Handwritten PDF Extractor — Technical Report

1. Executive Summary

The Handwritten PDF Extractor is a web-based tool that leverages AI to convert scanned or handwritten PDF documents into structured digital text. Users can upload a PDF, receive an accurate transcription in Markdown, view an AI-generated summary, and translate the summary into multiple languages. The solution comprises a FastAPI backend, a static HTML JavaScript frontend, and integration with Google's Gemini generative model and Google Translate.

2. Approach & Technology Stack

- **Backend (FastAPI)**
    - **Framework:** FastAPI for high-performance, asynchronous API endpoints.
    - **OCR & PDF Rendering:** PyMuPDF (fitz) to convert each page into a PNG image for AI processing.
    - **Generative AI:** Google Generative Language API (Gemini) to transcribe handwritten text and summarize content.
    - **Translation:** googletrans library to translate summaries via Google Translate.
    - **CORS:** Starlette's CORSMiddleware configured to allow only the frontend origin in production.
    - **Deployment:** Uvicorn server, containerized on Render.com, with environment variables for secrets.
- **Frontend (HTML + Vanilla JS)**
    - **UI Framework:** Custom responsive CSS with Flexbox and media queries.
    - **Markdown Rendering:** marked.js to convert AI-generated Markdown into HTML.
    - **File Handling:** Drag & drop and file input control to select PDFs.
    - **Async Requests:** fetch() calls to /extract-handwriting/ and /translate/ endpoints.
    - **UX Enhancements:** Progress bar, tabbed results view, copy/download buttons, and language selector.

3. Key Challenges & Solutions

Challenge	Solution
API Key Management	Used python-dotenv locally and Render environment variables for secure key injection.
Asynchronous Translation Call	Updated googletrans.translate invocation to be awaited in FastAPI async endpoints.
Markdown-to-HTML Rendering	Integrated marked.js and updated the frontend to parse Markdown rather than text.
CORS Configuration for Security	Replaced wildcard origins with a single allowed origin via FRONTEND_URL environment var.
PDF Page Extraction Performance	Tuned PyMuPDF rendering matrix and handled exceptions to avoid resource leaks.
Error Handling & User Feedback	Centralized try/except blocks in backend, JSONResponse errors, and frontend alert banners.

4. Testing & Validation

- **Manual QA**
  - Validated summary coherence against ground-truth documents.
  - Tested translations into Hindi, Tamil, and Marathi to confirm correct language codes and no UI regressions.
  - Ran frontend on both desktop (Chrome, Firefox) and mobile viewports to ensure responsive layouts.
- **Performance Checks**
  - Measured average API response times on Render, optimized image extraction to keep per-page processing under 1.5 seconds.
  - Confirmed memory usage under 200MB and no significant CPU spikes during multi-page uploads.

5. Conclusion & Next Steps

The Handwritten PDF Extractor successfully demonstrates an end-to-end AI pipeline for OCR, summarization, and translation. Future improvements include:

- Adding user authentication and per-user document history.
- Integrating a more robust translation model (e.g., Gemini multilingual) for better fidelity.
- Implementing batch uploads and background processing with WebSocket's for real-time progress updates.
- Expanding testing to include end-to-end Cypress tests for UI workflows.