AI-POWERED AMOUNT DETECTION IN MEDICAL DOCUMENTS
PROJECT REPORT

Prepared By: Mishal V S

Date: 22/01/2026

AI-POWERED AMOUNT DETECTION IN MEDICAL DOCUMENTS
PROJECT REPORT

1. INTRODUCTION

Medical bills and receipts often arrive in unstructured formats such as scanned images, crumpled paper bills, or poorly formatted text. Extracting financial information like total bill amount, paid amount, and due amount from such documents is challenging due to OCR errors, noise, and inconsistent layouts.

This project implements an AI-powered backend service that extracts financial amounts from medical bills or receipts using a multi-stage pipeline involving OCR, numeric normalization, and context-based classification. The system is designed to handle both text and image inputs and return structured JSON output with provenance and guardrails.

2. PROBLEM STATEMENT OVERVIEW

The objective is to design a backend service that performs:

1. OCR / Text Extraction
2. Numeric Token Detection and Normalization
3. Context-based Classification of Amounts
4. Final Structured JSON Output with Provenance

The system must handle noisy OCR outputs, digit misrecognition (e.g., l → 1, O → 0), and partial document visibility.

3. SYSTEM ARCHITECTURE

The system follows a modular pipeline-based architecture:

Input
→ OCR / Text Extraction
→ Numeric Token Extraction
→ Normalization
→ Context Classification
→ Structured JSON Output

The architecture is implemented as a FastAPI backend with a clean separation between API routing and business logic.

Main components:

- FastAPI Application Layer (API endpoints)
- OCR & Processing Utilities (utils.py)
- Temporary File Handling Layer
- Output Validation & Guardrails

4. TECHNOLOGY STACK

Backend Framework: FastAPI

OCR Engine: Tesseract OCR (via pytesseract)

Image Processing: Pillow (PIL)

Language: Python 3

Server: Uvicorn

Deployment: Localhost + Ngrok Tunnel

5. STEP-BY-STEP PIPELINE IMPLEMENTATION

5.1 STEP 1 – OCR / TEXT EXTRACTION

The system supports two input types:

- Raw text input
- Image upload (jpg/png)

For image input:

- Image is converted to grayscale
- Thresholding is applied to reduce noise
- Tesseract OCR extracts raw text

Extracted output is treated as unstructured text.

Guardrail:
If no text or numeric tokens are detected, the pipeline exits early with: {"status": "no_amounts_found", "reason": "document too noisy"}

5.2 STEP 2 – NUMERIC TOKEN EXTRACTION & NORMALIZATION

Raw numeric tokens are extracted using regex patterns that capture:

- Digits
- OCR noise characters (l, O)
- Commas and decimals
- Percentages

Normalization logic:

- Replace OCR errors (l → 1, O → 0)
- Remove commas and non-numeric symbols
- Convert valid values to integers

- Ignore percentages during numeric normalization

Example:
Raw Tokens: ["l200", "1000", "200", "10%"]

Normalized Amounts: [1200, 1000, 200]

Each step produces confidence metadata.

5.3 STEP 3 – CONTEXT-BASED CLASSIFICATION

Classification is done using keyword proximity and regex matching.

Supported classifications:

- total_bill
- paid
- due

The system searches for numeric values associated with keywords such as:

- "total"
- "paid"
- "due"

OCR keyword corrections are applied selectively (e.g., "t0tal" → "total", "pald" → "paid").

Each detected amount includes provenance showing where it was found in the original text.

5.4 STEP 4 – FINAL STRUCTURED OUTPUT

The final output contains:

- Currency
- Labeled financial amounts
- Source text

- Status indicator

Example final output:

{
"currency": "INR",
"amounts": [
{"type": "total_bill", "value": 3096, "source": "text: 'Total Bill Amount 3096'"},
{"type": "paid", "value": 3000, "source": "text: 'Amount Paid 3000'"},
{"type": "due", "value": 96, "source": "text: 'Balance Due 96'"}
],
"status": "ok"
}

6. API DESIGN

6.1 Health Check Endpoint

GET /health

Response:

{"status": "ok"}

Used to verify server availability.

6.2 Extract Amounts Endpoint

POST /extract-amounts

Accepts multipart/form-data:

- text (optional)
- file (optional)

Validation:

- At least one input must be provided

Returns:

- Step-by-step pipeline output
- Final structured result

---

7. ERROR HANDLING & GUARDRAILS

Implemented guardrails:

- Missing input validation
- OCR noise detection
- Empty token detection
- Graceful error responses with clear reasons

Examples:

- Missing input → 400 error
- No numeric tokens → early pipeline exit
- OCR failure → handled safely

---

8. TESTING STRATEGY

Testing was performed using:

- FastAPI Swagger UI
- PowerShell Invoke-WebRequest
- Curl commands
- Ngrok public URL testing

Test cases included:

- Clean text input
- Noisy OCR image

- Medical bills with multiple numbers
- Invalid inputs

9. DEPLOYMENT

The application runs locally using Uvicorn and is exposed publicly using Ngrok.

Ngrok enables evaluators to test the API without local setup.

10. PROJECT COMPLETION STATUS

All required steps from the problem statement are implemented:

- OCR handling
- Numeric normalization
- Context classification
- Guardrails
- Structured JSON output
- API documentation
- Local and public demo

11. CONCLUSION

This project successfully demonstrates an end-to-end AI-powered pipeline for extracting financial amounts from medical documents. The modular architecture ensures extensibility, while guardrails and normalization techniques improve reliability under noisy real-world conditions.

The solution meets all requirements outlined in the SDE Intern Assignment and is ready for evaluation.