**Milestone - III**

**Documentation for Llama-OCR and Data Extraction with Gradio UI**

This document provides detailed documentation for a Python-based OCR (Optical Character Recognition) and data extraction system implemented with the help of llama-ocr, Gradio UI, and various Python libraries. The system processes multiple document types, extracts relevant data fields, and visualizes the extracted information.

**1. Introduction**

The OCR and Data Extraction system allows users to:

* Upload multiple document images of specific types (e.g., Salary Slips, Bank Statements, Invoices).
* Extract meaningful data fields from these documents.
* Visualize the extracted information in graphical formats such as pie and bar charts.

The project uses the llama-ocr package for OCR processing, Gradio for building a user-friendly interface, pandas and matplotlib for data handling and visualization.

**2. Dependencies**

Ensure the following libraries and tools are installed:

* **Python Libraries:**
  + subprocess
  + os
  + pandas
  + gradio
  + matplotlib
  + re
  + dotenv
* **Node.js:** Required for running the llama-ocr script.
* **llama-ocr:** JavaScript OCR package.

**3. Environment Setup**

1. Install Python dependencies using pip install:

pip install pandas gradio matplotlib python-dotenv

1. Install Node.js and the llama-ocr package.

npm install llama-ocr

1. Create a .env file in the project directory and add your apiKey:

apiKey=YOUR\_API\_KEY\_HERE

**4. Different Techniques for OCR**

During the development of this project, various OCR techniques and tools were explored:

1. **EasyOCR:** Initially, EasyOCR was tested for data extraction. While it successfully recognized text from images, it could not reliably extract field-specific data.
2. **Cohere-Llama:** Cohere-Llama was then explored as an alternative. However, it also fell short in accurately extracting specific fields.
3. **Llama-OCR (Together.ai):** Finally, Llama-OCR was chosen for its efficient text recognition capabilities. It proved to be the most reliable tool for the project, successfully extracting field-specific data when combined with prompting techniques.

**5. Challenges Faced**

Several challenges were encountered during the implementation of the OCR system:

1. **Field-Specific Data Extraction:** Initially, it was difficult to extract specific fields such as "Basic Salary" or "Total Allowances" from the OCR output. The tools often returned generic text data without structure.

**Solution:** Prompting was introduced to instruct Llama-OCR to extract only the desired fields. By crafting detailed prompts, the system could focus on relevant data.

1. **Visualization of Extracted Data:** Another challenge was the lack of tools for directly visualizing the extracted data. Raw data needed to be converted into numeric values and structured appropriately for visualization.

**Solution:**

* + Extracted numeric values were parsed using regular expressions and converted into a structured format (DataFrame).
  + matplotlib was used to generate pie and bar charts for a clear and concise representation of the data.

1. **Error Handling:** Timeout errors and missing field values required robust error-handling mechanisms to ensure a smooth user experience.

**Solution:** Error-handling blocks were added to manage timeouts and unexpected inputs gracefully.

**6. Gradio Interface**

The Gradio interface provides a user-friendly way to interact with the system. It includes:

* **Inputs:**
  + Dropdown to select the document type.
  + File uploader for multiple documents.
* **Outputs:**
  + DataFrame displays the full extracted text.
  + DataFrame displays specific extracted fields.
  + Image of generated visualizations.

**Code:**

interface = gr.Interface(

fn=interface,

inputs=[gr.Dropdown(choices=document\_types, label="Document Type"),

gr.File(label="Upload Documents", file\_types=["image"], file\_count="multiple")],

outputs=[gr.Dataframe(label="Full Extracted Text (Raw Data)"),

gr.Dataframe(label="Specific Extracted Data"),

gr.Image(label="Charts (Pie & Bar)"),],title="Llama OCR with Data Visualization")

**7. How to Run the Application**

1. Save the code into a file, e.g., app.py.
2. Run the script:

python app.py

1. Open the URL displayed in the terminal to access the Gradio interface.

**8. Error Handling**

* **Timeouts:** The OCR subprocess is capped at 60 seconds to prevent hanging.
* **Invalid Inputs:** Functions include checks and sanitization to handle unexpected inputs gracefully.
* **Logs:** Errors and debug information are logged to the console for troubleshooting.

**Conclusion**

This OCR and Data Extraction system leverages powerful tools to provide a seamless experience for extracting and analyzing document data. By combining the capabilities of llama-ocr, Gradio, and Python, the system simplifies the processing and visualization of textual data. The challenges faced during the development process were addressed with effective solutions, making the system robust and user-friendly.