email varies from user to user → It is impossible to accurately determine user needs to evaluate priority scores or categorize email

7.2. Model and System Limitations

- Gemini API rate limit allows 15 requests/minute for a single Api key free access → lack processing power to process a large amount of email without using multiple Api keys when rate limit reached

- Streamlit library enable developing user interface easily but with quite limitation

- Email data retrieve from Gmail API are complex, which is relatively long to extract for further processing -→ The current system fixes on only handling emails within a week.

- Other factors of email such as seen/unseen/stared/flagged haven't been considered in our system main features

- Email replies haven't been handling carefully for displaying or considered for prioritizing, summarizing and reply suggestion

- The current system hasn't provided other side features such as Mark as seen/unseen, Multiple categories for email or Pin email to put it on top priority...

8. Future Works

8.1. UI/UX enhancements

- **Email tagging**: Use color-coded tags or icons to quickly convey the priority level of each email.

- **Manual email filter**: Allow users to filter and sort emails based on category, priority, or date with a user-friendly interface.

- **Enhance AI suggesting UX**: Display AI-generated reply suggestions in a conversational format, similar to chat bubbles, allowing one-click insertion into the reply box.

- **Suggestion customization**: Allow users to customize suggestions (e.g., tone: formal, friendly, concise) before insertion.

8.2. LLM model integrity

- **Model Switching on Failure:** In the event that the categorization API (or the primary NLP model) encounters 3 consecutive errors (such as timeouts, incorrect response formats, or HTTP 5xx errors), the system will automatically switch to a backup AI model—potentially an older version with fewer features but greater stability—to ensure continued system operation.

- **Alerts & Logging:** Send alerts (via email, Slack, Telegram) to administrators whenever a model switch or any abnormality is detected.

8.3. Deploying a custom machine learning model

- **Developing a proprietary model:** Instead of fully relying on the Gemini API, the team can develop and train their own email classification model, using methods such as BERT, DistilBERT, or specialized Transformer models, leveraging the available labeled dataset.

- **Integrating a CI/CD pipeline:** Set up an automated CI/CD pipeline for NLP so that whenever sufficient new user feedback is received (or the training data is updated), the model is automatically retrained, validated, and deployed to the production environment using pipelines (GitHub Actions, GitLab CI/CD, etc.).

- **Continuous Learning with user feedback:** Allow users to edit category labels and evaluate the accuracy of generated summaries/reply suggestions. This feedback will be stored and periodically used to update and retrain the model (active/continual learning).

8.4. Safe deployment & Model update strategies

- **Canary Deployment (A-B Testing):** When updating a model or releasing a new feature, avoid deploying immediately to all users. Instead, use canary deployment (testing with a small subset of users, e.g., 5%) or A/B testing, closely monitoring metrics such as error rates, user satisfaction, and performance.

- **Rapid Rollback:** If the new model or version encounters serious errors, the system should allow rolling back to a stable version within minutes by using Docker image versioning, Firestore snapshots, or automatically cloning back the previous code/model.

8.5. System expansion & Optimization

- **Adding other email providers:** Redesign the email retrieval module to support integration with providers beyond Gmail (such as Outlook, Yahoo), thereby increasing the potential user base.

- **Optimizing Prompts and NLP Pipeline:** Research and automate the selection of the best prompts (prompt optimization) or use a multi-prompt ensemble approach for challenging cases.

- **Improving UI/UX:** Design the interface to allow users to easily filter, search, and evaluate classification/suggestion results, making it easier to collect natural user feedback.

- Realtime Monitoring & Auto-Healing: Strengthen the real-time monitoring system (using Prometheus, Grafana) and implement automated system-level error recovery (auto-healing containers/pods, automatic restarts, request retries, etc.).

8.6. Effectiveness evaluation & Process standardization

- Regular model evaluation: Set up a regular schedule (monthly/quarterly) to reevaluate the model (accuracy, recall, F1-score, latency) using new test sets or real-world data validated by users.

- Standardizing Logging and Monitoring: Apply unified logging standards (JSON logs, structured logs) and deeply integrate with alert systems (Prometheus Alertmanager, Grafana Notification).

8.7. Security & Privacy

- Data Encryption & Privacy Protection: Ensure that all emails and user data are encrypted (both at rest and in transit), complying with standards such as GDPR or their equivalents.

- Access Auditing: Record detailed logs of all data access, model edits, and label modifications to ensure transparency and facilitate traceability when needed.

9. References

[1] Shantanu, Email classifier, GitHub repository, 2021. [Online]. Available: https://github.com /shxntanu/email-classifier.

[2] Streamlit Documentation. [Online]. Available: https://docs.streamlit.io/

[3] Gmail API Documentation. [Online]. Available: https://developers.google.com/gmail/api

[4] Sandra Gittlen, Ultimate guide to CI/CD: Fundamentals to advanced implementation, 2025. [Online]. Available: https://about.gitlab.com/blog/2025/01/06/ultimate-guide-to-ci-cd-fundamentals-to-advanced-implementation/

[5] Google OAuth documentation [Online]. Available: https://github.com/googleapis/google-api-python-client/blob/main/docs/oauth.md