The Advanced PDF-to-Summary Pipeline

This pipeline has two key innovations:

1. **Legend-First Processing:** We first identify and analyze the flowchart's legend (if one exists) to learn the custom shapes.
2. **Hybrid Data Extraction:** For each page, we extract text programmatically using a PDF library *and* use the LLM's vision capabilities to understand the layout, shapes, and connections. This combines the best of both worlds.

Phase 1: Context Acquisition (Handling Custom Shapes)

Before processing the main flowchart, we must learn its unique language.

Step 1: Identify the Legend Page
Often, the first or last page of a process document is a "Key" or "Legend" that defines the shapes. You can identify this either:

- **Manually:** For a one-off project, simply note which page is the legend.
- **Programmatically:** Use a PDF library to extract the text from the first/last few pages and search for keywords like "Legend," "Key," "Shape Definitions," etc.

Step 2: Extract Custom Shape Definitions with an LLM
Take an image of *only the legend page* and feed it to the multimodal LLM with a specialized prompt.

Prompt for Legend Extraction:

```
You are a system that reads documentation legends. Analyze the provided image of a flowchart legend page. Your task is to extract the meaning of each shape and return it as a single, valid JSON object mapping the shape's description to its meaning.

**Example Output Format:**
{
  "shape_definitions": {
    "Cloud": "Represents an external API call to a third-party service.",
    "Hexagon": "Indicates a mandatory security review step.",
    "Rectangle with wavy base": "Represents a printed document or report."
  }
}

Now, analyze the following image and provide the JSON output.
```

Outcome: You now have a `shape_definitions.json` file. This is the "context" or "domain knowledge" for your specific set of flowcharts.

Phase 2: Hybrid Per-Page Extraction

Now we process the actual flowchart pages, armed with our custom shape knowledge.

Step 1: Convert Page to Image (for Vision)
As before, convert the PDF page to a high-resolution PNG image. This will be used for visual analysis (detecting shapes, arrows, layout).

```
from pdf2image import convert_from_path
# This step remains the same
images = convert_from_path('my_flowchart.pdf', dpi=300)
```

Step 2: Extract High-Fidelity Text Data (Programmatically)
Instead of relying on the LLM's OCR, use a Python library like `PyMuPDF` (`fitz`) to extract all text on the page along with its precise coordinates. This is far more accurate than OCR.

```
import fitz # PyMuPDF

doc = fitz.open("my_flowchart.pdf")
page = doc[page_number] # The current page we are processing
text_data = page.get_text("json") # Gets text with coordinate data
```

The `text_data` will look something like a structured list of text blocks with their `(x0, y0, x1, y1)` bounding boxes.

Step 3: The Master Extraction Prompt (Improvised)
This is where we combine everything. We give the LLM the page *image*, our pre-extracted *text data*, and the *custom shape definitions*.

The Improvised Master Prompt:

```
You are an expert system for process analysis. Your task is to analyze an image of a flowchart
page and its corresponding text data to create a structured JSON output.

**Context - Custom Shape Definitions:**
{
  "Cloud": "Represents an external API call to a third-party service.",
  "Hexagon": "Indicates a mandatory security review step."
}
// Note: This section would be dynamically inserted from your shape_definitions.json
```

High-Fidelity Text Data from the Page:
// Insert the JSON output from PyMuPDF here.
// This provides perfect text, which you should map to the shapes you see.

Instructions:
1. Analyze the provided image to identify all nodes (shapes) and edges (arrows).
2. Use the "High-Fidelity Text Data" to determine the text inside each shape by matching text coordinates to shape locations. Do not rely on your own OCR unless absolutely necessary.
3. For each node, identify its shape. Use standard shape names (Oval, Rectangle, Diamond) but also use the custom shape names from the provided context (Cloud, Hexagon) when they apply.
4. Map all connections (edges) between nodes. For decisions, capture the edge labels ("Yes", "No").
5. Identify any off-page connectors and their labels.

Output Format:
Your output MUST be a single, valid JSON object with the keys "nodes", "edges", and "off_page_connectors".

```
{
  "nodes": [
    {
      "id": <integer>,
      "shape": "<Oval | Rectangle | Diamond | Cloud | Hexagon>", // Include custom shapes!
      "text": "<The text inside the shape from the provided data>"
    }
  ],
  "edges": [
    { "source_id": <integer>, "target_id": <integer>, "label": "<Yes, No, or null>" }
  ],
  "off_page_connectors": [
    { "id": <integer>, "direction": "<Out | In>", "label": "<A, B-1, etc.>" }
  ]
}
```

Now, analyze the following image using the provided context and text data.

Step 4: Execute and Consolidate

The rest of the process is similar to before:

- Loop through each page.
- Perform the hybrid extraction (programmatic text + LLM vision).
- Send the combined prompt to the LLM API.

- Save the resulting `page_N_data.json` file.
- After processing all pages, run a final script to stitch the pages together using the `off_page_connectors`, creating one unified graph of the entire process.

==

Analyzing the Request

I've analyzed the request. The focus is on transforming a flowchart image into a JSON structure. My initial steps will involve understanding the provided image and zoomed-in details. I'm now focusing on the necessary JSON format for the output.

Mapping the Strategy

I'm now mapping out the steps I'll take to simulate the hybrid extraction. I'll start by analyzing the provided OCR data and the flowchart image. The absence of a legend requires that I use standard flowchart conventions. I'll then construct a JSON output.

Refining the Process

I'm now focusing on the specifics of node and edge identification. I've identified the informational boxes and the starting connector. The swimlane titles provide context that can be ignored at this point. The next step will be to associate the OCR text with the appropriate shapes using the IDs already present in the diagram, such as node 0.

Constructing Node Definitions

I'm now focused on the structure of the nodes. I'm creating the node ID, shape type, associated text, and outgoing edges, including their labels. I'm currently defining nodes "Greeting", and 1, and 2 in the JSON schema, assigning unique IDs and relationships. I am using the OCR and image to determine the edge labeling.

Implementing the Logic

I'm now implementing the flowchart structure based on my analysis. I'm working through the nodes, assigning IDs and relationships. I am now associating each node with OCR text. The main task now is matching the OCR to the shapes and creating the `edges` array to link the nodes.

Constructing the Structure

I'm now focused on the relationships between the nodes and edges. I am creating the edge connections based on my shape analysis. The next step involves creating the node definitions and then combining everything into the complete JSON output, to complete the flowchart.

Focusing on Implementation

I'm now immersed in the specifics of transforming the identified shapes into JSON. The structure is solid, but the key is correctly associating OCR text and the established edges. I'm using descriptive labels, such as 'AssignEN' and 'GetPolicySymDetails', as node IDs. The work is proceeding smoothly.

Constructing the Node Structure

I'm now establishing the basic structure for each node: ID, shape type, text, and incoming/outgoing edges. The OCR text and image details provide all the information necessary for the JSON structure. I am systematically adding the node definitions in a structured manner.

Mapping the Structure

I'm now focusing on re-evaluating the descriptive node IDs I initially established, such as "Greeting" and "AssignEN," to ensure they align with the numeric IDs. This step will facilitate easier matching and the creation of the final JSON structure. I am using a key-value pair approach to ensure the mapping is correct, and the process is now underway.

Refining the Structure

I'm now focused on the "ConfirmZipCode" node, assigning its properties, edges, and corresponding OCR text. I am now identifying and mapping the nodes and ensuring that all nodes are linked correctly. I will then focus on the final JSON file to include these updates.

Reasoning & Methodology

1. **Context Acquisition:** I first scanned the image for a "Legend" or "Key." None was present. Therefore, I will use standard flowchart conventions for shape interpretation (e.g., Rounded Rectangle for prompts/processes, Parallelogram for user input, Rectangle for assignments, Circle/Oval for connectors). The blue-header rectangle for `GetPolicySymDetails` is treated as a `Sub-process`. The boxes at the very top are informational and not part of the active flow.
2. **Hybrid Data Extraction:** I used the image to identify all shapes, their relationships (arrows), and the labels on those arrows. I used your provided OCR text as the high-fidelity source for the content within each shape, mapping it to the visual elements.

3. **ID Assignment:** I have used the numeric labels inside the circles (0, 1, 2, 100, etc.) as the primary `id` for those nodes. For shapes without a pre-assigned number, I have assigned new, unique integer IDs (starting from 200) for consistency.
4. **Relationship Mapping:** I traced every arrow to define the `source_id` and `target_id` for each edge. I've also captured the text labels on the arrows (e.g., "-ANI Match-", "1", "2"). The implicit decision within the "Greeting and Language Offer" box is mapped as two separate paths originating from that node. The loop from the "Customer enters Policy/Phone number" input back to the "GetPolicySymDetails" step is also captured.

```
{
  "nodes": [
    {
      "id": 300,
      "shape": "Info",
      "text": "Geico Home Owners\n800-268-6146, DNIS 4035451\n800-350-9293, DNIS 4035031"
    },
    {
      "id": 301,
      "shape": "Info",
      "text": "DN: ASP_DN_4035451_8002686146\nDN: ASP_DN_4035031_8003509293"
    },
    {
      "id": 0,
      "shape": "Oval",
      "text": "0"
    },
    {
```

"id": 200,

"shape": "Rounded Rectangle",

"text": "Greeting and Language Offer\nThank you for calling Assurant, servicer of the single-family home insurance program, for Geico. Para espanol oprima numero ocho?\nBargeIn=FALSE\nNOTE: Greeting should be dynamic based on DNIS"

},

{

"id": 1,

"shape": "Oval",

"text": "1"

},

{

"id": 2,

"shape": "Oval",

"text": "2"

},

{

"id": 201,

"shape": "Rectangle",

"text": "Assign: Language = EN"

},

{

"id": 202,

"shape": "Rectangle",

"text": "Assign: Language = ES\nNOTE: call flow will proceed in Spanish language"

```
},
{
  "id": 203,
  "shape": "Sub-process",
  "text": "GetPolicySymDetails\n(ANI lookup)"
},
{
  "id": 100,
  "shape": "Oval",
  "text": "100"
},
{
  "id": 101,
  "shape": "Oval",
  "text": "101"
},
{
  "id": 102,
  "shape": "Oval",
  "text": "102"
},
{
  "id": 103,
  "shape": "Oval",
      "text": "103"
    },
    {
      "id": 204,
      "shape": "Rounded Rectangle",
      "text": "Account Verfication Options\nIf you want to locate your policy by policy number, press 1. To search by phone number, press 2"
    },
    {
      "id": 104,
      "shape": "Oval",
      "text": "104"
    },
    {
      "id": 105,
      "shape": "Oval",
      "text": "105"
    },
    {
      "id": 205,
      "shape": "Rounded Rectangle",
      "text": "Collect Zip code\nPlease enter the 5 digit zip code of the covered location."
    },
    {
      "id": 206,
    "shape": "Parallelogram",
    "text": "Customer enters zip code"
  },
  {
    "id": 107,
    "shape": "Oval",
    "text": "107"
  },
  {
    "id": 207,
    "shape": "Rounded Rectangle",
    "text": "Confirm Zip code\nYou entered <zip code>."
  },
  {
    "id": 208,
    "shape": "Rounded Rectangle",
    "text": "Collect Policy Number\nPlease enter your policy number, which should be either 7 or 10 digits long. Ensure there are no letters included."
  },
  {
    "id": 209,
    "shape": "Rounded Rectangle",
    "text": "Collect Phone Number\nPlease enter the telephone number including the area code for the property covered"
  },
    {
      "id": 210,
      "shape": "Rectangle",
      "text": "Assign: dataEntry = PolicyNum"
    },
    {
      "id": 211,
      "shape": "Rectangle",
      "text": "Assign: dataEntry = PhoneNum"
    },
    {
      "id": 212,
      "shape": "Parallelogram",
      "text": "Customer enters Policy/Phone number"
    }
  ],
  "edges": [
    { "source_id": 0, "target_id": 200, "label": null },
    { "source_id": 200, "target_id": 1, "label": "!=8" },
    { "source_id": 200, "target_id": 2, "label": null },
    { "source_id": 1, "target_id": 201, "label": null },
    { "source_id": 2, "target_id": 202, "label": null },
    { "source_id": 201, "target_id": 203, "label": null },
    { "source_id": 202, "target_id": 203, "label": null },
    { "source_id": 203, "target_id": 100, "label": "-ANI Match-" },
    { "source_id": 203, "target_id": 101, "label": "-ANI NM-" },
    { "source_id": 203, "target_id": 102, "label": "API Error" },
    { "source_id": 100, "target_id": 103, "label": null },
    { "source_id": 101, "target_id": 204, "label": null },
    { "source_id": 103, "target_id": 205, "label": null },
    { "source_id": 204, "target_id": 104, "label": "1" },
    { "source_id": 204, "target_id": 105, "label": "2" },
    { "source_id": 205, "target_id": 206, "label": "NI3/NM3" },
    { "source_id": 206, "target_id": 107, "label": null },
    { "source_id": 107, "target_id": 207, "label": null },
    { "source_id": 104, "target_id": 208, "label": null },
    { "source_id": 105, "target_id": 209, "label": null },
    { "source_id": 208, "target_id": 210, "label": null },
    { "source_id": 209, "target_id": 211, "label": null },
    { "source_id": 210, "target_id": 212, "label": null },
    { "source_id": 211, "target_id": 212, "label": null },
    { "source_id": 212, "target_id": 203, "label": null }
  ],
  "off_page_connectors": []
}
```