Project Documentation - Intelligent Image Text & Graph Data Extraction System

Project Overview

This project is an AI-powered extraction system that:

- Extracts structured text and tables from document images
- Extracts structured numerical data from graph images
- Uses a fallback mechanism for improved accuracy
- Returns structured, machine-readable output

The system combines:

- **OpenCV** for preprocessing
- **Unstructured Library** for table extraction
- **Mistral OCR** as a fallback OCR engine
- **Google Gemini (Structured Output)** for graph understanding

System Architecture

The project consists of two independent but related modules:

1. **Text & Table Extraction Module**
2. **Graph Data Extraction Module**

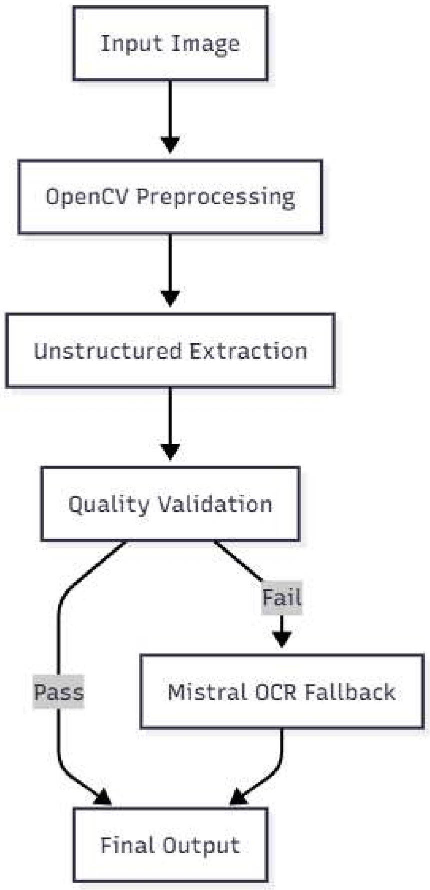
MODULE 1: TEXT & TABLE EXTRACTION

Objective

To extract text and structured table data from document images using:

- Local parsing (Unstructured library)
- AI fallback (Mistral OCR)
- Quality validation mechanism

Workflow Overview



Step 1: Image Preprocessing (OpenCV)

The image is enhanced before OCR to improve accuracy.

Operations Performed:

1. Convert to Grayscale
2. Sharpen Image using Kernel
3. Resize (Minimum width = 2000px)
4. Noise Reduction (Gaussian Blur optional)

Why Preprocessing is Important?

- Improves OCR accuracy
- Enhances faint text
- Removes noise
- Improves number recognition

Step 2: Local Extraction (Unstructured Library)

```
elements = partition_image(filename=image_path, strategy="hi_res")
```

This library:

- Detects document elements
- Classifies content (Table, Title, NarrativeText)
- Extracts table content as HTML

Step 3: Quality Gate System

You implemented a validation system that checks:

✓ At least one table detected
✓ Minimum 2 rows
✓ Less than 50% empty cells

If any condition fails → It triggers fallback.

This prevents:

- Broken tables
- Empty extraction
- Poor structure

Step 4: Mistral OCR Fallback

If local extraction fails:

- Image is converted to base64
- Sent to Mistral OCR API
- Returns Markdown output

This ensures:

- High accuracy
- Better number recognition
- Graph/table detection support

Evaluation of Current Approach

➔ Images Handled Well by Unstructured (Local Extraction)

The current local approach performs well when:

- Tables are clearly structured with visible grid lines
- Text is high contrast and well-aligned
- Image resolution is sufficient
- Tables follow standard row-column format

- No heavy background noise is present

In such cases:

- Extraction is fast
- Structure is preserved
- No external API cost is incurred
- Output remains consistent

➔ Cases Where Mistral OCR Is Required

Fallback to Mistral OCR is necessary when:

- Tables lack visible grid lines
- Image contains shadows or lighting issues
- Text is faint or slightly blurred
- Cells contain dense numerical values
- Local extraction produces incomplete HTML
- More than 50% empty cells are detected
- Rows are misaligned or merged

In these cases, Unstructured may:

- Miss tables completely
- Return fragmented rows
- Produce structurally inconsistent output

Mistral OCR handles these inconsistencies more effectively.

Output

- Extracted text/tables printed in terminal
- Processed image displayed using OpenCV
- Returns structured textual output

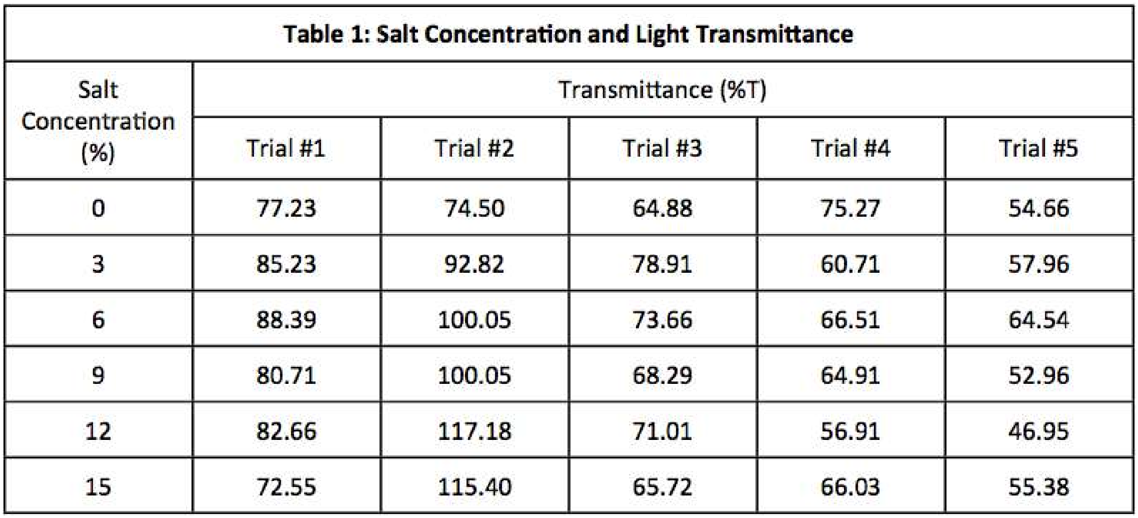
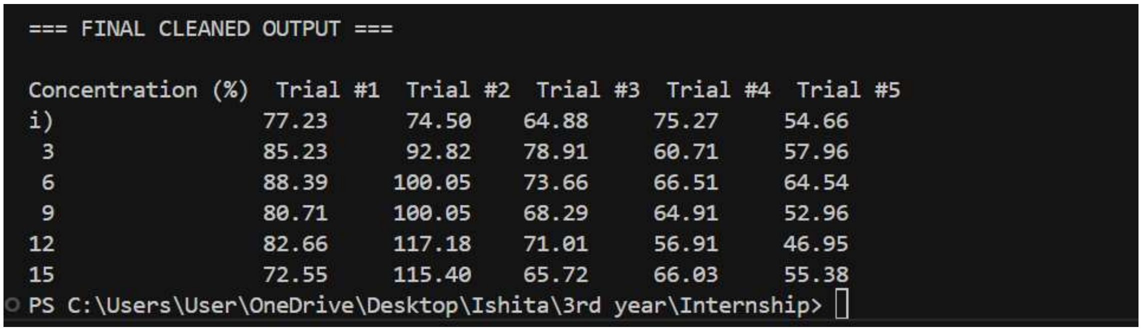


Table 1: Salt Concentration and Light Transmittance

Salt Concentration (%)	Transmittance (%T)				
	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5
0	77.23	74.50	64.88	75.27	54.66
3	85.23	92.82	78.91	60.71	57.96
6	88.39	100.05	73.66	66.51	64.54
9	80.71	100.05	68.29	64.91	52.96
12	82.66	117.18	71.01	56.91	46.95
15	72.55	115.40	65.72	66.03	55.38



```
=== FINAL CLEANED OUTPUT ===

Concentration (%)  Trial #1  Trial #2  Trial #3  Trial #4  Trial #5
i)                77.23      74.50    64.88     75.27     54.66
 3                85.23      92.82    78.91     60.71     57.96
 6                88.39     100.05    73.66     66.51     64.54
 9                80.71     100.05    68.29     64.91     52.96
12                82.66     117.18    71.01     56.91     46.95
15                72.55     115.40    65.72     66.03     55.38
PS C:\Users\User\OneDrive\Desktop\Ishita\3rd year\Internship> 
```

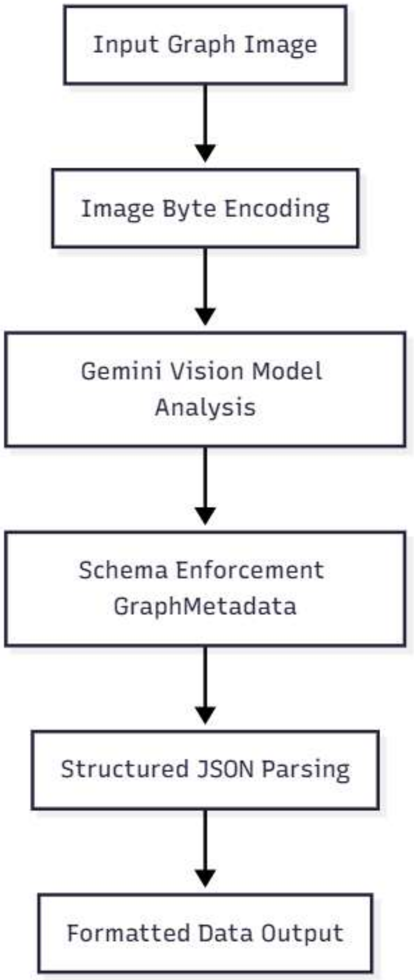
MODULE 2: GRAPH DATA EXTRACTION

Objective

To extract structured technical metadata and numerical data points from graph images using:

- Multimodal AI understanding (Gemini Vision Model)
- Schema-enforced structured output (Pydantic)
- Deterministic configuration for higher numerical accuracy

Workflow Overview



Step 1: Image Input & Encoding

The graph image is read as binary data before being sent to the AI model.

Operations Performed:

- Open image in binary mode
- Convert image into byte stream
- Attach image bytes to Gemini request
- Provide extraction instruction prompt

Why is this Step Important?

- Ensures model receives full visual data
- Preserves graph clarity
- Maintains high-resolution details
- Prepares image for multimodal processing

Step 2: Multimodal Graph Understanding (Gemini Model)

```
client.models.generate_content(...)
```

The Gemini Vision model analyzes:

- Graph structure (Line, Bar, Pie, etc.)
- Axis titles
- Axis scales (Linear, Logarithmic, Dates)
- Legend items
- Visual plot elements
- Data point positions

This is not a traditional OCR.
The model performs **visual reasoning + semantic interpretation**.

Step 3: Schema Enforcement (Structured Output)

The system enforces structured output using:

```
response_schema=GraphMetadata
```

Schema Includes:

- Graph type
- X-axis title
- Y-axis title
- X-axis scale
- Y-axis scale
- Legend items
- Extracted data points

Why is Schema Enforcement Important?

- Prevents free-text responses
- Prevents hallucinated explanations
- Forces correct data types
- Improves reliability and consistency

Step 4: Data Point Structuring

Each extracted data point is mapped into:

```
DataPoint:
- x_value
- y_value
- series_name
```

The model performs:

- Visual coordinate detection
- Mapping pixel positions to axis values
- Associating color/marker with legend series
- Numeric value extraction
- This converts visual graph representation into structured dataset format.

Model Configuration

The system uses:

Model: gemini-3-flash-preview

Configuration:

- Low temperature (0.1) → reduces randomness
- response_mime_type = "application/json"
- response_schema = GraphMetadata

Why Low Temperature?

- Improves numerical precision
- Reduces inconsistent outputs
- Ensures deterministic extraction

Step 5: Parsed Output Handling

```
data = response.parsed
```

The response is automatically parsed into validated Python objects.

Operations Performed:

- Type validation
- Field validation
- JSON-to-object conversion
- Structured table formatting

Output

The system outputs:

- Graph type
- Axis scale information
- Legend items
- Structured data table

The extracted data is:

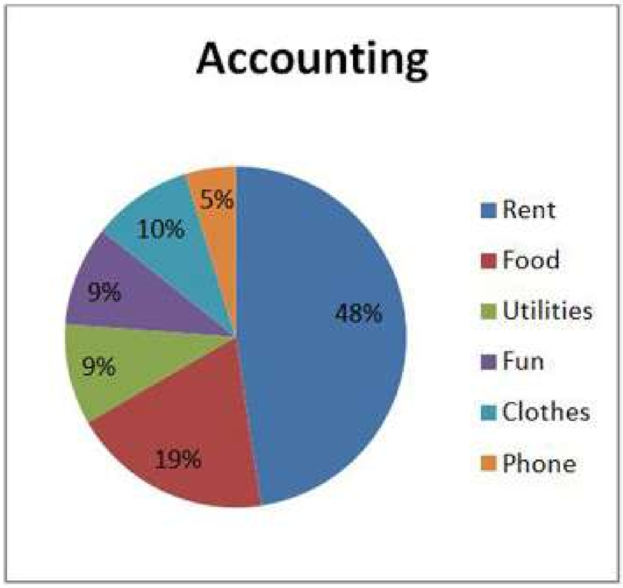
- Machine-readable
- Cleanly formatted
- Ready for CSV/DataFrame conversion
- Suitable for analytics pipelines

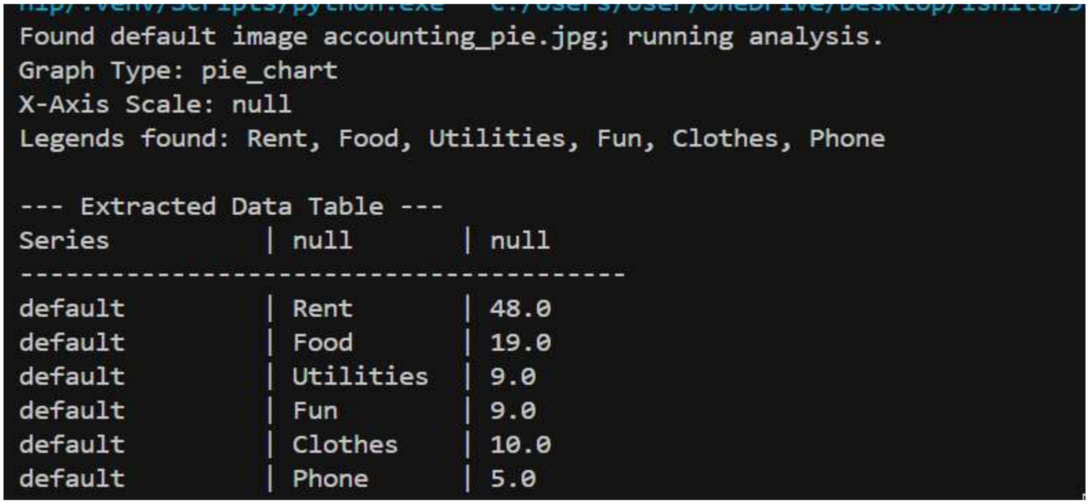
Final Result

The Graph Extraction Module transforms graphical information into structured, analyzable data through:

- Visual understanding
- Semantic interpretation
- Coordinate reasoning
- Strict schema enforcement

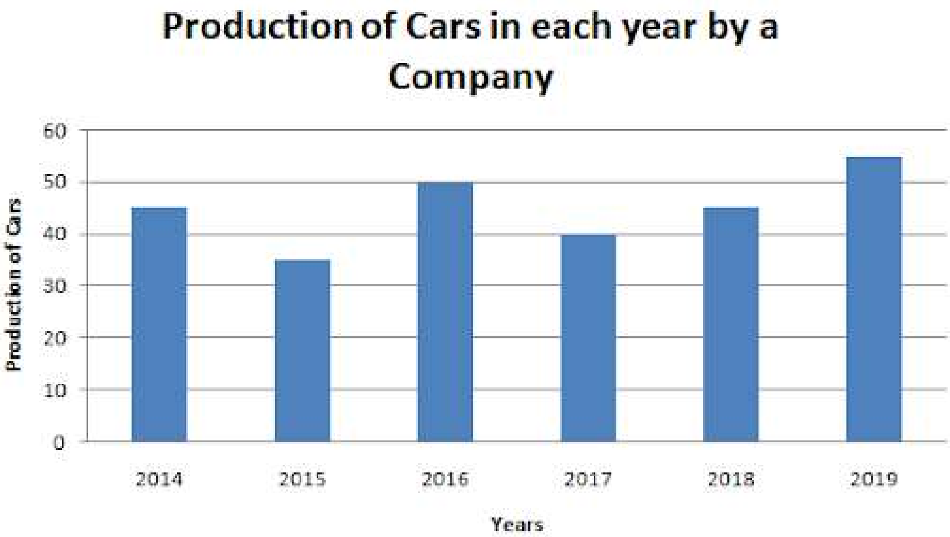
It bridges the gap between visual data representation and machine-readable structured datasets.

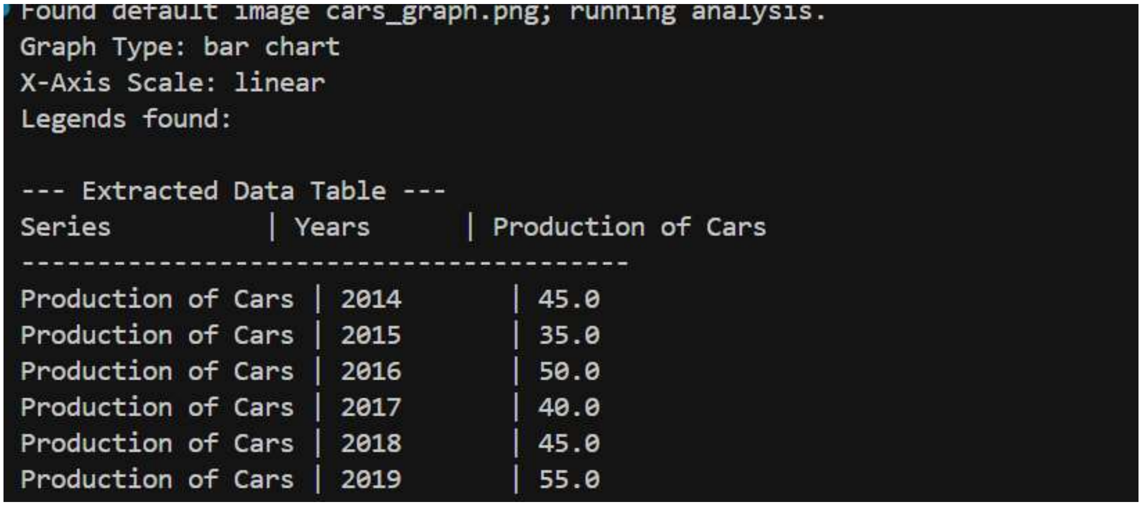




```
hip/.venv/Scripts/python.exe   c:/Users/User/OneDrive/Desktop/Ishita/S
Found default image accounting_pie.jpg; running analysis.
Graph Type: pie_chart
X-Axis Scale: null
Legends found: Rent, Food, Utilities, Fun, Clothes, Phone

--- Extracted Data Table ---
Series          | null        | null
----------------------------------------
default         | Rent        | 48.0
default         | Food        | 19.0
default         | Utilities   | 9.0
default         | Fun         | 9.0
default         | Clothes     | 10.0
default         | Phone       | 5.0
```





```
Found default image cars_graph.png; running analysis.
Graph Type: bar chart
X-Axis Scale: linear
Legends found:

--- Extracted Data Table ---
Series          | Years       | Production of Cars
----------------------------------------
Production of Cars | 2014       | 45.0
Production of Cars | 2015       | 35.0
Production of Cars | 2016       | 50.0
Production of Cars | 2017       | 40.0
Production of Cars | 2018       | 45.0
Production of Cars | 2019       | 55.0
```