Medical Report Simplifier

An AI-powered backend service that processes medical reports (text or images) and provides patient-friendly explanations using OCR, text normalization, and Google Gemini AI.

Features

- **Multi-format Input**: Supports both text input and image uploads (PNG, JPG, PDF)
- **OCR Processing**: Extracts text from medical report images with confidence scoring
- **Text Normalization**: Standardizes medical test names, values, units, and reference ranges
- **AI-Powered Explanations**: Uses Google Gemini to generate patient-friendly summaries
- **AI-Powered Validation**: Prevents hallucinated results using semantic comparison
- **4-Step Pipeline**: OCR → Normalization → AI Explanation → Final Output

Tech Stack

- **FastAPI** - Modern web framework for APIs
- **Google Gemini API** - AI for text processing and explanations
- **Tesseract OCR** - Text extraction from images
- **Pydantic** - Data validation and serialization
- **Python 3.9+** - Core programming language

Setup Instructions

Prerequisites

1. **Python 3.9+** installed
2. **Tesseract OCR** installed:

```
# Ubuntu/Debian
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

Installation

PROF

1. **Clone the repository**:

```
git clone <your-repo-url>
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**:

```
cp .env.example .env
```

Edit `.env` and add your Google Gemini API key:

```
GEMINI_API_KEY=your_api_key_here
```

Running the Application

PROF

1. **Start the server**:

```
python3.9 -m uvicorn app.main:app --reload --port 8000
```

2. **Access the API**:
   - API Documentation: http://localhost:8000/docs
   - Health Check: http://localhost:8000/health

API Usage Examples

1. Process Text Input

```
```

```
curl -X POST "http://localhost:8000/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

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

Expected Response Format

```
{
  "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.",
  "status": "ok"
}
```

PROF

Architecture

```
app/
├── main.py                 # FastAPI application entry point
├── models/
│   └── schemas.py          # Pydantic models for request/response
├── services/
│   ├── ocr_service.py      # OCR and text extraction
│   ├── normalization_service.py  # Medical test normalization
```

```
│   ├── ai_service.py       # Gemini AI integration
│   └── medical_data.py     # Medical reference data
├── api/
│   └── endpoints.py        # API route handlers
└── core/
    ├── config.py           # Configuration management
    └── utils.py            # Utility functions
```

Testing

Run tests with:

```
pytest tests/ -v
```

Deployment

Local Demo with ngrok

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
   ```

PROF

Cloud Deployment

The application can be deployed to:

- **Railway**: Connect GitHub repo and deploy
- **Render**: Connect GitHub repo and deploy
- **Heroku**: Use Procfile for deployment

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: Semantic Context Comparison

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

```
def validate_against_hallucination(self, original_tests: List[str],
                                   normalized_tests: List[NormalizedTest])
-> Tuple[bool, str]:
    """
    Use AI to validate that normalized tests don't contain hallucinated
information
    by comparing semantic context between original and normalized data
    """
```

How It Works

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
```

PROF

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

License

MIT License