## Document Log: Advanced AI OCR System Development

**Objective**

The goal is to build an advanced AI-based OCR system for local processing that can accurately extract heterogeneous data (text, tables, images, charts, etc.) from PDFs.

**Approach 1: OCR-Based Extraction (Initial Attempt)**

**Tools Used:**

* Tesseract OCR
* OpenCV for preprocessing
* PDFplumber for text extraction
* PyMuPDF for PDF processing
* Detectron2 for object detection (tables, images)
* LayoutParser for improved table extraction

**Steps:**

1. **Preprocessing:** Used OpenCV to enhance image quality by applying thresholding, denoising, and contrast adjustments.
2. **Text Extraction:** Used Tesseract OCR with different configurations (--oem 3 --psm 6).
3. **Table Extraction:**
   * Initially used PDFplumber and PyMuPDF but faced inconsistencies.
   * Integrated Detectron2 for object detection, improving table detection accuracy.
   * Used LayoutParser to enhance structured table extraction.
   * Experimented with Camelot and PDFTables but found limitations with multi-line headers.
4. **Image & Diagram Extraction:** Used PyMuPDF to extract images and store them separately.

**Challenges & Failures:**

* Poor text extraction accuracy on complex PDFs.
* Initial table extraction using PDFplumber was inconsistent, requiring Detectron2 and LayoutParser integration.
* **Charts with text were not extracted**

**Approach 2: LLaMA 3.2 Vision API for Document Chunking & Extraction**

**Tools Used:**

* LLaMA 3.2 Vision API
* LangChain for document processing
* Semantic chunking for better extraction

**Steps:**

1. **Chunking Approach:** Instead of extracting page-by-page, applied semantic chunking.
2. **Vision API Processing:** Sent chunks to the LLaMA 3.2 Vision API for structured extraction.

**Challenges & Failures:**

* **Ended in rate limit error**

**Approach 3: Exploring Landing AI's Agentic Document Extraction**

**Tools Used:**

* Landing AI’s Agentic Document Extraction
* NLP-based structuring techniques

**Steps:**

1. **Tried automated extraction using Landing AI’s approach.**
2. **Compared with OCR-based and Vision API-based methods.**

**Challenges & Failures:**

* **Limited documentation and customization:** Difficult to fine-tune for specific document structures.
* **Integration issues with local processing:** Required more cloud dependency than desired and ended in **RATE LIMIT ERROR**

**Approach 4: Fully Local AI OCR with Ollama & Table Transformer**

**Tools Used:**

* Ollama for local LLM inference
* Table Transformer for table extraction
* Image segmentation for structured data extraction
* PaddleOCR for alternative OCR processing

**Steps:**

1. **Set up Ollama for local LLM inference.**
2. **Used Table Transformer for improved table recognition.**
3. **Tested PaddleOCR as an alternative to Tesseract for text extraction.**
4. **Integrated text, table, and image extraction into a structured format.**

**Challenges & Failures:**

* **Table extraction inconsistencies:** Some tables were detected incorrectly.
* **JSON serialization errors:** Encountered serialization issues while structuring the extracted data.
* **Table Transformer did not show improvements compared to primary approach—Need to improvise**

**Approach 5: Mistral OCR API for Heterogeneous Document Processing**

**Tools Used:**

* Mistral OCR API (<https://docs.mistral.ai/capabilities/document/>)
* Python for API requests
* JSON & Pandas for structuring extracted data

**Steps:**

1. Set up API access for Mistral OCR.
2. Uploaded PDFs locally by specifying their path instead of using a URL.
3. Extracted text, images, charts, and tables using the API.
4. Saved extracted tables in a single Excel file with separate sheets per table.
5. Saved all extracted text in a single .txt file.
6. Stored images and diagrams in a separate directory.

**Challenges & Failures:**

* Need to resolve 401 client error and 502 server error caused due to formatting and server issues which will be handled