Lab Report Digitization with OCR, Rule-Based Extraction, and HITL-Guided Learning

Abstract

This project delivers an end-to-end system that converts unstructured laboratory reports (PDF or images) into structured JSON comprising patient demographics and test results with confidence scores. The pipeline performs robust preprocessing (deskew, denoise, binarize) to improve OCR fidelity, extracts tokens with positional metadata, reconstructs readable lines, and applies post-processing to remove headers/footers, reference ranges, and noise. A rule-based extractor then identifies patient fields and common assays using resilient regex and heuristics. A human-in-the-loop (HITL) workflow lets users submit corrected JSON outputs via an API; after five or more curated corrections, a lightweight memory-based model trains and blends with the rule-based confidences to raise certainty for previously seen fields. All file I/O is anchored to the project directory to avoid path drift, ensuring deterministic behavior when running the FastAPI service. The result is a practical, locally runnable system suitable for heterogeneous clinical report templates, offering rapid adaptation through low-effort supervision.

Methodology

1. **Preprocessing and Rendering**

- Input handling: The service accepts PDF and image uploads through a simple web UI backed by FastAPI endpoints. PDFs are split into pages and rendered to images at approximately 300 DPI to balance quality and performance.
- Image cleanup: To improve OCR outcomes, images are deskewed using Hough-based line detection, then denoised and binarized (adaptive threshold). Morphological closing reduces small gaps. Multi-page PDFs yield page_XX.png files in a processed directory for traceability.
- Path control: All paths are strictly anchored to the project root using a small helper (e.g., P(...)). The working directory is forced to the project folder to avoid external reloader effects and ensure data, outputs, and models persist in known subdirectories.

2. **OCR and Tokenization**

- OCR configuration: Tesseract runs in a mode suited for block text with mixed fonts and tables. The system uses image_to_data to obtain word-level text, confidence, and bounding boxes for downstream grouping.
- Token structuring: Tokens are normalized and grouped into lines by proximity in the vertical axis, with left-to-right ordering. Each page produces a tokens_page_XX.json file containing text, coordinates, and confidence details for auditability.

3. **Line Reconstruction and Post-Processing**

- Line normalization: A normalization pass collapses whitespace and removes lines matching headers, footers, addresses, metadata (page numbers, postal codes), and pure reference ranges. This reduces spurious rows that would otherwise appear as “tests.”
- Wrapped-line merging: Subsequent lines without obvious numeric values are joined to the previous line to counter OCR line breaks in the middle of tests.

- Flexible row extraction: A permissive pattern captures “Name Value [Unit]” even in the presence of OCR artifacts, stray punctuation, or spacing. A canonicalization map consolidates synonyms (e.g., “LDL” to “LDL Cholesterol (Direct)”, “HbA1C” normalization), stabilizing downstream keys and de-duplication.
- Dedupe and blend: Rows from post-processing and rule-based modules are merged by (name, unit), keeping the highest-confidence candidate.

4. Rule-Based Extraction

- Patient fields: Regex heuristics scan early lines and global text to extract age, gender, patient ID, test/report dates, and often the full patient name. Confidence reflects pattern specificity, plausibility ranges (e.g., age bounds), and line positioning.
- Tests: Patterns detect common analytes and formats across varied layouts, including fallback for tabular sections. This stage acts as a conservative baseline that prioritizes precision.

5. Human-in-the-Loop (HITL) Corrections

- Correction workflow: The UI exposes a POST endpoint to submit “original” and “corrected” JSON. Each submission is stored under data/corrections/ as a standalone artifact.
- Training trigger: After five or more corrections are present, the next correction submission triggers training of a simple memory-based extractor that remembers corrected field values per key (e.g., frequent patient fields).

6. ML Blending and Persistence

- Lightweight model: The memory-based extractor acts like a calibrated whitelist that boosts confidence for values seen in corrections. This strikes a pragmatic balance between adaptability and complexity for limited data.
- Confidence blending: At inference, rule-based confidence is blended with the memory score to produce higher confidence in fields that match learned entries. Model state persists to models/ so improvements persist across runs.

7. API and Demo UI

- Endpoints: POST /upload processes files and returns structured JSON, POST /correct accepts corrections and triggers training when the dataset is sufficient, GET /health reports subsystem status, and GET /stats summarizes processed reports and training signals.
- Demo: A simple page displays patient fields, test tables, and confidence bars, alongside the raw JSON. This supports fast manual review and correction creation.

8. Engineering Considerations

- Deterministic I/O: Forcing the working directory to the project root and using path helpers ensures all artifacts live under data/, outputs/, and models/.
- Windows support: The system exposes config for Tesseract and Poppler paths. Temporary upload handling is guarded to prevent Windows file-lock issues.
- Extensibility: The post-processing and canonicalization lists are designed for incremental extension as new templates are observed.