**Research Report: Extracting "Approved Makes and Manufacturer" Tables from PDFs**

**1. Executive Summary**

This report addresses the challenges of extracting the "Approved Makes and Manufacturer" tables from over 100 unstructured PDF documents. The documents include digital PDFs as well as scanned and handwritten versions. The proposed solution uses a hybrid approach that leverages both OCR and PDF parsing libraries, with fallback processing via language models if necessary. The report details the problem statement, analysis of challenges, a comparison of available methods and tools, the final selected approach, and potential risks with their mitigations.

**2. Problem Statement**

The goal is to extract the "Approved Makes and Manufacturer" table from a wide variety of PDFs into a structured JSON format. Key challenges include:

* **Variability:** Different table formats (bordered, borderless, merged cells) across documents.
* **Multi-Page Tables:** Tables that span multiple pages need to be merged.
* **OCR Challenges:** Handling low-quality scans, handwritten text, and skewed layouts.
* **Scalability:** The solution must generalize well to unseen documents and handle edge cases gracefully.

**3. Problem Breakdown**

| **Challenge** | **Details** |
| --- | --- |
| **Document Types** | Mix of digital PDFs (text-based) and scanned/handwritten PDFs (image-based). |
| **Table Identification** | Detect tables with titles like "Approved Makes" despite inconsistent phrasing. |
| **Structural Variability** | Tables may lack borders, have merged cells, or irregular row/column splits. |
| **Multi-Page Continuity** | Split tables across pages must be merged into a single structured output. |
| **Error Handling** | Avoid silent failures; flag low-confidence extractions for manual review. |

**4. Methods & Tools Comparison**

**A. OCR-Based Extraction**

* **Tesseract OCR**
  + **Pros:** Free, open-source, integrates with Python.
  + **Cons:** Poor table detection; requires extensive manual post-processing.
* **AWS Textract**
  + **Pros:** High accuracy with built-in table detection, handles multi-page documents, and supports OCR for scanned/handwritten content.
  + **Cons:** Cost is incurred at approximately $0.0015 per page and it relies on cloud infrastructure.

**B. PDF Parsing Libraries**

* **Camelot**
  + **Pros:** Specializes in table extraction (supports both lattice and stream modes), works well for digital PDFs.
  + **Cons:** Struggles with scanned/handwritten PDFs.
* **PyMuPDF**
  + **Pros:** Fast extraction of text and coordinates, suitable for digital PDFs.
  + **Cons:** Lacks built-in table detection; requires custom heuristics for table structure.

**C. LLMs for Post-Processing**

* **GPT-4 / Llama 2**
  + **Pros:** Provides contextual understanding to parse ambiguous text into tables.
  + **Cons:** High latency and cost (approximately $0.03 per 1k tokens), and there is a risk of hallucination.

**D. Hybrid Approaches**

* **Hybrid (AWS Textract + Camelot)**
  + **Pros:** High accuracy, variable cost, handles multi-page and complex tables, and supports scanned/handwritten documents when needed.
  + **Cons:** Increased complexity and potential latency issues.

**5. Comparison Table**

| **Tool/Method** | **Accuracy** | **Cost** | **Latency** | **Complexity** | **Handwriting Support** | **Multi-Page** |
| --- | --- | --- | --- | --- | --- | --- |
| **Tesseract** | Low | Free | High | High | Partial | No |
| **AWS Textract** | High | $0.0015/page | Medium | Low | Yes | Yes |
| **Camelot** | Medium | Free | Low | Medium | No | Limited |
| **PyMuPDF** | Medium | Free | Low | Medium | No | Limited |
| **GPT-4 / Llama 2** | Variable | $0.03/1k tokens | High | High | No | Yes |
| **Hybrid (Textract + Camelot)** | High | Variable | Medium | High | Yes | Yes |

**6. Final Approach Selection**

**Primary Pipeline:**

1. **Document Type Detection:**
   * Use PyPDF2 (or similar) to detect whether a PDF is digital or scanned/handwritten.
2. **Extraction Based on Document Type:**
   * **Digital PDFs:**
     + Use **Camelot** (lattice mode for bordered tables) in combination with **PyMuPDF** for extracting text coordinates.
   * **Scanned/Handwritten PDFs:**
     + Use **AWS Textract** to perform OCR and table extraction.
3. **Validation:**
   * Apply regular expressions (regex) to confirm the presence of table titles such as "Approved Makes" and "Manufacturer" in the extracted data.
4. **Fallback Processing:**
   * For ambiguous or low-confidence cases, use **GPT-4** to process and parse raw text into a structured JSON format.

**Rationale:**

* **Accuracy:** AWS Textract outperforms alternatives in processing scanned/handwritten PDFs.
* **Cost Efficiency:** Utilize Camelot (free) for digital PDFs and reserve AWS Textract for cases that require higher accuracy.
* **Multi-Page Handling:** AWS Textract provides native support for multi-page tables, while Camelot may need additional logic.
* **Error Handling:** Files with missing table headers or low confidence in extraction are flagged for review.

*Note:* LLMs are used only as a fallback due to their higher cost and latency.

**7. Risks & Mitigations**

* **Risk:** Camelot might miss borderless tables.  
  **Mitigation:** Integrate **PDFPlumber** to detect text alignment and whitespace boundaries.
* **Risk:** AWS Textract could misread handwritten text.  
  **Mitigation:** Implement a confidence threshold (e.g., discard results with less than 90% confidence).
* **Risk:** High processing time for large PDF batches.  
  **Mitigation:** Use Python’s multiprocessing to parallelize the extraction process.

**8. Conclusion**

The proposed hybrid approach effectively balances accuracy, cost, and scalability. By using AWS Textract for scanned/handwritten documents and Camelot for digital PDFs, most extraction scenarios are covered. GPT-4 is reserved for fallback scenarios to handle ambiguous cases. This strategy not only addresses the variability in PDF formats and table structures but also incorporates robust error handling and multi-page continuity, ensuring a comprehensive solution for the TenderBot assignment.

Bottom of Form