Project Report: Automated Document Processing Module

Developed by Ahmed Ayman

November 2025

1. Introduction and Project Objective

The objective of this module was to design and implement a scalable, intelligent data pipeline capable of automatically processing and classifying critical business documents, specifically **Invoices**, **Purchase Orders (POs)**, and **Approvals**. The solution integrates state-of-the-art Large Language Models (LLMs) for classification and data extraction, robust Optical Character Recognition (OCR) for handling diverse document formats (scanned images and PDFs), and a persistent NoSQL database (MongoDB) for structured storage and data integrity.

2. System Architecture and Technologies

The solution comprises a backend API service (Flask), a dedicated user interface (Streamlit), and a containerized deployment setup (Docker), utilizing a modern Python ecosystem.

2.1 Core Components

- **Backend API (Flask):** Manages file uploads, data processing, and database interactions. Configured with file size limits (16MB) and secure handling of uploads.
- **Frontend Application (Streamlit):** Provides an intuitive web interface for document submission and visualization of extracted data.
- **Deployment (Docker):** Ensures environment consistency across development and production, bundling the Python runtime, dependencies, and necessary system tools like `tesseract-ocr`.
- **Database (MongoDB):** Used for permanent storage, categorized into three distinct collections (`invoices`, `purchase_orders`, `approvals`) for efficient querying.

2.2 Technology Stack

- **LLM Integration:** `ChatGoogleGenerativeAI` (Gemini-2.5-Flash) via LangChain is utilized for complex tasks: document classification and structured information extraction (JSON output).
- **Text Extraction/OCR:** `PyTesseract` handles text extraction from image formats (PNG, JPG), while `pdfplumber` is used for reading text directly from PDF files, ensuring wide document compatibility.

- **Data Embedding:** The `SentenceTransformer` (`paraphrase-multilingual-MiniLM-L12-v2`) generates vector embeddings for the extracted text, enabling future semantic search capabilities and enhancing duplicate detection.

3. Automated Processing Pipeline

The document processing follows a five-step pipeline designed for robustness and high throughput.

1. **File Validation and Text Extraction:** The API receives a file (PDF or image). It checks the file type and uses the appropriate tool (`pdfplumber` or `PyTesseract`) to convert the document content into raw, searchable text.
2. **Document Classification:** The raw text is passed to the Gemini LLM with a specific system prompt to classify the document into one of the three target categories: `invoice`, `purchase_order`, or `approval`.
3. **Structured Data Extraction:** A second LLM call is made, leveraging the model's ability to structure information. It extracts key-value pairs relevant to the document type (e.g., invoice number, vendor, total amount) and formats them as JSON.
4. **Embedding Generation:** A high-dimensional vector representation (embedding) of the extracted text is created using the Sentence Transformer model.
5. **Storage and Integrity Check:** Before saving, the system checks for duplicates based on file name or extracted text. New documents are stored in their corresponding MongoDB collection, complete with the raw text, structured JSON data, and the embedding vector.

4. Addressing Module Requirements and Edge Cases

The implementation directly addresses the key requirements outlined in the module, particularly concerning data integrity and pipeline stability.

- ⋆ **Handling Invoices, POs, and Approvals:** Handled via dedicated classification logic and separate collections in MongoDB.
- ⋆ **Multi-modal LLMs / Cloud OCR (Conceptual):** The use of Gemini/LangChain and dedicated OCR tools (`pytesseract`, `pdfplumber`) mimics the functionality of multi-modal models and cloud OCR services by performing image-to-text conversion and complex reasoning/structuring.
- ⋆ **Build Data Pipeline:** A complete pipeline is built, including text extraction, LLM-based transformation, validation (duplicate check), and storage.
- ⋆ **Edge Cases:**
  - **Low-Quality Scans/Handwriting:** Addressed by using image pre-processing (conversion to grayscale) before applying `PyTesseract` OCR, enhancing text extraction fidelity.
  - **Missing Fields/Complex Layouts:** Addressed by relying on the LLM's superior contextual understanding to parse and extract data from complex, unstructured text, providing resilience against variable document formats.

5. Deployment and Access

The application components have been successfully deployed to the Hugging Face Spaces platform, providing public access to both the frontend interface and the backend API service.

- **Streamlit Frontend Application:** Accessible via the following link, providing the user interface for document upload and data visualization: Invoice Reader Streamlit App
- **Flask Backend API:** The underlying processing service is deployed separately for scalability and decoupled access: Document Processor API Endpoint