Documentation

1. Understanding Data and PS

1.1. P.S: Invoices — they're an unavoidable part of business, from local shops to global companies. But manual invoice processing? It's inefficient, prone to errors, and just not scalable. Enter automation. The aim here is to build a system that extracts critical information from invoices, whether they're pristine PDFs, scanned copies, or a wild combo of both. In essence, we're creating a solution that deciphers and extracts meaningful data from any type of invoice document.

1.2. Data: The variety of invoices we're dealing with is far from straightforward:

- **Standard PDFs**: Easy text extraction but still need precision.
- **Scanned PDFs**: These are essentially images inside PDFs, trickier to deal with.
- **Mixed PDFs**: A blend of text and images that can easily trip up basic extraction methods.

From these, we aim to pull the essentials:

- **Invoice Number**
- **Invoice Date**
- **Customer Details**
- **Line item details**
- **Place of supply**
- **Due date**
- **Total(including the taxes) & total discount**

2. Approach Explanation

2.1. Approach:

So, how does this work? Let me walk you through

2.1.1. **User Input**: It all starts with users uploading PDFs or images of invoices.

2.1.2. **PDF to Image Conversion**: If it's a PDF, I convert each page to an image because working with images is more reliable for handling both clean and scanned documents.

2.1.3. **Google AI Integration:** Once we've got the images, I leverage Google's **Gemini-1.5-pro-002** model. It's like giving the system a pair of super-smart glasses to read and extract invoice data accurately, no matter how the content is formatted.

2.1.4. **Gemini-1.5-pro-002:** It's a mid-size multimodal model, optimised for scaling across a wide-range of tasks.Gemini 1.5 Pro comes with a standard 128,000 token context window.,A 128,000 token context window can typically handle about **80,000 to 96,000** words. Given that a standard page of text contains approximately **250 to 300** words, this context window can accommodate roughly **320-384** pages of a document, depending on formatting and content density.

2.2. Comparison with Other Approaches:

2.2.1. How does this stack up against basic extraction techniques?

2.2.1.1. **Basic PDF Text Extraction:**Utilises libraries like PyPDF2 or PDFMiner to extract selectable text from PDFs

- Works well when the text is nice and clean
- Downside: Fails miserably with scanned invoices or mixed content. Also can only produce text in paragraphs or lines. But inefficient for tabular data extraction.

2.2.1.2. **Optical Character Recognition (OCR):**Tools like Tesseract or Google Cloud Vision convert images of text into machine-readable text.

Pros:

- Effective for scanned documents and images, making it versatile for various formats.
- Can handle non-standard fonts and handwriting to some extent.

Cons:

- Accuracy can vary significantly based on image quality and layout complexity.
- Often requires preprocessing (e.g., image cleaning) to enhance performance.

2.2.1.3. **Deep Learning Models**: Models like CRNN (Convolutional Recurrent Neural Network) designed specifically for sequence-to-sequence tasks, processing images directly to produce structured data outputs

Pros:

- High accuracy for complex layouts and diverse formats
- Ability to learn directly from raw data without extensive preprocessing.

Cons:

- Requires substantial computational resources for training and inference
- Potentially overkill for simpler invoices or low-volume task.

2.3. Code Explanation:

2.3.1. Function get_gemini_response(input, images, prompt):

- Functionality: This function initialises the Gemini model and generates a response based on the input prompt and images provided. It returns the text response from the model.

2.3.2. Function input_image_setup(images):

- Functionality: Prepares the images for sending to the Gemini API by converting them into a byte format and appending them to a list with their MIME type.

2.3.3. Function validate_invoice_data(response):

- Functionality: Analyses the model's response to identify potential issues, such as missing total amounts or mismatches between the total and line items.

2.3.4. Streamlit App Configuration:

- **Functionality**: Configures the Streamlit application, sets the page title, and creates a text input field for user prompts. It also allows users to upload multiple files in PDF or image formats.

2.3.5. Image Processing and Display:

- **Functionality:**
  - The application processes uploaded files, converting PDFs to images and appending them to a list.
  - It limits the number of images displayed to **3** by using the MAX_DEFAULT_IMAGES constant.
  - Thumbnails of the images are shown in a grid layout (five columns), allowing users to select images they want to view in detail.

2.3.6. Handling User Submission:

- **Functionality:**
  - Handles the button submission by checking if images have been uploaded.
  - Calls the get_gemini_response function to get a response based on the input prompt and selected images.
  - Validates the response for potential issues and displays the output accordingly.

2.4. Dependencies Installed:

2.4.1. Pillow (PIL): For handling image processing

2.4.2. Streamlit: For building a simple, interactive web app

2.4.3. pdf2image: To break PDFs into manageable image files

2.4.4. **google.generativeai**: The star of the show, helping us extract structured data from unstructured images

3. Accuracy Assessment:

The accuracy assessment was conducted on 41 test PDFs containing invoices with varying structures and details. The goal was to evaluate the performance of the application in extracting essential invoice information. The following fields were successfully and consistently extracted from every PDF:

- **Invoice Number**
- **Invoice Date**
- **Customer Details**
- **Line item details**
- **Place of supply**
- **Due date**
- **Total(including the taxes) & total discount**

3.1. Detailed Breakdown:

3.1.1. Invoice Number: Extracted correctly for all 41 PDFs.

3.1.2. Invoice Date: Accurate identification of the invoice date, regardless of its position or formatting.

3.1.3. Customer Details: The customer name, address, and contact information were extracted without issues across all test cases.

3.1.4. Line Item Details: Line items, including descriptions, quantities, and prices, were extracted accurately, even when line items varied in number.

3.1.5. Place of Supply: Extracted correctly in all test PDFs.

3.1.6. Due Date: Identified and extracted accurately in every instance.

3.2. Error Handling and Reporting:

Although no major errors or inaccuracies were encountered during testing, the system includes a robust error detection mechanism. We have implemented code to validate and flag potential issues, such as:

- Missing or unclear total amounts
- Mismatches between total amounts and line items

These validations ensure that any extraction discrepancies are caught and reported to the user for review. This proactive approach enhances trust in the extracted data and provides clear feedback when manual verification might be required.

3.3. Data reliability:

The system consistently extracted the correct data across all 41 test PDFs, handling varying invoice formats with ease. Even when certain fields (e.g., line items or customer details) were presented in different formats, the model successfully interpreted and extracted the relevant information. This demonstrates the system's adaptability and robustness, making it highly reliable for diverse invoice structures. Consequently, the system can be applied to a broad range of invoice types without requiring manual intervention or format-specific adjustments.

4. Performance and Cost Analysis

4.1. Trust Determination Logic:

We employ a trust scoring mechanism to assess the reliability of the extracted invoice fields. Each field is assigned a score out of 100, based on its importance in invoice accuracy:

- Invoice Number: 9
- Invoice Date: 9
- Customer Details: 9
- Line Item Details: 35
- Place of Supply: 9
- Total (Including Taxes & Discount): 20
- Due Date: 9

Line item details and total amounts carry the highest weight, as these are crucial for financial verification.

4.2. Cost-Effectiveness:

Our solution utilises Google's Gemini 1.5 API free tier, which offers:

- 15 Requests per Minute (RPM)
- 1 million Tokens per Minute (TPM)
- 1,500 Requests per Day (RPD)
- Context caching: Free for up to 1 million tokens per hour
- Tuning service: Free of charge

With **128K** tokens accommodating **96k** words or roughly **320-384** pages, the free plan is highly cost-effective for moderate-volume invoice extraction, making it ideal for small-to-medium businesses. Higher volumes can be managed by moving to the paid tier, where detailed pricing can be found **[here]**.

4.3. Scalability:

The API's free tier supports scalable testing and small business operations. For higher volumes, the system can be upgraded to handle more requests while maintaining optimal speed. The model's ability to process large PDFs efficiently ensures minimal latency, making it suitable for real-time or near-real-time applications.

5. Instructions To Use:

NOTES:

- Please go to the bin of poppler file and add this path to your system using Environment variable
- Please generate a google gemini API from here and use it in .env file.
- Please create a virtual environment and activate it. Then install your dependencies there
- Create virtual env -> python -m venv venv
- Activate the venv -> venv\Scripts\activate
- To run the app , use comman -> streamlit run app.py
-