This Python script is designed to **process medical prescriptions** (images or PDFs), extract structured information using **OCR and an LLM**, and output **clean, validated JSON** for further use. The goal is to handle both handwritten and printed prescriptions reliably, normalize medical shorthand, and flag ambiguous fields for manual QA.

The script has three main stages:

1. **Image/PDF preprocessing and OCR**
2. **LLM-based parsing (OpenRouter API)**
3. **Output management and fallback handling**

**1. Configuration**

OPENROUTER\_API\_KEY = ""

MODEL\_NAME = "google/gemini-2.5-flash"

INPUT\_FILE = "prescriptions.json"

OUTPUT\_FILE = "extracted\_prescriptions\_clean.json"

IMAGES\_FOLDER = "images"

* **Why**: Sets global variables and file paths.
* **Purpose**: Allows easy modification of model, input, and output paths without changing the core logic.
* **Developer Note**: Make sure the API key is valid and has sufficient quota.

MASTER\_PROMPT = """ ... """

* **Why**: Provides the LLM (OpenRouter) with a **detailed instruction set** on how to parse prescriptions.
* **Purpose**: Ensures:
  + Shorthand expansions (OD, BD, SOS…)
  + Drug normalization and OCR error correction
  + Extraction of patient info, vitals, medicines, diagnostics, doctor info
  + Clean JSON output
  + Manual QA flags for uncertain data
* **Developer Note**: This prompt is central to the LLM’s behavior.

**2. Image Preprocessing**

def preprocess\_image(image):

image = image.convert('L') # grayscale

enhancer = ImageEnhance.Contrast(image)

image = enhancer.enhance(3.0) # increase contrast

enhancer = ImageEnhance.Sharpness(image)

image = enhancer.enhance(2.5) # increase sharpness

return image.point(lambda p: p > 150 and 255) # binarize

* **Why**: Handwritten or scanned prescriptions are often low quality.
* **Purpose**: Improves OCR accuracy by:
  + Converting to grayscale
  + Increasing contrast and sharpness
  + Binarizing (black & white)
* **Developer Note**: Preprocessing parameters (contrast/sharpness thresholds) may need tuning for different scans.

**3. OCR Functions**

**3.1 Images**

def extract\_text\_from\_image\_file(file\_path: str) -> str:

img = Image.open(file\_path)

processed = preprocess\_image(img)

return pytesseract.image\_to\_string(processed, config="--psm 6")

* **Why**: Converts the image into text.
* **Purpose**: pytesseract reads text from the preprocessed image.
* **Developer Note**: --psm 6 is used because it treats the image as a uniform block of text.

**3.2 PDFs**

def extract\_text\_from\_pdf\_file(file\_path: str) -> str:

pages = convert\_from\_bytes(pdf\_bytes)

text = ""

for page\_num, page in enumerate(pages, start=1):

processed = preprocess\_image(page)

page\_text = pytesseract.image\_to\_string(processed, config="--psm 6")

text += f"\n--- Page {page\_num} ---\n{page\_text}\n"

return text.strip()

* **Why**: PDFs may contain multiple pages.
* **Purpose**: Converts each page to an image → preprocess → OCR → concatenates results.
* **Developer Note**: Handles both local PDFs and URLs.

**4. Base64 Encoding**

def encode\_file\_to\_base64(file\_identifier: str) -> list:

...

* **Why**: The OpenRouter API supports sending images in **base64** for additional context.
* **Purpose**: Converts local files or URLs (PDFs/images) into base64 strings.
* **Developer Note**: PDFs are converted page by page to images.

**5. OpenRouter API Call**

def call\_openrouter(ocr\_text: str, file\_identifier: str) -> dict:

...

* **Why**: OCR text alone may be messy; an LLM can clean, normalize, and structure the data.
* **Workflow**:
  1. Sends MASTER\_PROMPT + OCR text + optional images to LLM.
  2. LLM returns **JSON output** according to schema.
  3. If JSON parsing fails, error is returned.
* **Developer Note**: Includes fallback logic in case the API fails.

**6. Main Processing Loop**

def process\_prescriptions():

results = []

# Process URLs from JSON

for entry in prescriptions:

...

ocr\_text = extract\_text\_from\_pdf\_file(url) # or image

parsed = call\_openrouter(ocr\_text, url)

if "error" in parsed: # fallback

parsed = { ... }

# Process local images

for filename in files:

...

* **Why**: Handles prescriptions from two sources:
  + URLs listed in JSON
  + Local files in images/ folder
* **Purpose**:
  + Extract OCR text
  + Send to OpenRouter LLM for structured extraction
  + Apply fallback if LLM fails
  + Save incremental results in OUTPUT\_FILE
* **Developer Note**: Keeps numbering consistent (prescription\_number) and ensures all results are captured.

**7. Output Management**

with open(OUTPUT\_FILE, "w", encoding="utf-8") as f:

json.dump(results, f, indent=2, ensure\_ascii=False)

* **Why**: Persist processed results.
* **Purpose**:
  + Indented for readability
  + ensure\_ascii=False keeps special characters intact
* **Developer Note**: Each prescription includes:
  + source (URL or filename)
  + parsed\_output (clean JSON)
  + Manual QA flags if needed

**8. Fallback Logic**

* **Why**: LLM may fail due to:
  + API errors
  + Unexpected input formats
* **Purpose**: Ensures no prescription is left unprocessed.
* **Fallback JSON** example:

parsed = {

"Patient\_Name": {"Value": None},

"Medicines": [],

"Manual\_QA\_Flags": ["LLM failed, use OCR-only"],

"Confidence\_Level": {"Overall": "low", "Reason": "LLM failed"}

}

**9. Execution**

if \_\_name\_\_ == "\_\_main\_\_":

process\_prescriptions()

* **Why**: Standard Python entry point.
* **Purpose**: Ensures process\_prescriptions() runs when the script is executed directly.

**Summary of Flow**

1. **Load inputs**: JSON URLs or local folder files
2. **Preprocess images/PDFs** → increase OCR accuracy
3. **Run OCR** → extract raw text
4. **Encode images to base64** (if needed)
5. **Send OCR + images to LLM (OpenRouter)** → get structured JSON
6. **Fallback** if LLM fails → keep OCR output minimal
7. **Save results** incrementally to JSON
8. **Repeat** for all inputs

✅ **Result**: Clean, normalized, and structured JSON for each prescription with **manual QA flags** for ambiguous fields.