Project: Document Classifier System

Task: improve the existing classifier by adding features and optimisations to handle (1) poorly named files, (2) scaling to new industries, and (3) processing large volumes of documents.

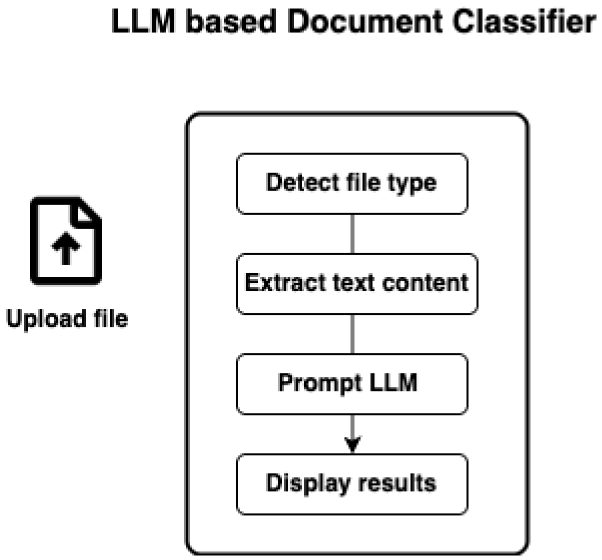
The **Document Classifier System** is a Flask-based web API designed to classify a wide range of document types (e.g., PDFs, images, Word documents, Excel spreadsheets, etc.) using the OpenAI API. It includes support for secure API token-based authentication, file validation, and integration with Tesseract OCR for text extraction from images.

Key Features:

- **Multi-format Support**: Accepts `.pdf`, `.docx`, `.txt`, `.xlsx`, `.csv`, `.png`, `.jpg`, and other common document types.
- **Text Extraction**:
  - Uses Python libraries for parsing structured files.
  - Uses **Tesseract OCR** for extracting text from image-based documents.
- **OpenAI Integration**: Uses `gpt-3.5-turbo` via the OpenAI API to classify documents based on their textual content.
- **Token Truncation**: Automatically trims input text to fit the model's context window to prevent overlength errors.
- **Authentication**: Implements **Bearer token authentication** to secure access to the `/classify_file` endpoint.
- **Swagger UI**: Offers an interactive Swagger-based API documentation interface with built-in token auth for testing.
- **Performance Tracking**: Measures and reports processing time and cost in microdollars per request.
- **Dockerized**: Can be containerized and deployed using Gunicorn for production use.

Data Flow

1. User uploads a document via the `/classify_file` endpoint.
2. The system extracts readable text from the file.
3. It prepares a prompt and sends it to OpenAI's chat model.
4. The model responds with the predicted document type.
5. The API returns the classification, cost, and time taken in JSON format.



Each document costs less than one-tenth of a cent to process. To enable precise cost tracking over time, costs are recorded in micro dollars. By capping the maximum number of input tokens, we can effectively control API usage costs.

- Maximum API cost per document: $0.000152

- Processing time: Varies based on file size

The application supports the following file types:

File Type	Library Selected for Text Extraction
Pdf	Pdfplumber
Text	-
Docx	Py-docx
Image: Png, jpeg, jpg	Tesseract, Pytesseract, PIL

Spreadsheets: xlsx, xls, csv	openpyxl , xlrd, csv

Additional Notes

1. The LLM prompt is designed to be industry-agnostic and adaptable to various document types. Since the model processes the actual text content, it can accurately identify the document type even if the filename is incorrect.
2. For production deployment, the application uses **Gunicorn** instead of Flask's built-in server, offering a robust, multi-threaded, and production-ready setup.
3. The application is **Dockerized** to support seamless storage in a remote container registry and streamlined deployment.
4. **Token-based authentication** is implemented: API access is permitted only when the token provided in the request header matches the one defined in the environment file.
5. Comprehensive **tests** are included to validate the service's functionality.

Future Improvements:

1. Add a React/VueJS based UI which allows multiple users to login and upload their files for classification.
2. The API can be extended to accept batches of documents.
3. Another API can be created which reads a file location and automatically classifies all the files in that location.
4. Database connectivity can be added to track the file location and document type in a table.
5. The prompt can be altered to choose from a target list of document types.
6. The user can be allowed to select which OpenAI model they want to use for the task.

I also surveyed the following models for this task:

Model	**Cost**	**Inference Time**	**Observations**
Kl3m https://kl3m.ai/	Free	10 seconds	Completed the document text instead of classifying the document

Mistral https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2	Free	>2 minutes	Could not observe, took too long to generate answer
GPT-3.5-Turbo (4k context length)	**Input tokens**: $0.0015 per 1,000 tokens **Output tokens**: $0.0020 per 1,000 tokens	~ 1 second	OpenAI’s cheapest model, generalizes to industries very well

gpt-3.5-turbo was selected since it was the clear winner.

Rough Notes/Brainstorming

Part 1: Enhancing the Classifier

Q. What are the limitations in the current classifier that's stopping it from scaling?

1. *it only handles certain filenames*
2. *it only handles certain file types*
3. *it doesn't check for the content of the files*

Q. How might you extend the classifier with additional technologies, capabilities, or features?

Check the text content of the files in addition to just the name, because the name may be incomplete or inaccurate

Part 2: Productionising the Classifier

Q. How can you ensure the classifier is robust and reliable in a production environment?

- make sure files aren't stored in the container

Q. How can you deploy the classifier to make it accessible to other services and users?

- Dockerize the Flask application

- expose the classifier endpoint via an API page

Marking Criteria

1. Functionality: Does the classifier work as expected?
   - an LLM can classify based on text content
   - for images, extract the text using tesseract
   - for pdfs, extract the text using pymupdf
   - for txts, just use the text

2. Scalability: Can the classifier scale to new industries and higher volumes?
   - since it is not limited to a set number of document types, it can handle new industries as well
3. Maintainability: Is the codebase well-structured and easy to maintain?
4. Creativity: Are there any innovative or creative solutions to the problem?
   - use a cheap LLM so that the cost is low
   - prompt engineer the LLM to handle different types of documents
5. Testing: Are there tests to validate the service's functionality?
   - write tests for document set
6. Deployment: Is the classifier ready for deployment in a production environment?
   - make a docker container that is ready to deploy on AWS

Possible Ideas / Suggestions

1. Train a classifier to categorize files based on the text content of a file
   - are there any existing classifiers?
2. Generate synthetic data to train the classifier on documents from different industries
   - hmmmm
3. Detect file type and handle other file formats (e.g., Word, Excel)
   - can do for sure, detect the file format and handle separately
4. Set up a CI/CD pipeline for automatic testing and deployment
   - make a docker container and then look up how to make a CI/CD pipeline
5. Refactor the codebase to make it more maintainable and scalable