Zolvit Problem Statement

Code for the above problem statement is available at https://github.com/Shrihari-Nair/zolvit_invoice_data_extraction. Please follow `README.md` file for running the code.

Background

You are tasked with developing a solution to extract data from invoices. The invoices are in PDF format, which may include a mixture of regular PDFs, scanned documents, and PDFs containing both text and images.

Objective

Your goal is to implement the most cost-effective approach while maximizing accuracy in data extraction. Accuracy takes priority, with an expected accuracy rate of >90%.

Code Structure

```
|-- ./
    |-- .git/
    |-- .env
    |-- .gitignore
    |-- main.py
    |-- extract.py
    |-- accuracy_check.py
    |-- gemini.py
    |-- json_structure.py
    |-- README.md
    |-- requirements.txt
    |-- trust_determination.py
    |-- utils.py
    |-- artifacts/
        |-- ground_truth_jsons/
        |-- accuracy_values/
            |-- regular/
            |-- scanned/
        |-- ground_truth_jsons/
        |-- images/
        |-- invoice_snaps/
        |-- json_dumps/
            |-- regular_pdf_jsons/
            |-- scanned_pdf_jsons/
        |-- text_data/
    |-- data/
        |-- Jan to Mar/
    |-- notebooks/
```

- **`main.py`**: Acts as the entry point for the data extraction process, handling both regular and scanned PDFs.
- **`data` folder**: Contains the PDF files that are used for extraction purposes.
- **`extract.py`**: Contains the logic to determine whether a PDF is regular or scanned and applies the appropriate extraction algorithm accordingly.
- **`accuracy_check.py`**: Responsible for computing the accuracy of the extraction after the data is stored in JSON format, which is saved in the `artifacts/json_dumps` folder.
- **`json_structure.py`**: Converts the raw text extracted by `extract.py` into a structured JSON format. It compares this output with the manually created ground truth JSON files located in `artifacts/ground_truth_jsons`.
- **`trust_determination.py`**: Evaluates the quality of the extraction by cross-verifying the extracted data based on invoice logic and calculating various metrics present in the PDF.
- **`gemini.py`**: Demonstrates how large language models (LLMs) can consistently capture and structure data in an alternate manner.
- **`artifacts` folder**: Stores intermediate outputs such as extracted texts, images and `json` files containing accuracy values for each PDF.

Data Extraction

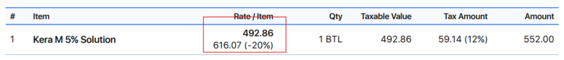
The first step in the data extraction pipeline involves determining whether the input PDF is regular or scanned, which was accomplished using the `pdfplumber` package.

Regular PDF

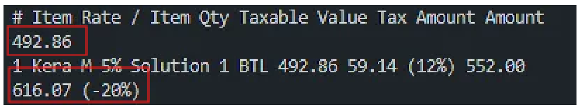
PyPDF2 and pdfplumber

- The `PyPDF2` and `pdfplumber` packages were used for extracting text and images from regular PDFs.
- However, they struggled to accurately capture table data, often resulting in misalignments and occasionally omitting fields.
- While these packages were effective at extracting images from PDFs, they were not suited for handling structured entities like tables.

Example of alignment mismatch.



#	Item	Rate / Item	Qty	Taxable Value	Tax Amount	Amount
1	Kera M 5% Solution	492.86 616.07 (-20%)	1 BTL	492.86	59.14 (12%)	552.00



```
# Item Rate / Item Qty Taxable Value Tax Amount Amount
492.86
1 Kera M 5% Solution 1 BTL 492.86 59.14 (12%) 552.00
616.07 (-20%)
```

PyMuPDF

- **PyMuPDF** proved more effective for capturing text from tables as well.
- Although the extracted text was not always in the correct order, this issue was addressed using appropriate regular expression techniques to reprocess the data.

Google Gemini API

- The **pdf2image** package was used to convert PDF files into images, which were then processed using the **Google Gemini Model** (versions 1.5 Flash and 1.5 Pro).
- With effective system and prompt engineering, the model successfully extracted text from the images and formatted it into JSON directly, eliminating the need for additional text-to-JSON conversion.
- **Gemini 1.5 Pro** demonstrated superior extraction accuracy compared to 1.5 Flash.

- **Advantage**: The model was capable of identifying missing fields, correcting misalignments, and addressing issues that the other packages were unable to handle.
- **Disadvantage**: The Google Gemini API introduces dependency on third-party services, and we'd prefer having model weights locally for better control and cost efficiency, as API usage incurs additional charges.
- Configuration used is shown below.

```
MODEL_CONFIG = {
    "temperature": 0.1,
    "top_p": 1,
    "top_k": 32,
    "max_output_tokens": 4096,
    }
```

The configuration is optimized for precision and reliability, with a low temperature to reduce randomness and a constrained set of tokens (
`top_k`) to ensure that only the most likely tokens are considered. This setup suits tasks where high accuracy is needed, such as extracting structured data from documents like invoices. The higher token limit ensures that the model can handle large outputs, which is essential for complex or lengthy PDFs.

Scanned PDF

The PDFs in the dataset were converted into images and processed through the pipeline designed for handling scanned PDFs. This was achieved using `pdf2image` package.

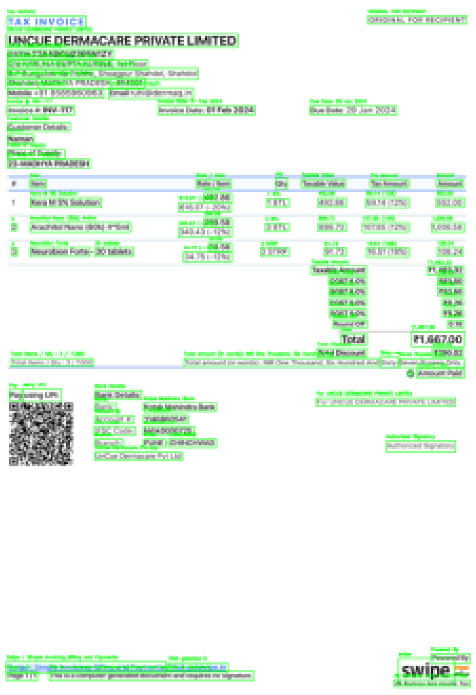
Pytesseract

- The package successfully captures most words, but lacks spatial information, meaning all words are placed on new lines, making it difficult to determine which words were originally grouped together in a single line.
- There is also inconsistency in the structure of the extracted data.

- Too much noise in the extracted data.

EasyOCR

- `easyocr` captures most of the information similar to `pytesseract` , but it also provides spatial information in the form of bounding box coordinates.
- This spatial data is utilized to ensure words are captured on the same line. A small tolerance value is applied to determine if the next word belongs to the same line as the current word by comparing the bottom `y` coordinate of the current word with that of the next word.
- It performs significantly better than `pytesseract` .
- **Disadvantage**: The initial loading time for the model is high during the first use.



TAX INVOICE

ORIGINAL FOR RECIPIENT

UNCUE DERMACARE PRIVATE LIMITED

GSTIN [illegible]

[illegible]

[illegible]

[illegible] MADHYA PRADESH, [illegible]

Mobile +91 [illegible] Email [illegible]

Invoice #: INV-117 Invoice Date: 01 Feb 2024 Due Date: 29 Jan 2024

Customer Details:

Naman

Place of Supply:

23-MADHYA PRADESH

#	Item	Rate / Item	Qty	Taxable Value	Tax Amount	Amount
1	Kera M 5% Solution	492.86 [illegible] (-20%)	1 BTL	492.86	59.14 (12%)	552.00
2	Arachitol Nano (60K) 4*5ml	299.58 340.43 (-12%)	3 BTL	898.73	107.85 (12%)	1,006.58
3	Neurobion Forte - 30 tablets	[illegible] 34.75 (-12%)	3 STRIP	91.73	16.51 (18%)	108.24

Taxable Amount	₹1,483.32
CGST 6.0%	₹83.50
SGST 6.0%	₹83.50
CGST 9.0%	₹8.26
SGST 9.0%	₹8.26
Round Off	0.16
Total	₹1,667.00
Total Discount	₹280.03

Total Items / Qty : 3 / 7.000

Total amount (in words): INR One Thousand, Six Hundred And Sixty-Seven Rupees Only.

Amount Paid

Pay using UPI:

Bank Details:

Bank:	Kotak Mahindra Bank
Account #:	[illegible]
IFSC Code:	[illegible]
Branch:	PUNE - CHINCHWAD
	UnCue Dermacare Pvt Ltd

For UNCUE DERMACARE PRIVATE LIMITED

Authorized Signatory

Swipe | Simple Invoicing, Billing and Payments | Visit getswipe.in

Page 1 / 1 This is a computer generated document and requires no signature.

Powered by swipe

Bounding boxes around each word detected by EasyOCR.

Once the text is extracted, it is passed to `json_structure.py` to convert it into a structured `json` format to use it later for comparing to the ground truth. The logic to extract data is different for both regular and scanned PDF as the text structure after passing through the pipeline is slightly different for both.

Google Gemini API

- Similar to the regular PDFs, Google API was used to capture words and structure it properly to `json` using proper prompts.
- Like above, the processing step of converting text file to `json` is skipped in this case.

Example of the JSON structure is shown below. This ground truth JSON file has a consistent format for outputs from both regular and scanned PDFs.

```
{
    "company_name": "UNCUE DERMACARE PRIVATE LIMITED",
    "gst_number": "23AADCU2395N1ZY",
    "address": "C/o KARUNA GUPTA KURELE, 1st Floor\nS.P Bungal
ow Ke Pichhe, Shoagpur Shahdol, Shahdol\nShahdol, MADHYA PRADE
SH, 484001",
    "mobile_number": "+91 8585960963",
    "email": "ruhi@dermaq.in",
    "invoice_number": "INV-117",
    "invoice_date": "01 Feb 2024",
    "due_date": "29 Jan 2024",
    "customer_details": "Naman",
    "place_of_supply": "23-MADHYA PRADESH",
    "items": [
        {
            "Item Number": "1",
            "Item Name": "Kera M 5% Solution",
            "Rate / Item": {
                "Base Rate": "492.86",
                "Discounted Rate": "616.07",
                "Discount": "-20%"
            },
            "Qty": "1 BTL",
            "Taxable Value": "492.86",
            "Tax Amount": {
                "Amount": "59.14",
                "Percentage": "12%"
            },
            "Amount": "552.00"
        }
    ],
    "taxable_amount": "1,483.32",
    "cgst_6": "83.50",
    "sgst_6": "83.50",
    "cgst_9": "8.26",
    "sgst_9": "8.26",
    "igst_12": 0,
    "igst_18": 0,
    "round_off": "0.18",
    "total": "1,667.00",
    "total_discount": "290.02",
    "total_items_qty": {
        "Total Items": "3",
        "Total Qty": "7.000"
    },
    "total_in_words": "INR One Thousand, Six Hundred And Sixty
-Seven Rupees Only.",
    "bank_details": {
        "Bank Name": "Kotak Mahindra Bank",
        "Account Number": "1146860541",
        "IFSC Code": "kkbk0000725",
        "Branch": "PUNE - CHINCHWAD"
    }
}
```

Accuracy Check and Trust Determination

Accuracy Check

- The ground truth `json` files are present in `artifacts/ground_truth_jsons` , while the json files extracted from invoices are present at `artifacts/json_dumps/scanned_pdf_jsons` and `artifacts/json_dumps/regular_pdf_jsons` .
- `accuracy_check.py` is used to compare ground truths and predictions. Every field of the prediction is compared to the ground truth.
- Fields that are not present in the ground truth is ignored. There are cases where some invoices don't contain certain fields which are present in the other like IGST fields. This conditions are handled in the code.
- Harsh accuracy check: The comparison is field by field. If they are not equal, it is considered as a mismatch. But this can be harsh at times.
  In the code below, the string 'C/o' is not present in the prediction but the

remaining portion is captured correctly. Hence just direct comparison must be avoided and some incentive must be given for the overlap between ground truth and prediction.

```
# GROUND TRUTH
"address": "C/o KARUNA GUPTA KURELE, 1st Floor S.P Bungalow Ke
Pichhe, Shoagpur Shahdol, Shahdol Shahdol, MADHYA PRADESH, 484
001"

# PREDICTION
"address": "KARUNA GUPTA KURELE, 1st Floor S.P Bungalow Ke Pic
hhe, Shoagpur Shahdol, Shahdol Shahdol, MADHYA PRADESH, 48400
1"
```

- Accuracy check using `SequenceMatcher` : This package is used to find the longest contiguous matching subsequence between two sequences and calculate a similarity ratio based on the matches and differences. This makes it useful for tasks like finding differences between strings or determining how similar two strings are. It gives similarity between 0 and 1.
  Alternatively,
  `Levenshtein` distance can also be used.
- Accuracy is assessed for each field and stored in the `artifacts/accuracy_values` directory. The JSON structure in this folder mirrors that of the ground truth, with accuracy values replacing the corresponding field values.
- Overall accuracy of a PDF is computed by averaging out all the field accuracies. This value is visible in the files present in `artifacts/accuracy_values` directory.
- Average Overall accuracy is computed by averaging out all the Overall accuracies across all the PDFs.

Trust Determination

- The data extracted is cross verified using the inherent invoice data calculations. If the math works out in accordance to the data extracted, then it is safe to say that the pipeline can be trusted.
- This functionality is handled by `trust_determination.py` .
- The following computations were checked

```
1. Discounted rate = Base rate - (Discount Percentage * Base r
ate)
2. Taxable Value =  Discounted rate * Quantity
3. Tax Amount = Percentage of the Taxable value mentioned in t
he invoice.
4. Amount = Tax Amount + Taxable Value
5. Taxable Amount = Total of all Amounts across the items purc
hased
6. Total = Taxable Amount + CGST 6.0% + SGST 6.0% + CGST 9.0%
+ SGST 9.0% + IGST 12% + IGST 18% + Round Off
```

- If the above cases pass, we can assure that the critical information is extracted accurately and can be used for further analysis.
- Additionally, it is also checked whether the Item Name (medicine in this case) is present in the predefined set of items collected across all the items across PDFs.

Output of `trust_determination.py` for file where all cases pass.

```
['Item 1: Item Name condition fulfilled.', 'Item 1: Base Rate
condition fulfilled.', 'Item 1: Taxable Value condition fulfil
led.', 'Item 1: Tax Amount condition fulfilled.', 'Item 1: Amo
unt condition fulfilled.', 'Item 2: Item Name condition fulfil
led.', 'Item 2: Base Rate condition fulfilled.', 'Item 2: Taxa
ble Value condition fulfilled.', 'Item 2: Tax Amount condition
fulfilled.', 'Item 2: Amount condition fulfilled.', 'Item 3: I
tem Name condition fulfilled.', 'Item 3: Base Rate condition f
ulfilled.', 'Item 3: Taxable Value condition fulfilled.', 'Ite
m 3: Tax Amount condition fulfilled.', 'Item 3: Amount conditi
on fulfilled.', 'Taxable Amount condition fulfilled.', 'Total
condition fulfilled.']
```

Performance Analysis

Regular PDFs

Approach	Accuracy
Gemini Pro	99.4%
Gemini 1.5 Flash	97.13%
PyMuPdf	91.3%
PyPDF2	80.2%

Approach	Compute Time (seconds/per PDF)
Gemini Pro	6.28
Gemini 1.5 Flash	4.22
PyMuPdf	0.32
PyPDF2	0.87

Scanned PDFs

Approach	Accuracy
Gemini Pro	97.9%
Gemini 1.5 Flash	94.13%
EasyOCR	90.9%
Pytessaract	78.5%

Approach	Compute Time (seconds/per PDF)
Gemini Pro	6.28
Gemini 1.5 Flash	4.22
EasyOCR	7.71
Pytessaract	3.17

Note: The accuracy mentioned here is the average overall accuracy computed across all the PDFs computed using Longest Match Algorithm (`SequenceMatcher`).

Observations:

1. **Gemini APIs** deliver the highest accuracy but have slow response times for both regular and scanned PDFs.
2. **PyMuPDF** offers good accuracy and the fastest compute time for regular PDFs.
3. **EasyOCR** achieves decent accuracy, exceeding the 90% threshold, but has slower compute times compared to the API-based approach due to the initial model loading time.
4. **PyPDF2** and **Pytesseract** fall below the required accuracy threshold, despite having faster compute times than their competitors.
5. Based on these observations, **PyMuPDF** strikes the best balance between cost-effectiveness and accuracy for regular PDFs, as it is more than 10 times faster than the API approach. For scanned PDFs, **EasyOCR** offers a solid balance between cost and accuracy.

6. If API usage is not a concern, **Google Gemini** is the preferred choice for optimal accuracy.
7. API calls are not free and hence require additional charges compared to other approaches. And it is always better to not have dependency on a third-party API. A pre-trained model which is fine tuned on the invoice dataset can serve as an alternative.
8. **API Rate Limits**: Consider the potential bottlenecks from rate limits on API calls, which might introduce latency or additional costs (e.g., needing to upgrade API plans).
9. The pipeline does store the intermediate outputs which can work like a cache, reducing processing frequently used PDFs from scratch.
10. **Building custom models**: Initially, API-based OCR or LLMs may be cost-effective, but as the system scales, training in-house models (e.g., fine-tuned OCR models or LLMs) could save costs over time.

Accuracy across data fields (Averaged across PDFs)

Regular

```
{
    "company_name": 1.0,
    "gst_number": 1.0,
    "address": 0.9649122807017544,
    "mobile_number": 1.0,
    "email": 1.0,
    "invoice_number": 1.0,
    "invoice_date": 1.0,
    "due_date": 1.0,
    "customer_details": 0.96,
    "place_of_supply": 1.0,
    "taxable_amount": 1.0,
    "cgst_6": 1.0,
    "sgst_6": 1.0,
    "cgst_9": 0.9830508474576271,
    "sgst_9": 0.983050847457
```

Scanned

```
{
    "company_name": 1.0,
    "gst_number": 1.0,
    "address": 0.9333333333333333,
    "mobile_number": 1.0,
    "email": 1.0,
    "invoice_number": 1.0,
    "invoice_date": 1.0,
    "due_date": 1.0,
    "customer_details": 0.6923076923076923,
    "place_of_supply": 1.0,
    "taxable_amount": 1.0,
    "cgst_6": 1.0,
    "sgst_6": 1.0,
    "cgst_9": 0.9090909090909091,
    "sgst_9": 0.909090909090
6272,
    "igst_12": 1.0,
    "igst_18": 0.94915254237
28814,
    "round_off": 0.814536667
8224333,
    "total": 1.0,
    "total_discount": 1.0,
    "total_in_words": 1.0,
    "total_items_qty": {
        "Total Items": 1.0,
        "Total Qty": 1.0
    },
    "bank_details": {
        "Bank Name": 1.0,
        "Account Number": 1.
0,
        "IFSC Code": 1.0
    },
    "items": [
        {
            "Item Number":
1.0,
            "Item Name": 0.9
824561403508771,
            "Qty": 1.0,
            "Taxable Value":
1.0,
            "Amount": 1.0,
            "Tax Amount": 1.
0,
            "Rate / Item":
1.0
        }
    ],
    "overall_accuracy": 0.91
32095838890576
}
```

```
9091,
    "igst_12": 1.0,
    "igst_18": 0.28571428571
42857,
    "round_off": 0.333333333
3333333,
    "total": 1.0,
    "total_discount": 1.0,
    "total_in_words": 1.0,
    "total_items_qty": {
        "Total Items": 1.0,
        "Total Qty": 1.0
    },
    "bank_details": {
        "Bank Name": 1.0,
        "Account Number": 1.
0,
        "IFSC Code": 1.0
    },
    "items": [
        {
            "Item Number":
1.0,
            "Item Name": 0.9
491525423728814,
            "Qty": 1.0,
            "Taxable Value":
1.0,
            "Amount": 1.0,
            "Tax Amount": 1.
0,
            "Rate / Item":
1.0
        }
    ],
    "overall_accuracy": 0.90
17221863701112
}
```

1. Most of the fields are captured properly by both the extractors.
2. Scanned PDF struggle to capture the `round off` and `customer_details` information properly.
3. Except for `Item Name` , all other item information is captured well.
4. Both extractors fail to capture completely `cgst_9` , `sgst_9`, `igst_18`, `round_off` with 100% accuracy.

Future Work and Scope for Improvement

- Create a model from scratch on the PDF data which can then identify different structures and patterns in the invoice and create a `json` file in a much more efficient manner.
- Fine tune an existing model if there is enough data and capture bounding boxes which contain certain information and pass the cropped version into EasyOCR or some other text extraction model. This can save time as the search space is confined just to the cropped version of the image.
- The above work has API dependency which can be useful in some cases but in long term need a model built from scratch to extract texts.
- Use of regular expressions can makes the processing confined to this PDF format. Use of LLMs can make the text extraction more robust.
- Can use OCR confidence parameter as a metric for evaluation.