Gen AI-Based Email Classification and OCR

This document outlines the proposed solution for automating the processing of service requests received by Commercial Bank Lending teams via email. It addresses the requirements detailed in the "Technology Hackathon - Early Talent Gen AI-Based Email Classification and OCR" and leverages the approach and solution implemented in the provided Python scripts.

1. Introduction

- Problem: Commercial Bank Lending teams face a high volume of diverse service requests via email, demanding significant manual effort for triage, classification, and data extraction. This manual process is time-consuming, error-prone, and inefficient, especially with large volumes.

- Solution: A Gen AI-Based Email Classification designed to automate the classification of incoming email requests and the extraction of relevant data from both the email body and attachments. This will minimize manual intervention, improve accuracy, and accelerate turnaround times.

2. Business Requirements

- Accurate Extraction and Categorization: The system must accurately extract and categorize information based on the content and context of emails and attachments, understanding the sender's intent.
- Handling Structured Data: The solution should effectively process structured data present in both email bodies and various attachment formats.
- Primary Request Intent Identification: The system needs to identify the primary request intent, even in emails containing multiple requests.
- Prioritized Customizable Extraction: Prioritized extraction of user-defined key fields from relevant documents is essential.
- Duplicate Request Identification: The solution should identify and flag duplicate inbound service requests to prevent redundant processing.

3. Proposed Solution Architecture and Components

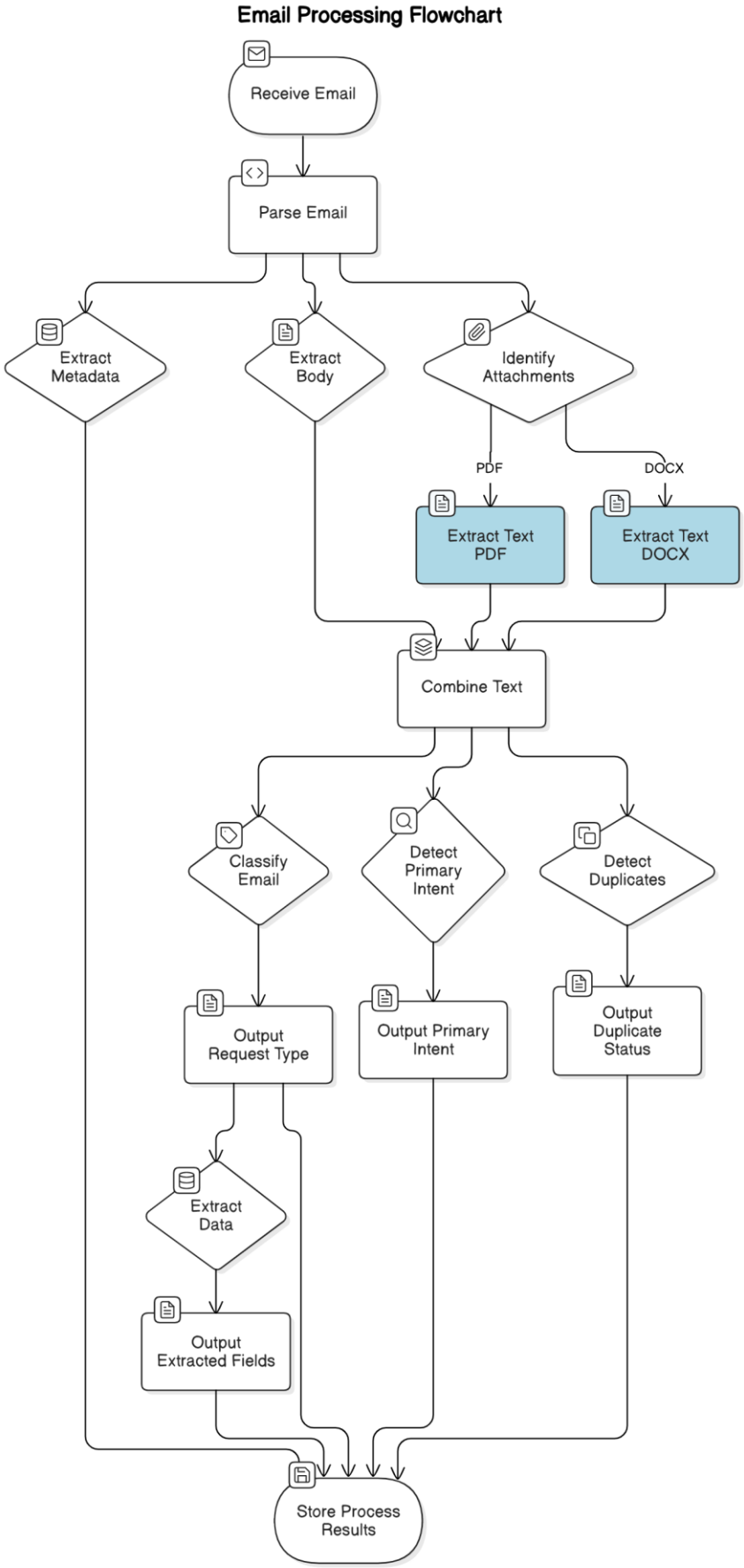
- Email Reception and Parsing:
    - The system will process emails (currently reading from .eml files as demonstrated in main.txt using the email library).
    - It will extract key metadata: **sender, subject, and body**.
    - **Attachments** will be identified and saved.
- Attachment Processing:

    - **PDF documents:** Text will be extracted using the PyPDF2 library.
    - **DOCX documents:** Text will be extracted using the docx2txt library.
- Text Extraction and Combination: The text from the email body and all processed attachments will be combined into a single text input for the Gen AI model.
- Email Classification (using Google Gemini Pro):
    - The solution leverages the **Google Gemini Pro model** (implemented using google.generativeai in main.py) for classifying emails into predefined request types.
    - The classify_email function takes the combined email text and a list of request_types (e.g., "Adjustment", "Fee Payment") as input.
    - The expected output includes the **request type, sub-request type, reasoning, and a confidence score**.
- Primary Intent Detection (using Google Gemini Pro):
    - The detect_primary_intent function in main.txt utilizes Gemini Pro to determine the main intent of the email content.
- Data Extraction (using Google Gemini Pro):
    - The extract_data function in main.txt uses Gemini Pro to extract specific, relevant fields (e.g., "Deal Name", "Amount", "Expiration Date") from the email text based on the identified request_type. If a field is not found, it returns "N/A".
- Duplicate Email Detection:
    - The detect_duplicate_emails function in duplicate_detector.txt and main.txt employs TF-IDF vectorization and cosine similarity (using sklearn) to identify emails with high textual similarity.
    - A threshold (e.g., 0.8) is used to flag potential duplicate emails.

4. Technical Components and Tools

- Programming Language: Python
- Email Handling: email standard library
- PDF Text Extraction: PyPDF2 library
- DOCX Text Extraction: docx2txt library
- Gen AI Model: **Google Gemini Pro** (accessed via google.generativeai)
- NLP Libraries: scikit-learn (sklearn) for TF-IDF and cosine similarity
- Environment Management: dotenv for managing API keys
- Logging: logging module for error handling and warnings

5. Detailed Architecture and Workflow



6. Solution Workflow

- Receive Email (.eml file): The process begins with receiving an email, which in the current implementation is read from an .eml file.
- Parse Email: The email is parsed to separate its components.
- Extract Metadata: Sender, Subject: Key information like the sender's address and the email's subject are extracted.
- Extract Body Text: The main textual content of the email is retrieved.
- Identify and Save Attachments?: The system checks if the email has any attachments.
- Process Attachments: If attachments exist, they are further processed.

  Is Attachment PDF?: The system checks if an attachment is a PDF file.

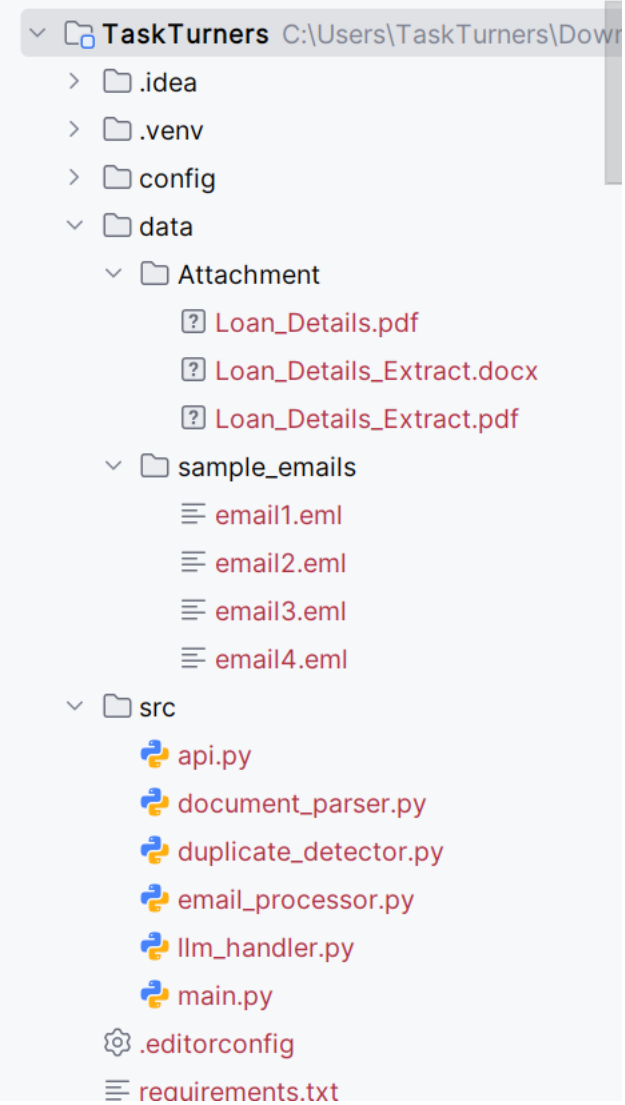
  Extract Text using PyPDF2: If it's a PDF, the text is extracted using the PyPDF2 library.

  Is Attachment DOCX?: The system checks if an attachment is a DOCX file.

  Extract Text using docx2txt: If it's a DOCX, the text is extracted using the docx2txt library.
- Combine Email Body Text and Attachment Text (if any): The text extracted from the email body and any processed attachments is combined into a single text.
- Use Email Body Text: If there are no attachments, only the email body text is used for further processing.
- Classify Email using Google Gemini Pro: The combined text is sent to the classify_email function, which uses the Google Gemini Pro model to classify the email into predefined request types. The output includes the request type, sub-request type, the reasoning behind the classification, and a confidence score.
- Output: Request Type, Sub-Request Type, Reasoning, Confidence Score: The classification results are obtained.
- Detect Primary Intent using Google Gemini Pro: The combined text is also sent to the detect_primary_intent function, which uses Google Gemini Pro to determine the main intent of the email.
- Output: Primary Intent: The primary intent of the email is identified.
- Extract Data using Google Gemini Pro (based on Request Type): Based on the classified request type, the combined text is sent to the extract_data function, which uses Google Gemini Pro to extract specific, relevant data fields (e.g., Deal Name, Amount, Expiration Date).
- Output: Extracted Fields (e.g., Deal Name, Amount, Expiration Date): The extracted data fields are obtained.
- Perform Duplicate Email Detection (Optional): The text content of the processed email can be compared with other processed emails using TF-IDF and cosine similarity to detect potential duplicates. This step is currently implemented to run on a list of processed emails.
- Output: List of Potential Duplicate Emails with Similarity Score: If duplicates are detected, a list of the duplicate pairs and their similarity scores is generated.
- Store/Process Results: Finally, all the extracted information (sender, subject, classification details, primary intent, extracted data, and duplicate information if detected) is stored or further processed as needed.

7. GitHub Code Repository Structure

- The code for this solution organized in a GitHub repository with the following structure:



- **config:** This directory houses configuration files.
  - **. env:** This file is used to store environment variables, such as API/ GOOGLE_API_KEY.
- **data:** This directory contains data files used by the application.
  - **sample_emails:** This subdirectory holds sample email files (e.g., email1.eml) that can be used for testing and demonstration purposes.
- **src:** This is the primary directory containing the source code of the application.
  - **api.py:** This script contains all functions for extracting text from different document types, such as PDFs and DOCX files. It uses libraries like PyPDF2 and docx2txt for this purpose. This code defines a **Flask web application** designed to **process emails** and their attachments. It includes functionalities to **extract text from various document formats (like PDF and DOCX)**, **classify emails and extract data using the Gemini API**, and **detect duplicate emails** based on text similarity. The application exposes API endpoints to process a single email and to detect duplicate emails from a list of email texts.

- **document_parser.py:** This script likely contains functions for extracting text from different document types, such as PDFs and DOCX files. It uses libraries like PyPDF2 and docx2txt for this purpose.
- **duplicate_detector.py:** This script implements functionality to detect duplicate emails. It uses techniques like TF-IDF and cosine similarity from the sklearn library to calculate the similarity between email texts. A similarity threshold (e.g., 0.8) is used to identify potential duplicates.
- **email_processor.py:** This file might contain functions for reading and processing email files (.eml). It likely uses the email library to parse email content, extract the body and attachments, and potentially save attachments.
- **llm_handler.py:** This script interacts with a Large Language Model (LLM), specifically the Gemini model via the langchain_google_genai library.
- **main.py:** This is likely the main entry point of the application. It orchestrates the overall workflow, including reading emails, processing attachments, classifying emails, extracting data, and detecting duplicates. It uses functions from the other modules to perform these tasks.

- **requirements.txt:** This file lists the Python packages and their versions that are required to run the application. Users will need to install these dependencies using **pip install -r requirements.txt**.

8. GitHub Code Repository Details

ewfx/gaied-task-turners: Repository for team: gaied-task-turners

9. Installation Instructions (Pip Installs)

To run the solution, for this first create a virtual environment using below command.
Navigate to the project directory and use terminal and run the following command:
python -m venv .venv

Also, please install the necessary Python packages

pip install -r requirements.txt
The requirements.txt file contains the following dependencies:

PyPDF2
docx2txt
scikit-learn
langchain-google-genai
python-dotenv
google-generativeai

And, create/declare an environment variable in the system running:

Variable Name = GOOGLE_API_KEY

Variable Value = AIzaSyBYdyPDdqP34b0Q4blSZJxbNcExdxXWV9s

10. Test Details

1. email1.eml (Email with no attachment)
   gaied-task-turners/code/test at main · ewfx/gaied-task-turners
2. email2.eml (Email with attachment)
   gaied-task-turners/code/test at main · ewfx/gaied-task-turners
3. email3.eml (Email with attachment)
   gaied-task-turners/code/test at main · ewfx/gaied-task-turners
4. email4.eml (Email with Docx attachment)
   gaied-task-turners/code/test at main · ewfx/gaied-task-turners

For api.py testing use below command to POST the request.

curl -X POST -F "email_file=@C:/Users/TaskTurners/Downloads/TaskTurners/data/sample_emails/email1.eml" http://127.0.0.1:5000/process_email

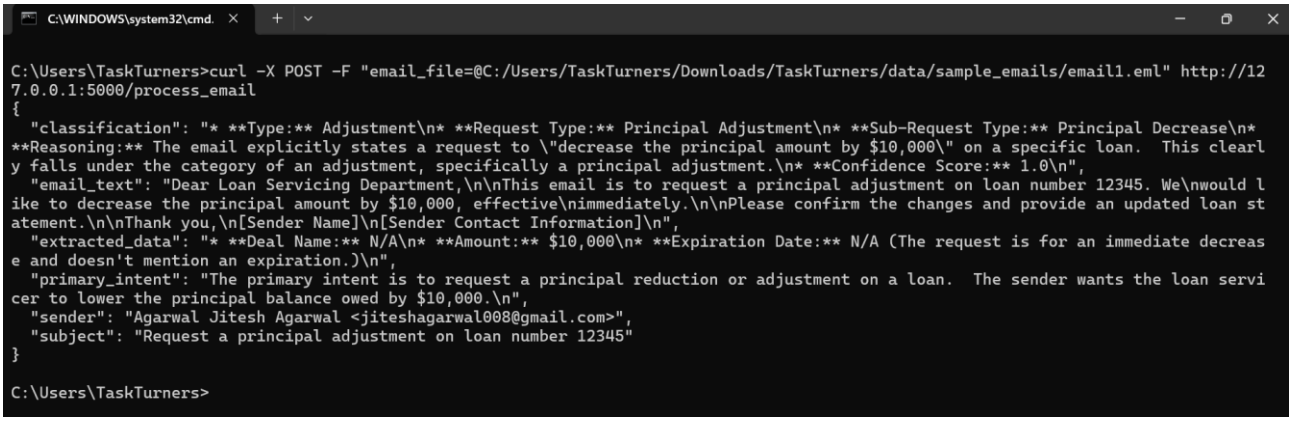
11. Outputs (via **main.py** testing)

email1.eml (Email with no attachment)

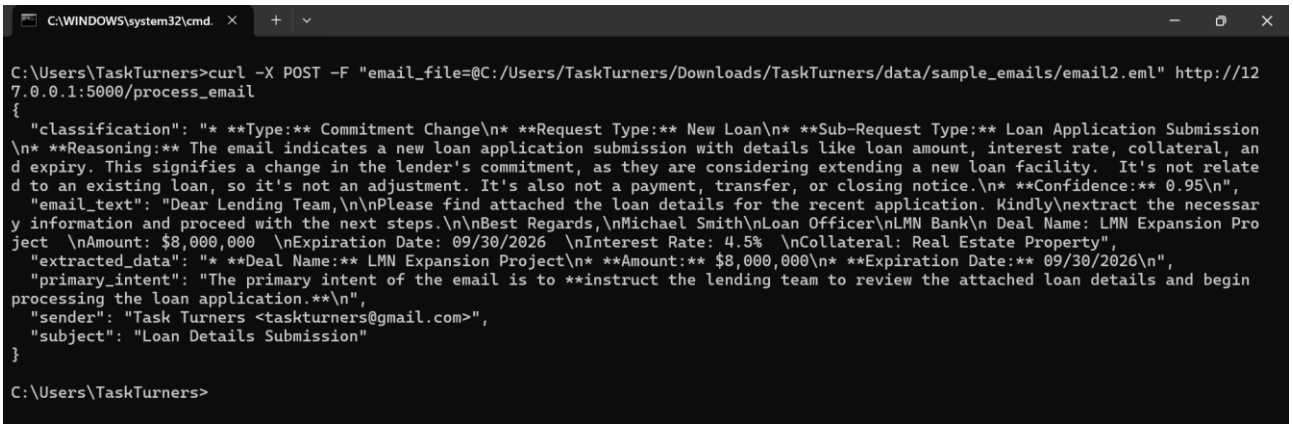
*Set email_files = ["C:/Users/TaskTurners/Downloads/TaskTurners/data/sample_emails/**email1.eml**"]*

12. Outputs (via **api.py** testing)

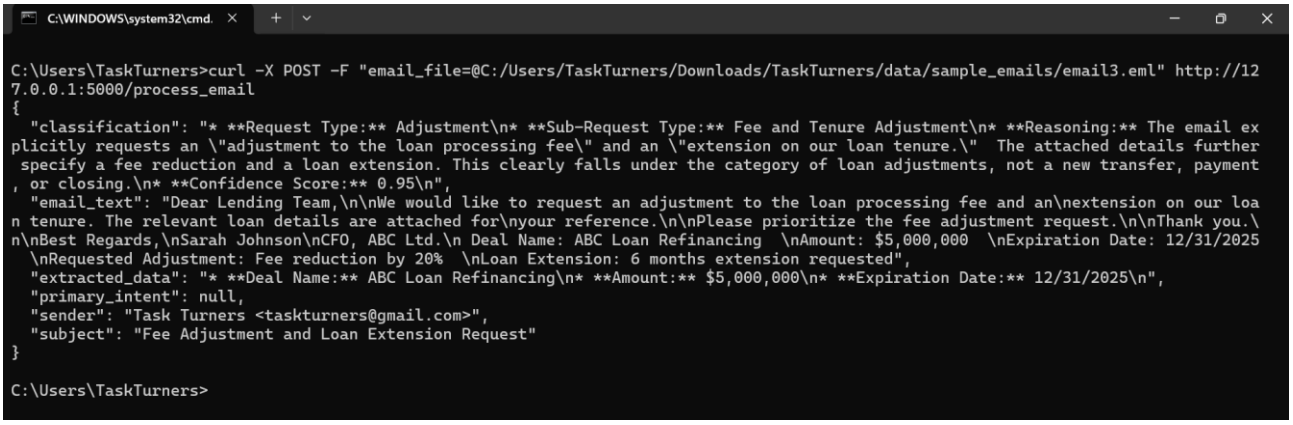
1. email1.eml (Email with no attachment)



2. email2.eml (Email with attachment)



3. email3.eml (Email with attachment)



4. email4.eml (Email with Docx attachment)