DeepSeek Email Classification & OCR - Documentation

1. Project Overview

This project processes emails (.eml/.msg) to classify their request types and extract key details (amount, date, deal name) using a fine-tuned DeepSeek AI model. It also applies OCR for attachment processing.

Objective: Automate the classification and processing of financial service emails to improve operational efficiency and reduce manual effort.

Actors:

- **End Users**: Operations teams handling financial transactions.
- **System**: The AI-powered email classification tool.

Preconditions:

- User provides .eml or .msg files as input.
- Trained model is available for classification.

Workflow:

1. **User uploads an email file** via API or dashboard.
2. **Email text extraction** is performed.
3. **Classification model predicts request type & sub-type**.
4. **Key details (amount, date, deal name) are extracted**.
5. **Results are stored and displayed in the dashboard**.
6. **User exports results if needed**.

Postconditions:

- Email is classified accurately.
- Extracted details are saved in the database.
- User gets structured insights for further processing.

Architecture diagram

Components:

1. **User Interface (API & Dashboard)**
    - A FastAPI-based API that accepts .eml and .msg email files.
    - A Blazor-based dashboard for viewing classification results.
2. **Preprocessing Layer**
    - Extracts email body content from .msg and .eml files.
    - Uses OCR (Tesseract) to extract text from attachments (PDF, images).

    - Converts extracted text into a structured format.

3. **Classification Layer**
    - Uses a fine-tuned DeepSeek LLM to classify emails into predefined request types and sub-types.
    - Extracts key details such as **amount, date, and deal name**.
    - Computes a **confidence score** for classification.
4. **Storage & Data Handling**
    - Saves classified emails and extracted details into a PostgreSQL database.
    - Detects duplicate emails to prevent redundant processing.
5. **Model Training & Fine-tuning**
    - Uses a dataset (email_training_data.json) for supervised fine-tuning.
    - Fine-tunes the DeepSeek model for improved classification accuracy.
    - Saves the trained model (fine_tuned_model/) for inference.

---

2. File Structure

```
GenAIEmailClassification/
|-- data/                # Folder containing sample email data (.eml, .msg) and attachments
|   ├── sample1.eml
|   ├── sample2.msg
|   ├── attachments/
|       ├── invoice1.pdf
|       ├── receipt2.png
|
|-- scripts/             # Folder for core scripts
|   ├── model.py              # Model definition & loading
|   ├── finetune.py           # Script for fine-tuning the model
|    ├── api.py                    # FastAPI implementation for classification & OCR
|    ├── utils.py                  # Helper functions for email processing
|    ├── deepseek_email_classification.py  # Classification logic (renamed)
|    ├── extract_key_details.py        # OCR & data extraction logic
|
|-- trained_model/                # Folder containing the trained model
|    ├── config.json
|    ├── pytorch_model.bin
|    ├── tokenizer.json
|
|-- test/                         # Folder for testing scripts & results
|    ├── test_samples/              # Sample emails for testing
|    ├── test_results.csv            # Output file with classification results
|
|-- requirements.txt              # Dependencies for installation
|-- README.md                     # Documentation
|-- Deepseek Test Steps.docx         # Test steps document
```

---

3. File Descriptions

- **data/**: Contains sample .eml and .msg emails with attachments for testing.
- **scripts/**: Houses all core scripts.
    - **model.py**: Loads the trained model for email classification.
    - **finetune.py**: Fine-tunes the model with labeled training data.
    - **api.py**: Implements a FastAPI web service to classify emails and extract text.
    - **utils.py**: Helper functions for processing emails.

    - **deepseek_email_classification.py**: The core classification logic.
    - **extract_key_details.py**: Extracts important details like amount, date, and deal name using OCR.
- **trained_model/**: Stores the trained model files (weights, tokenizer, and configuration).
- **test/**: Contains test samples and the results of classification.
- **requirements.txt**: Lists dependencies required for running the project.
- **README.md**: Main documentation file with installation and usage instructions.
- **Deepseek Test Steps.docx**: A step-by-step guide for testing the solution.

---

4. Request Types, Definitions & Subtypes

The model is trained to classify emails into the following request types along with their subtypes:

1. **Adjustment** - Emails related to adjustments in financial transactions.
    - Account Reconciliation
    - Transaction Correction
    - Fee Adjustments
2. **AU Transfer** - Requests for transferring assets under management.
    - Internal Transfer
    - External Transfer
    - Asset Consolidation
3. **Closing Notice** - Notifications regarding closing of a deal or account.
    - Account Closure
    - Final Settlement
    - Loan Closure Notice
4. **Commitment Change** - Requests to modify financial commitments.
    - Credit Line Adjustment
    - Loan Modification

    - Agreement Renewal
5. **Fee Payment** - Emails related to processing fee payments.
    - Invoice Payment
    - Penalty Fees
    - Service Charges
6. **Money Movement Inbound** - Requests concerning inbound fund transfers.
    - Customer Deposits
    - Wire Transfers Received
    - Refund Processing
7. **Money Movement Outbound** - Requests concerning outbound fund transfers.
    - Vendor Payments
    - Loan Disbursements
    - Customer Withdrawals

---

5. Test Steps

1. Environment Setup

- Ensure the Python environment is set up with required dependencies.
- Activate the virtual environment (if applicable):
- source venv/bin/activate (Linux/Mac)
- venv\Scripts\activate (Windows)
- Install dependencies if not already installed:
- pip install -r requirements.txt

2. Running the API

- Navigate to the API script location:
- cd scripts
- Start the FastAPI server using Uvicorn:

- uvicorn api:app --reload
- Verify that the server is running at http://127.0.0.1:8000/docs.

3. Preparing Test Data

- Collect sample .eml and .msg files representing different request types.
- Ensure that some test files contain attachments (PDFs, images) for OCR testing.

4. Uploading Emails for Classification via API

- Use Postman or CURL to send a request to the API endpoint:
- curl -X 'POST' \
-   'http://127.0.0.1:8000/classify-email' \
-   -H 'accept: application/json' \
-   -H 'Content-Type: multipart/form-data' \
-   -F 'file=@sample_email.eml'
- Verify that the response includes a classified request type and extracted details.

5. Testing OCR Extraction via API

- Submit emails with PDF or image attachments.
- Confirm extracted text from images/PDFs is included in the response.

6. Validating API Responses

- Check classification accuracy against expected request types.
- Ensure extracted details (amount, date, deal name) are correctly identified.
- Log results in a CSV file for analysis.

7. Testing api.py End-to-End

- Run the API and upload test emails.
- Check logs and ensure proper processing of .eml and .msg files.
- Verify OCR extraction and classification outputs.
- Test error handling for unsupported file types and incorrect formats.

8. Performance & Error Handling Tests

- Test handling of unsupported file formats.
- Assess response time for various email sizes.
- Verify API stability with multiple concurrent requests.

9. Logging & Exporting Results

- Collect API responses and store them in test_results.csv.
- Review and analyze the accuracy of classification and extraction.

10. Model Fine-Tuning Validation

- Run the fine-tune script using:
- python finetune.py
- Re-test classification accuracy after fine-tuning.
- Ensure the newly trained model is used in model.py.

11. Final Review & Documentation

- Verify all functionalities work as expected.
- Update documentation with any additional findings or improvements needed.