🏥 Medical Report Simplifier

An AI-powered backend service that processes medical reports (text or images) and provides patient-friendly explanations. Built for the SDE Intern Assignment focusing on OCR → Test Extraction → Plain-Language Explanation.

Python 3.9+

FastAPI 0.68+

License MIT

🎯 Problem Statement Solution

This service implements a complete 4-step pipeline for medical report processing:

1. **OCR/Text Extraction** - Extract and clean medical test data
2. **Normalization** - Standardize test names, values, units, and ranges
3. **Patient-Friendly Summary** - Generate simple explanations
4. **Final Output** - Return validated, normalized results

✨ Features

- 🔍 **Multi-format Input**: Text and image processing (PNG, JPG, PDF)
- 📄 **OCR Processing**: Tesseract-based text extraction with confidence scoring
- 🤖 **AI-Powered OCR Fixing**: Gemini AI corrects common OCR errors and typos
- 📊 **Smart Normalization**: Standardizes medical test names, values, units, and reference ranges
- 💬 **Patient-Friendly Explanations**: Simple, non-technical summaries using Gemini AI
- 🛡️ **Hallucination Prevention**: AI semantic validation prevents fabricated test results
- ⚡ **4-Step Pipeline**: Complete processing from raw input to final output
- 🚨 **Error Handling**: Proper validation with "unprocessed" status for invalid inputs

PROF

🏗️ Architecture

```
Input (Text/Image)
     ↓
Step 1: OCR/Text Extraction + AI Error Fixing
     ↓
Step 2: AI-Powered Normalization
     ↓
Step 3: Patient-Friendly Summary Generation
     ↓
Step 4: AI Semantic Validation & Final Output
```

🚀 API Endpoints

Production Endpoints

- `POST /api/v1/process-text` - Process text input (returns final output)
- `POST /api/v1/process-image` - Process image input (returns final output)

Demo/Evaluation Endpoints

- `POST /api/v1/demo-problem-statement` - Shows exact 4-step format from assignment
- `POST /api/v1/debug-steps` - Detailed step-by-step processing for text
- `POST /api/v1/debug-steps-image` - Detailed step-by-step processing for images

Utility

- `GET /api/v1/health` - Health check

🛠️ Tech Stack

- **FastAPI** - Modern, fast web framework for building APIs
- **Google Gemini AI** - Advanced language model for text processing and explanations
- **Tesseract OCR** - Optical character recognition for image text extraction
- **Pydantic** - Data validation and serialization
- **Python 3.9+** - Core programming language
- **python-multipart** - File upload handling
- **python-magic** - File type detection
- **Pillow** - Image processing

📋 Setup Instructions

Prerequisites

1. **Python 3.9+** installed

2. **Tesseract OCR** installed:

   ```
   # Ubuntu/Debian
   sudo apt-get update
   sudo apt-get install tesseract-ocr

   # macOS
   brew install tesseract

   # Windows - Download from: https://github.com/UB-Mannheim/tesseract/wiki
   ```

3. **Google Gemini API Key**:

   - Go to Google AI Studio
   - Create a new API key
   - Copy the API key for configuration

Local Installation

1. **Clone the repository**:

```
git clone https://github.com/yourusername/medical-report-
simplifier.git
cd medical-report-simplifier
```

2. **Create virtual environment**:

```
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3. **Install dependencies**:

```
pip install -r requirements.txt
```

4. **Configure environment**:
   Create a `.env` file in the root directory:

```
touch .env
```

   Add your Google Gemini API key:

```
GEMINI_API_KEY=your_api_key_here
AI_TEMPERATURE=0.3
VALIDATION_CONFIDENCE_THRESHOLD=0.7
MAX_FILE_SIZE=10485760
```


Running the Application

1. **Start the server**:

```
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
```

2. **Access the API**:

   - **API Documentation**: http://localhost:8000/docs
   - **Health Check**: http://localhost:8000/api/v1/health

- **Alternative docs**: http://localhost:8000/redoc

🛰️ API Usage Examples

1. Process Text Input (Production)

```
curl -X POST "http://localhost:8000/api/v1/process-text" \
  -H "Content-Type: application/json" \
  -d '{
    "text": "CBC: Hemoglobin 10.2 g/dL (Low), WBC 11,200 /uL (High)"
  }'
```

2. Process Image Input (Production)

```
curl -X POST "http://localhost:8000/api/v1/process-image" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@medical_report.png"
```

3. Demo Problem Statement Format (Perfect for Evaluation)

```
curl -X POST "http://localhost:8000/api/v1/demo-problem-statement" \
  -H "Content-Type: application/json" \
  -d '{
    "text": "CBC: Hemglobin 10.2 g/dL (Low), WBC 11,200 /uL (Hgh)"
  }'
```

4. Debug Step-by-Step Processing

PROF

```
curl -X POST "http://localhost:8000/api/v1/debug-steps" \
  -H "Content-Type: application/json" \
  -d '{
    "text": "CBC: Hemoglobin 8.5 g/dL (Low), WBC 15000 /uL (High)"
  }'
```

Expected Response Formats

Final Output Format (Production):

```
{
  "tests": [
    {
```

```
      "name": "Hemoglobin",
      "value": 10.2,
      "unit": "g/dL",
      "status": "low",
      "ref_range": {"low": 12.0, "high": 15.0}
    },
    {
      "name": "WBC",
      "value": 11200,
      "unit": "/uL",
      "status": "high",
      "ref_range": {"low": 4000, "high": 11000}
    }
  ],
  "summary": "Low hemoglobin and high white blood cell count.",
  "explanations": ["Low hemoglobin may indicate anemia.", "High WBC can
occur with infections."],
  "status": "ok"
}
```

Error Response Format:

```
{
  "status": "unprocessed",
  "reason": "No medical tests found in input text"
}
```

4-Step Demo Format (Problem Statement):

```
{
  "step1_ocr_extraction": {
    "tests_raw": ["CBC: Hemglobin 10.2 g/dL (Low), WBC 11,200 /uL
(Hgh)"],
    "confidence": 0.95
  },
  "step2_normalized_tests": {
    "tests": [...],
    "normalization_confidence": 0.95
  },
  "step3_patient_friendly": {
    "summary": "...",
    "explanations": [...]
  },
  "step4_final_output": {
    "tests": [...],
    "summary": "...",
    "status": "ok"
```

```
  }
}
```

🏗️ Architecture

```
medical-report-simplifier/
├── app/
│   ├── main.py                    # FastAPI application entry point
│   ├── models/
│   │   └── schemas.py             # Pydantic models for
request/response
│   ├── services/
│   │   ├── ocr_service.py         # OCR and text extraction
│   │   ├── ai_normalization_service.py  # Medical test normalization
│   │   ├── ai_service.py          # Gemini AI integration & validation
│   │   ├── processing_service.py  # Main processing pipeline
│   │   └── medical_data.py        # Medical reference data
│   ├── api/
│   │   └── endpoints.py           # API route handlers
│   └── core/
│       ├── config.py              # Configuration management
│       └── utils.py               # Utility functions
├── data/
│   └── medical_references.json   # Medical test reference data
├── tests/
│   ├── test_api.py               # API endpoint tests
│   └── test_validation.py       # Validation tests
├── requirements.txt              # Python dependencies
├── test_validation.py           # Demo validation script
└── README.md                    # This file
```

🧪 Testing

PROF

Run Built-in Tests

```
pytest tests/ -v
```

Run Validation Demo

```
python test_validation.py
```

Manual Testing with Postman

Import the following collection to test all endpoints:

- Health check
- Text processing
- Image processing
- Debug endpoints
- Demo format

🚀 Deployment Options

🚂 Railway Deployment (Recommended)

Railway is perfect for this project because it supports both Python and automatic environment detection.

Step 1: Prepare for Railway

1. **Create railway.json** (Railway configuration):

```
{
  "$schema": "https://railway.app/railway.schema.json",
  "build": {
    "builder": "NIXPACKS"
  },
  "deploy": {
    "startCommand": "uvicorn app.main:app --host 0.0.0.0 --port $PORT"
  }
}
```

2. **Create Procfile** (Alternative start command):

```
web: uvicorn app.main:app --host 0.0.0.0 --port $PORT
```

PROF

3. **Update requirements.txt** to include all dependencies:

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
python-multipart==0.0.6
python-magic==0.4.27
Pillow==10.1.0
pytesseract==0.3.10
google-generativeai==0.3.2
pydantic==2.5.0
pydantic-settings==2.1.0
python-dotenv==1.0.0
```

Step 2: Deploy to Railway

1. **Push to GitHub**:

```
git add .
git commit -m "Prepare for Railway deployment"
git push origin main
```

2. **Deploy on Railway**:
   - Go to railway.app
   - Sign in with GitHub
   - Click "New Project" → "Deploy from GitHub repo"
   - Select your repository
   - Railway will automatically detect it's a Python project

3. **Configure Environment Variables**:
   - In Railway dashboard, go to your project
   - Click "Variables" tab
   - Add:

     ```
     GEMINI_API_KEY=your_api_key_here
     AI_TEMPERATURE=0.3
     VALIDATION_CONFIDENCE_THRESHOLD=0.7
     MAX_FILE_SIZE=10485760
     ```

4. **Custom Start Command** (if needed):
   - In Railway dashboard, go to "Settings"
   - Under "Deploy", set start command:

     ```
     uvicorn app.main:app --host 0.0.0.0 --port $PORT
     ```

PROF

Step 3: Access Your Deployed API

- Railway will provide a URL like: `https://your-app-name.railway.app`
- API docs: `https://your-app-name.railway.app/docs`
- Health check: `https://your-app-name.railway.app/api/v1/health`

🌐 Local Demo with ngrok

For quick demo without cloud deployment:

1. **Install ngrok**: https://ngrok.com/download
2. **Start the FastAPI server**:

```
uvicorn app.main:app --host 0.0.0.0 --port 8000
```

3. **Expose with ngrok**:

```
ngrok http 8000
```

4. **Use the ngrok URL** for testing: `https://abc123.ngrok.io`

🔧 Other Cloud Options

- **Render**: Auto-deploy from GitHub, similar to Railway
- **Heroku**: Use Procfile, add Tesseract buildpack
- **DigitalOcean App Platform**: Deploy from GitHub
- **AWS/GCP/Azure**: Container deployment with Docker

🛡️ AI-Powered Hallucination Validation

The Problem

Traditional validation methods use simple string matching, which can miss:

- Fabricated test results that weren't in the original data
- Incorrect values or units
- Completely invented medical tests

Our Solution: AI Semantic Context Comparison

We use Gemini AI to compare the **semantic context** between original text and normalized results:

1. **Semantic Analysis**: AI understands the meaning of medical tests, not just keywords
2. **Context Preservation**: Validates that normalized data represents the same medical information
3. **Confidence Scoring**: Returns confidence levels for validation decisions
4. **Hallucination Detection**: Identifies when AI normalization adds non-existent tests

Validation Process

```
# 1. Compare original vs normalized semantically
is_valid, validation_error =
self.ai_service.validate_against_hallucination(
    original_tests,     # Raw extracted text
    normalized_tests    # AI-normalized results
)

# 2. Return appropriate response
if not is_valid:
    return ErrorResponse(
```

```
        status="unprocessed",
        reason=f"hallucinated tests not present in input:
{validation_error}"
    )
```

🎬 Demo Script

For your screen recording, use this sequence:

```
# 1. Start server
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000

# 2. Health check
curl http://localhost:8000/api/v1/health

# 3. Basic text processing (show final output)
curl -X POST "http://localhost:8000/api/v1/process-text" \
  -H "Content-Type: application/json" \
  -d '{"text": "CBC: Hemoglobin 10.2 g/dL (Low), WBC 11,200 /uL
(High)"}'

# 4. Demo 4-step format (perfect for evaluation)
curl -X POST "http://localhost:8000/api/v1/demo-problem-statement" \
  -H "Content-Type: application/json" \
  -d '{"text": "CBC: Hemglobin 10.2 g/dL (Low), WBC 11,200 /uL (Hgh)"}'

# 5. Error handling (should return "unprocessed")
curl -X POST "http://localhost:8000/api/v1/process-text" \
  -H "Content-Type: application/json" \
  -d '{"text": "This is just regular text without medical data."}'

# 6. Image processing (if you have a test image)
curl -X POST "http://localhost:8000/api/v1/process-image" \
  -F "file=@test_medical_report.png"
```


📊 Performance & Scalability

- **Response Time**: ~2-3 seconds for text processing
- **AI Processing**: Parallel OCR fixing and normalization
- **File Upload**: Supports up to 10MB images
- **Rate Limiting**: Configurable via settings
- **Caching**: Medical reference data cached in memory

🔒 Security Features

- File type validation (magic number checking)
- File size limits
- Input sanitization

- API key protection via environment variables
- No persistent storage of medical data

📝 Assignment Compliance

This implementation fully addresses the problem statement:

✅ **Step 1**: OCR/Text Extraction with confidence scoring
✅ **Step 2**: Normalized Tests JSON with standardized format
✅ **Step 3**: Patient-Friendly Summary with simple explanations
✅ **Step 4**: Final Output with combined results
✅ **Guardrail**: Error handling with "unprocessed" status
✅ **Bonus**: AI-powered hallucination prevention

🤝 Contributing

1. Fork the repository
2. Create a feature branch: `git checkout -b feature-name`
3. Make changes and test thoroughly
4. Submit a pull request

📄 License

👨‍💻 Author

Your Name
SDE Intern Assignment - Medical Report Simplifier
Built with ❤️ using FastAPI and Google Gemini AI

📞 Support

For questions or issues:

- 📧 Email: your.email@example.com
- 🐛 Issues: GitHub Issues
- 📖 Documentation: API Docs

1. **Input Comparison**: AI compares original medical text with normalized results
2. **Semantic Analysis**: Detects context mismatches beyond simple string matching
3. **Confidence Scoring**: Uses confidence thresholds (>0.7) for reliability
4. **Medical Standardization**: Allows reasonable conversions (e.g., "Hgb" → "Hemoglobin")
5. **Fabrication Detection**: Identifies completely invented tests or wrong values

Example Scenarios

✅ **Valid**: Original: `"Hgb 10.2 g/dL Low"` → Normalized: `"Hemoglobin: 10.2 g/dL (Low)"`

❌ **Invalid**: Original: `"Hgb 10.2 g/dL Low"` → Normalized: `"Cholesterol: 220 mg/dL (High)"`

Testing the Validation

Run the demo script to see validation in action:

```
python demo_validation.py
```

Run validation-specific tests:

```
pytest tests/test_validation.py -v
```

Error Handling

The API includes comprehensive error handling:

- Invalid file formats
- OCR processing failures
- AI service errors
- Validation errors
- Hallucination detection with detailed error messages

Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add tests
5. Submit a pull request

PROF

License

MIT License