
Project Scoping Submission -: MailMate – Email Assistant

Team Members:

- Shubh Desai
- Pushkar Kurhekar
- Aalap Desai
- Deep Prajapati
- Shubham Mendapara

---

1. Introduction

In today's fast-paced professional environment, email overload is a major challenge, especially after returning to work after a break or vacation time.

Our Email Assistant aims to alleviate this burden by:

1. Summarizing important messages so the user can quickly catch up upon return.
2. Suggesting a draft reply to an email thread upon the user's return.
3. Identifying action items for the user from the email thread.

This project aims to build an email assistant tool capable of summarizing email threads and suggesting context-aware draft replies. The system uses Natural Language Processing (NLP) techniques to summarize, identify action items, and generate replies. We plan to integrate this tool with at least one popular email service using their respective APIs, and will most probably be a browser extension.

2. Dataset Information

Dataset Introduction

We will be using the **Enron Email Dataset** which is a large corpus of real corporate emails. Despite being somewhat dated, it contains realistic exchanges and conversation threads suitable for training classification and summarization models.

Data Card

Attribute	Details
Name	Enron Email Dataset
Records	~500,000 emails
Size	1.7GB
Format	Mostly text files
Data Types	Plain text emails (subject, body, date, sender, recipient), Text, Metadata
Key Fields	Sender, Recipient, Subject, Body
Language	English
Usage	Train and fine-tune NLP models for email summarization and draft reply generation

Data Sources

- Enron Email Dataset
- URL: https://www.cs.cmu.edu/~enron/
- Additional APIs: Gmail API

Data Rights and Privacy

- The Enron Email Dataset is publicly available for research purposes.
- Privacy Considerations: The data is historical and has been anonymized and re-hosted under open-source or public research licenses.
- We ensure compliance with GDPR-like regulations in case of any personal data.
- The dataset is primarily from defunct accounts and thus does not contain active personal information.
- Real-world user data will only be accessed via API with explicit user consent.

3. Data Planning and Splits

1. **Loading**

    - We will load raw emails from the Enron dataset into a structured format (e.g., CSV, SQL table or a NoSQL DB).
    - Extract relevant information: Sender, Recipient, Timestamp, Subject, Body.
2. **Preprocessing**
    - Remove email footers, signatures, disclaimers, and forward headers.
    - Normalize text (lowercasing, removing extra whitespace, removing HTML tags if any).
    - Extract essential fields like `subject`, `body`, `sender`, `timestamp`.
    - Extract metadata like word count, sentiment, and named entities.
3. **Data Splits**
    - **Train**: ~70% of cleaned emails for model training.
    - **Validation**: ~15% for hyperparameter tuning and checking overfitting.
    - **Test**: ~15% to evaluate final performance.

4. GitHub Repository

GitHub Link: https://github.com/pshkrh/mlops-project

Planned Folder Structure

```
email-assistant/
├── data/
│   ├── raw/
│   ├── processed/
│   └── README.md
├── notebooks/
│   ├── EDA.ipynb
│   ├── classification.ipynb
│   └── summarization.ipynb
├── src/
│   ├── data_preprocessing.py
│   ├── classification_model.py
│   ├── summarization_model.py
│   └── inference_api.py
├── docker/
│   ├── Dockerfile
│   └── docker-compose.yml
```

```
├── tests/
│   ├── test_data_preprocessing.py
│   ├── test_classification_model.py
│   └── test_summarization_model.py
├── README.md
└── requirements.txt
```

5. Project Scope

Problems

- On returning, users must manually skim through a cluttered inbox to find important points about what occurred while they were out. This also applies to old email threads that have a long history.
- Reading through email backlog to find pending tasks / action items for self.
- Having to manually skim through a thread and come up with a reply (if needed) which takes up significant time.

Current Solutions

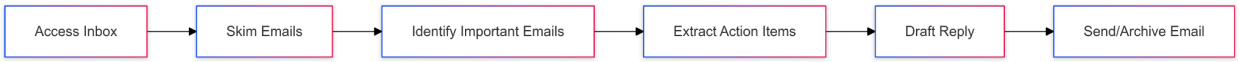
- Microsoft Copilot email summary function works for Outlook but not for other clients. Gmail also has a summarize option. No suggested reply or action item / to-do list items supported in Copilot or Gmail.
- Manual review of emails after an absence.
- Gmail and Outlook both offer auto-complete and some email generation capabilities but require prompting about what to write by the user.

Proposed Solutions

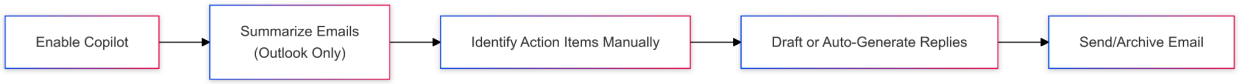
- Summarize each requested email thread into concise bullet points for the user's quick review.
- Generate draft replies for that thread using LLM APIs based on the email thread context to save user time.
- Locate action items from the thread for the current user.

6. Current Approach Flow Chart and Bottleneck Detection

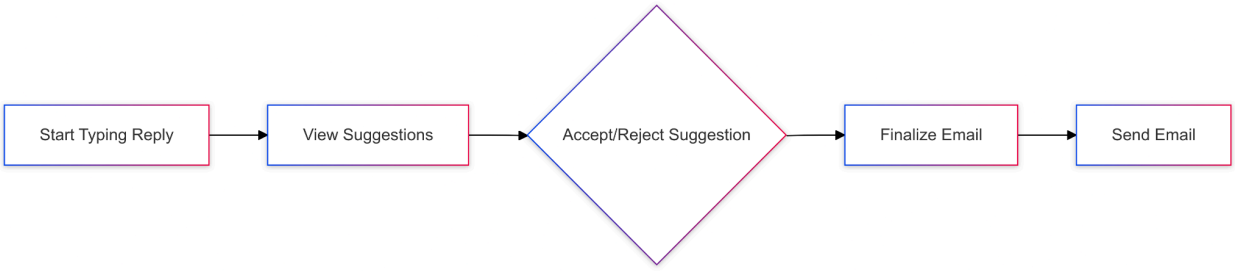
1. **Manual Review of Emails**



2. **Microsoft Copilot for Outloo**k



3. Auto-Complete in Gmail/Outlook



Potential Bottlenecks:

Manual Review of Emails:

- **Time-Consuming:** Users spend significant time skimming through email threads, identifying important information, and drafting responses.
- **Error-Prone:** Action items or critical details might be missed during manual review.

Microsoft Copilot for Outlook:

- **Limited Integration:** Works only for Outlook, leaving users of other email clients without support.
- **Incomplete Functionality:** Lacks features like actionable item detection or comprehensive draft suggestions.

Auto-Complete in Gmail/Outlook:

- **User Dependency:** Requires users to know what to type to trigger useful suggestions.
- **Limited Contextual Understanding:** Suggestions are generated without considering the entire email thread's context.

Possible Improvements:

- **Automate Email Skimming:** Implement automated email summarization to extract concise bullet points, highlighting key information for the user.
- **Action Item Extraction:** Use NLP models to identify actionable items and tasks from the email body, reducing the user's need to manually search.
- **Reply Context Awareness:** Train models to generate contextually accurate and complete draft replies using the full email thread.

7. Metrics, Objectives, and Business Goals

Key Metrics

1. **Summarization Quality**
    - Metric: ROUGE-N, ROUGE-L (Recall-Oriented Understudy for Gisting Evaluation)
    - Objective: Ensure summaries accurately capture essential information with high relevance and coherence.
    - Threshold: Target a ROUGE-1 score of 0.7 or higher for critical summaries.
2. **Action Item Extraction**
    - Metric: Precision, Recall, F1 Score
    - Objective: Accurately extract action items from email threads.
    - Threshold: Achieve at least 80% precision and 75% recall.
3. **User Satisfaction for Suggested Replies**
    - Metric: User feedback via thumbs-up/down ratings
    - Objective: Ensure that users are satisfied with at least 80% of the generated drafts or summaries.
    - Threshold: Positive feedback rate of 80%+.
4. **Efficiency**
    - Metric: Processing time per email/thread (measured in milliseconds or seconds)
    - Objective: Ensure system processes emails efficiently, averaging 10-15 seconds per email/thread.
    - Threshold: Achieve an average processing time of 10-15 seconds.

Objectives

1. Minimize time spent by users skimming emails by providing actionable summaries and drafts.
2. Achieve a notable reduction in the average time taken to clear a backlog of emails, or even a single email thread per user’s request.
3. Integrate the tool with at least one major email platform (e.g., Gmail, Outlook) with minimal user friction.
4. Increase user productivity by helping them focus on critical action items instead of sorting through email clutter.
5. Maintain user trust by adhering to data privacy standards.

6. Implement GDPR-compliant measures, anonymize personal data, and minimize data storage requirements.

Business Goals

1. **Enhance User Productivity**
    a. Provide concise, actionable email summaries and draft responses, helping users save time and quickly address important matters upon returning from an absence.
    b. Streamline the process of managing email backlogs to foster a smoother workflow and reduce stress.
2. **Improve Email Management Efficiency**
    a. Assist with routine tasks like summarizing threads, detecting action items, and drafting replies, allowing users to focus on higher-priority tasks instead of manually reviewing emails.
    b. Reduce the average time taken to process a large volume of emails post-absence.
3. **Increase User Satisfaction**
    a. Deliver personalized, context-aware email suggestions and summaries, ensuring users feel supported and confident in handling their inbox effectively.
    b. Build trust in the tool by maintaining accuracy and relevance in generated outputs.
4. **Provide Actionable Insights**
    a. Identify and present critical action items from email threads, enabling users to prioritize and address pending tasks more effectively.
5. **Ensure Seamless Integration**
    a. Integrate the tool with at least one popular email platform (e.g., Gmail, Outlook) to ensure ease of use and minimal disruption to users' existing workflows.
    b. Design a user-friendly interface for smooth adoption of the tool.
6. **Maintain Data Privacy and Ethics**
    a. Adhere to data privacy regulations by using secure APIs and anonymized datasets, ensuring user trust and compliance with academic standards.
    b. Emphasize ethical use of data and transparency in processing email content.

8. Failure Analysis

Potential Risks and Mitigation Strategies

1. **Summarization Failure**
    - **Issue:** Summaries may fail to capture key details or could include irrelevant or hallucinated content.

- **Mitigation:**
  - Implement user feedback loops for continuous improvement.
  - Use a fallback mechanism to generate extractive summaries when abstractive summarization fails.

2. **Action Item Detection Errors**
   - **Issue:** Incorrect or incomplete extraction of action items may lead to missed tasks or misunderstandings.
   - **Mitigation:**
     - Allow users to manually review and modify extracted action items for better accuracy.
     - Implement confidence score thresholds for action item extraction to filter out low-confidence results.
3. **LLM API Failure or Blocked requests**
   - **Issue:** The primary LLM API might be down or is blocking the tool's requests due to rate limits or API Key issues.
   - **Mitigation:**
     - Have fallback APIs to different LLMs to handle the failed requests with multiple retry attempts.
4. **Data Privacy Breach**
   - **Issue:** Processing or storing sensitive emails may result in legal consequences or loss of user trust.
   - **Mitigation:**
     - Anonymize user data before processing to minimize the exposure of sensitive information.
     - Employ encrypted communication channels for data transfer and limit data retention periods to comply with privacy laws (e.g., GDPR).

Pipeline Failures

1. **Email Data Ingestion Failures**
   a. **Description**: Failures during email data ingestion may occur due to network issues, API changes, authentication errors, or unavailable data sources.
   b. **Mitigation**:
      i. Implement retry mechanisms with exponential backoff and maximum retry limits.
      ii. Validate incoming data for completeness and format before processing.
      iii. Set up proactive error logging, monitoring, and alerting to track the health of the data ingestion process.
2. **Model Training/Inference Failures**

a. **Description**: Model training or inference failures can occur due to data quality issues, misconfigurations, or deployment pipeline problems, leading to incorrect predictions or outdated models.
b. **Mitigation**:
    i. Use model version control to ensure stable and easily retrievable versions.
    ii. Implement automated retraining pipelines to keep models updated.
    iii. Continuously monitor model performance with metrics like accuracy and precision to ensure reliable outputs.

3. **ETL/Preprocessing Failures**
    a. **Description**: Failures in ETL or data preprocessing can arise from errors in transformation logic or missing data.
    b. **Mitigation**:
        i. Implement data validation and quality checks at every stage of the ETL process.
        ii. Modularize the ETL pipeline into smaller, independent components to isolate failures.
        iii. Maintain detailed logging for each ETL step and set up alerts to notify when failures occur.

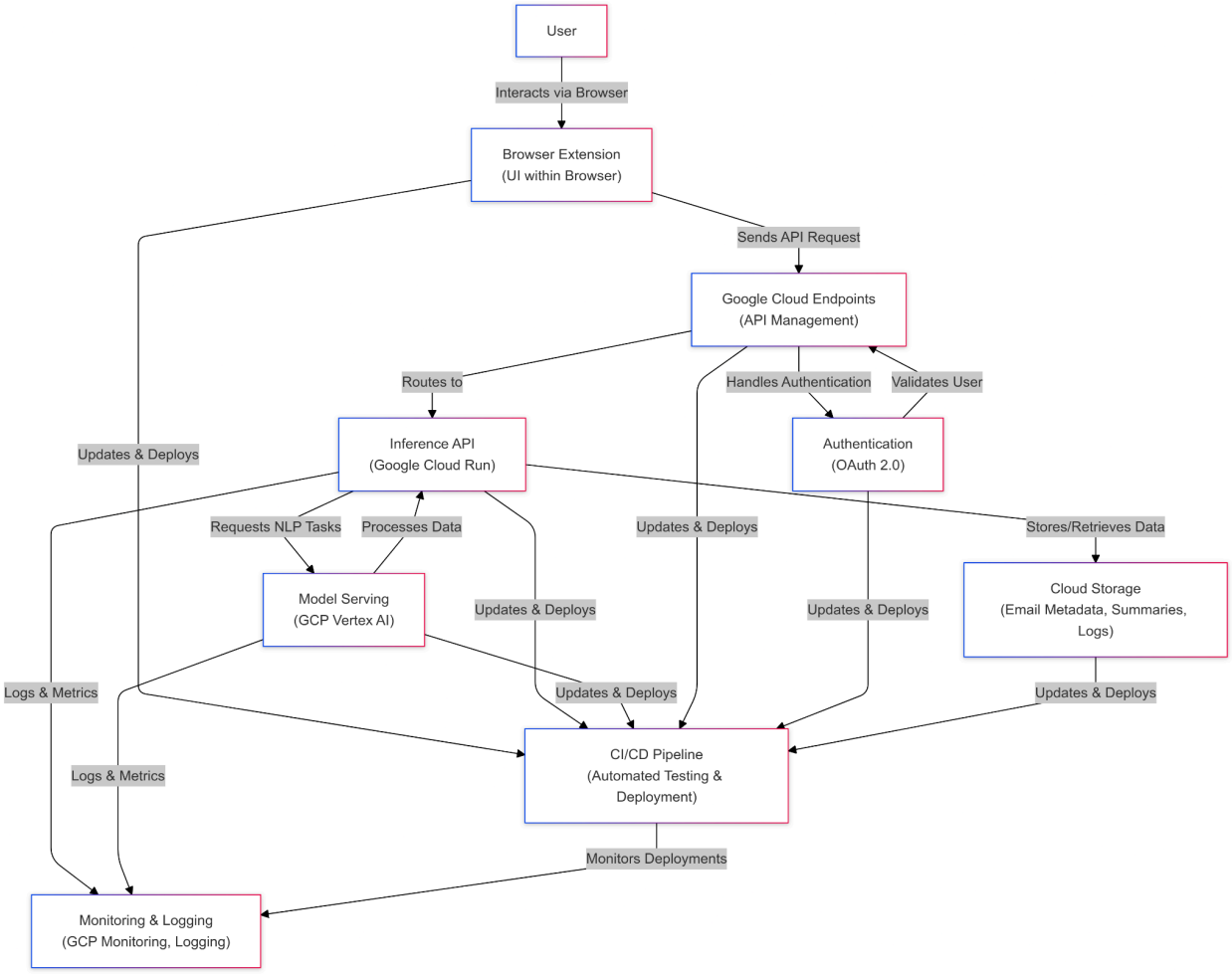
9. Deployment Infrastructure

Overview

Key components of our infrastructure include:

- **Browser Extension**: User interface within the browser.
- **API**: Handles requests and processes data.
- **Model Serving**: Executes NLP tasks for summarization, action item extraction, and draft reply generation.
- **Data Storage**: Securely stores email metadata and processed information.
- **CI/CD Pipeline**: Automated testing and deployment.
- **Monitoring & Logging**: Ensures system performance and reliability.

Reference Architecture



Components

1. **Browser Extension:** Provides the UI for users to interact with their email client. Initiates API requests when the user clicks the extension button.
2. **Google Cloud Endpoints:** Manage API requests from the browser extension, handles authentication, and routes traffic to backend services.
3. **Authentication:** Manage user authentication via OAuth 2.0, ensuring secure access to email data. Connects with Gmail API and other email services securely.
4. **Inference API (Google Cloud Run):** Host the backend application built with FastAPI or Flask to handle summarization, action item extraction, and draft reply generation.
5. **Model Serving (GCP Vertex AI):** Deploy and serve NLP models for processing email data. Can utilize pre-trained models or custom-trained models for specific tasks. Will also be used for calling LLM APIs.

6. **Cloud Storage:** Stores user data, email metadata, processed summaries, action items, and logs.

Deployment Environments

1. **Development:** Perform local development using Docker for containerization. Use Docker Compose to simulate the full stack.
2. **Staging:** Deploy to a GCP project environment mirroring production for final testing and quality assurance before production release.
3. **Production:** Host on Cloud Run for high availability, auto-scaling, and robust security measures.

CI/CD Pipeline

- **Version Control**: GitHub repository https://github.com/pshkrh/mlops-project
- **GitHub Actions**: Automated testing, linting, and building Docker images on each commit.
- **Google Cloud Build:** Deploys to Cloud Run or GKE upon successful CI pipeline completion. Manual approval step for production deployments to ensure stability.

Monitoring & Logging

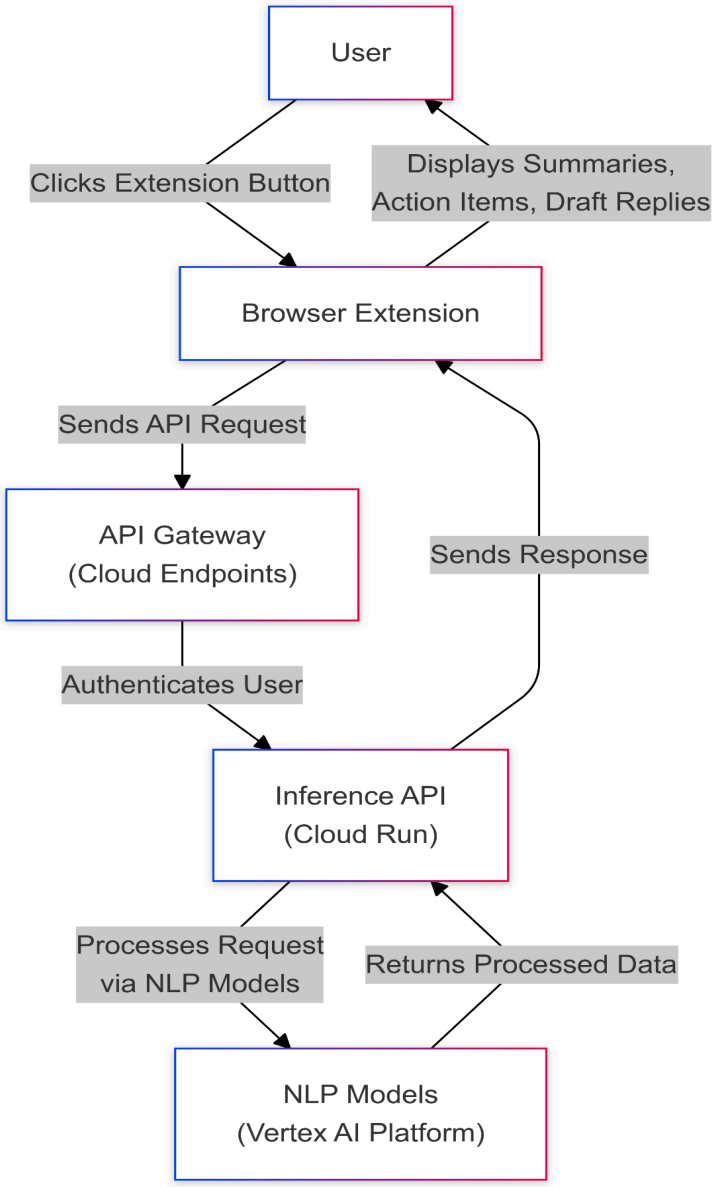
- **Monitoring**:
  - **Service**: **Google Cloud Monitoring**
  - **Metrics**: API response times, error rates, system health, and model performance.
  - **Alerts**: Set up alerts for performance degradation or system failures.
- **Logging**:
  - **Service**: **Google Cloud Logging**
  - **Functionality**: Centralized logging for debugging and audit trails.

Scaling & Caching

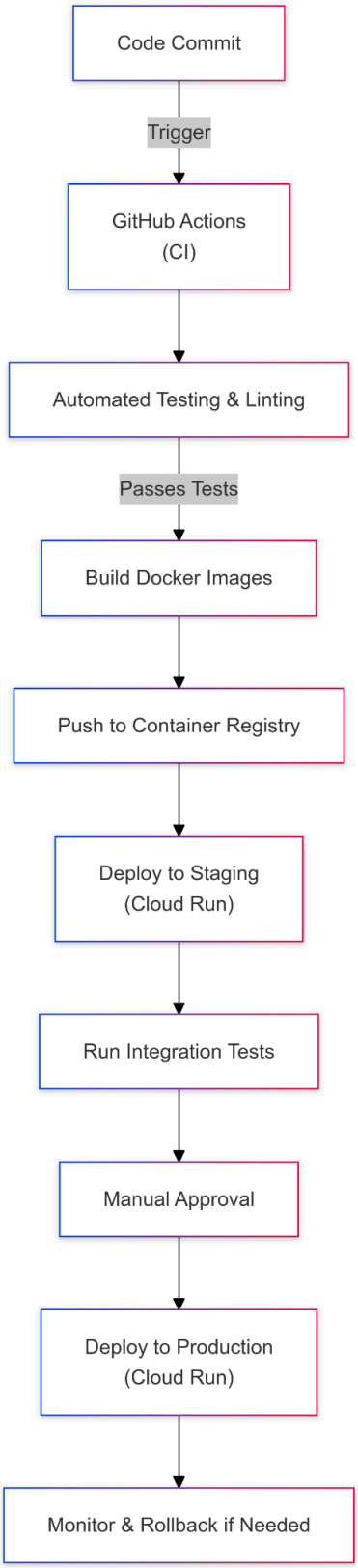
- **Auto-Scaling**: Cloud Run automatically scales based on incoming traffic.
- **Caching**: Google Cloud Memorystore (Redis). Can cache frequent or previous requests in a certain timeframe (possibly 1 week) to reduce latency and lower costs.

Flowcharts

1. User Interaction Flow:



2. CI/CD Pipeline Flow:


Code Commit
Trigger
GitHub Actions
(CI)
Automated Testing & Linting
Passes Tests
Build Docker Images
Push to Container Registry
Deploy to Staging
(Cloud Run)
Run Integration Tests
Manual Approval
Deploy to Production
(Cloud Run)
Monitor & Rollback if Needed

Supported Platforms

- **Email Clients**: Gmail, Outlook (future).
- **Browsers**: Google Chrome and other Chrome-based browsers.

10. Monitoring Plan

Monitoring Metrics

System Metrics

- **Infrastructure Health**:
    - CPU/Memory/Storage usage (GCP Monitoring).
    - API latency and request rates.
- **Error Tracking**:
    - Log errors to detect and respond to failures.
- **Response time for API calls**
    - To maintain a smooth and seamless user experience by minimizing delays in processing and delivering results.
- **API uptime and error rates**
    - To ensure system reliability and availability, reducing downtime and maintaining consistent performance for users.

Model Performance Metrics

- **Summarization**:
    - ROUGE-1, ROUGE-L, and BLEU scores.
    - To evaluate the quality and relevance of email summaries, ensuring that they capture critical information accurately and concisely.
- **Action Item Extraction**:
    - Precision/Recall for tasks identified from threads.
    - To verify that key tasks or responsibilities are correctly extracted from emails, helping users act efficiently without missing important details.
- **Acceptance rates of auto-generated reply drafts**
    - To gauge user trust and satisfaction with suggested responses, refining the model to align better with user needs and preferences.

Alerting and Thresholds

Alert if:

- API response time exceeds 15 seconds.
- Error rate increases beyond 5% of total requests.

Monitoring ensures:

i. **Reliability**: The system is up and running, responding within acceptable time limits.
ii. **Performance**: Model inference isn't slowing down under load.
iii. **Accuracy & Quality**: Summaries, classifications, and action item extractions remain at or above defined performance thresholds.
iv. **User Satisfaction**: The product continues to solve actual problems without creating new ones (e.g., incorrect summaries or slow response times).
v. **Concept Drift Detection**: Early detection of concept drift if email language changes or new topics emerge.

Feedback Loop and Continuous Improvement

- Use real-world feedback to iteratively fine-tune models.
- Introduce user-reported corrections into the training pipeline for better summaries.

11. Success and Acceptance Criteria

Success Criteria:

1. Summarization Accuracy

- **Metric**: ROUGE-1 Score
- **Objective**: Generate concise and meaningful summaries of email threads.
- **Threshold**: Attain a **ROUGE-1 score of 0.7 or higher**, indicating summaries effectively capture the essential information.

2. Action Item Detection

- **Metric**: Precision and Recall

- **Objective**: Identify actionable tasks from email threads for user convenience.
- **Threshold**:
    - Precision: **70% or higher**.
    - Recall: **70% or higher**.

4. Processing Efficiency

- **Metric**: Average Processing Time
- **Objective**: Ensure smooth and efficient user experience when processing emails.
- **Threshold**:
    - Average processing time for summarization, identifying action items and auto-generating reply drafts: 10-15 seconds per email thread.

5. Integration with existing email providers

- **Metric**: Successful integration with at least one email provider
- **Objective**: Integrate and work seamlessly with Gmail and/or other email services.
- **Threshold**: Demonstrate functional integration by accessing and processing email data securely and efficiently.

Acceptance Criteria:

1. **Functional Reliability**
    - The app / tool performs email summarization, classification, and reply draft generation reliably:
        - No major crashes or errors during normal usage.
        - Accurately provides summaries and action items, and draft replies for at least 80% of user emails.

2. **Privacy and Security Compliance**
    - User data is processed securely in compliance with data protection regulations such as GDPR:
        - User consent is obtained before accessing email data.
        - Sensitive information is anonymized, encrypted, and not stored unnecessarily.

12. Timeline Planning

Data Preparation

- **Week 1**: Data acquisition, initial EDA.
- **Week 2**: Clean and preprocess data.
- **Week 3**: Label and engineer features. Finalize the dataset and establish baseline metrics.

Modeling

- **Week 4:** Train/fine-tune initial models for summarization. Set up LLM APIs for action item identification and auto-generated draft replies.
- **Week 5**: Fine-tune models and engineer prompts for better performance and address edge cases.
- **Week 6**: Conduct error analysis, refine models, and validate with simulated emails. Start setting up the front-end UI.

Deployment and Testing

- **Week 7**: Integrate backend and front-end UI.
- **Week 8**: Set up MLOps infrastructure (CI/CD, Docker, monitoring).
- **Week 9**: Conduct rigorous end-to-end testing and gather user feedback.
- **Week 10**: Refactor code, finalize UI/UX, and prepare a live demo.