Healthcare Systems |

Detection of Cardiovascular Disease Using Machine Learning and Deep Learning

PULSEAI

Software Requirements Specification and System Design

TABLE OF CONTENTS

PULSEAI Introduction

Brief about Phase-1 Presentation

INTRODUCTION

- Cardiovascular diseases (CVDs) remain one of the leading causes of death worldwide, accounting for millions of fatalities each year.

- Early detection is vital to reducing mortality and improving patient outcomes. Among the most common diagnostic tools, the Electrocardiogram (ECG) plays a crucial role in monitoring and identifying heart conditions.

- With the advancement of Artificial Intelligence (AI), particularly Machine Learning (ML) and Deep Learning (DL), the automation of ECG analysis has become a viable solution.

The proposed system, **PULSEAI**, aims to leverage both ML and DL techniques to detect cardiovascular diseases from ECG images. Unlike many existing solutions that rely solely on raw signals, PULSEAI processes image-based ECG data, making it suitable for diverse clinical environments. The system's core contributions include:

- An **efficient CNN architecture** optimized for low computational cost, ensuring compatibility with low-resource devices.
- **Automated and robust classification** of multiple cardiovascular conditions, including normal rhythms, abnormal heartbeats, myocardial infarction, and history of myocardial infarction.
- A **comparative performance analysis** between ML classifiers (e.g., SVM, KNN) and DL models to identify optimal solutions.
- **Real-time processing capability** for practical and immediate diagnostic support.

Software Requirements Specification

SOFTWARE REQUIREMENTS

- Windows, macOS
- Python 3.9
- IDE:Visual Studio code
- Jupyter Notebook
- Streamlit(Python-based web UI) - Frontend
- MongoDB Atlas
- Chrome/Firefox
- Python-dotenv
- Google cloud
- Git and Docker Desktop

HARDWARE REQUIREMENTS

- **CPU**: Quad-core processor, Dual-core intel i3 minimum
- **RAM**: 8GB or more, 4GB minimum
- **Storage:** 10+ GB SSD

Software Requirements Specification

Functional Requirements

1. Upload an ECG image via UI.
2. Visualize intermediate steps
3. Convert per-lead signal to scaled 1D numeric series and combine.
4. Apply PCA for dimensionality reduction.
5. Predict class via pre-trained model and display result.
6. Export intermediate artifacts (images/CSVs) to local working directory.

Software Requirements Specification

Non-Functional Requirements

Usability

Single-page, step-by step feedback with expanders.

Performance

End-to-end prediction <10s for typical images.

Hardware Optimized

Using only the CPU, Without requiring specialized hardware..

Portability

Runs locally (venv) or in Docker; deployable to Cloud Run.

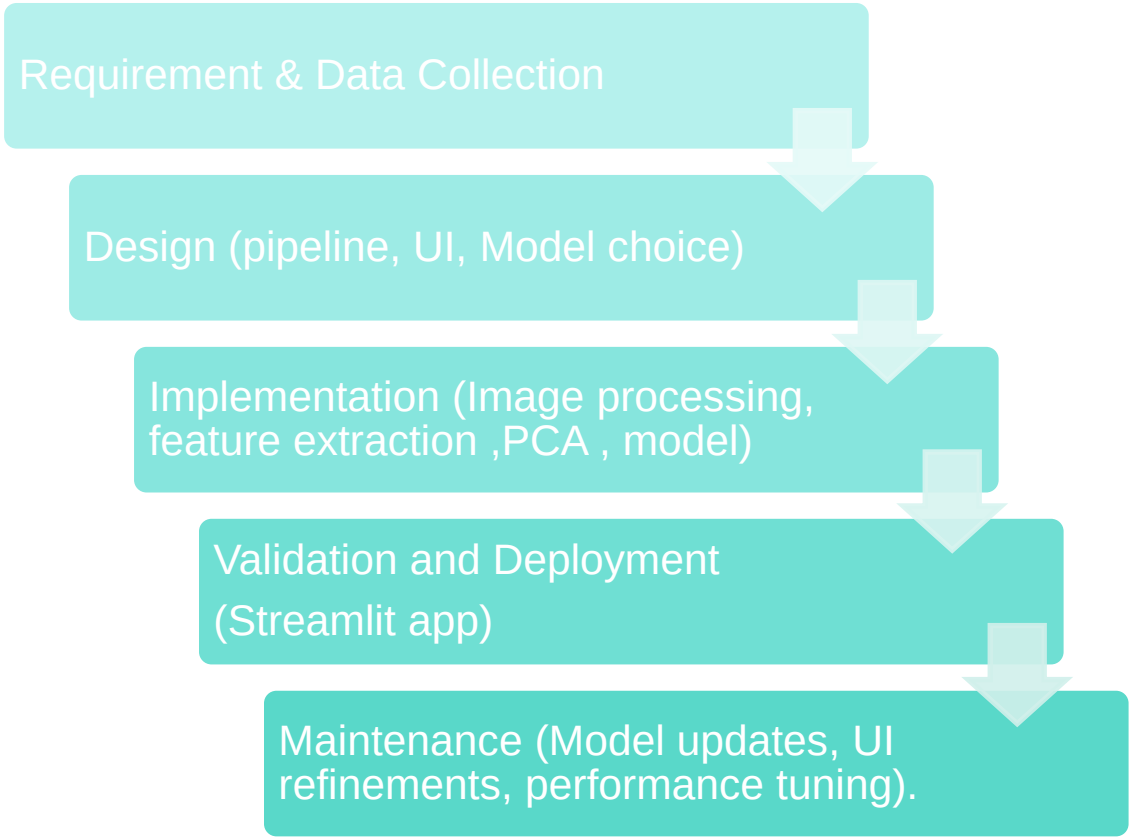
Maintainability

Clear modular steps in Ecg.py.

Observability

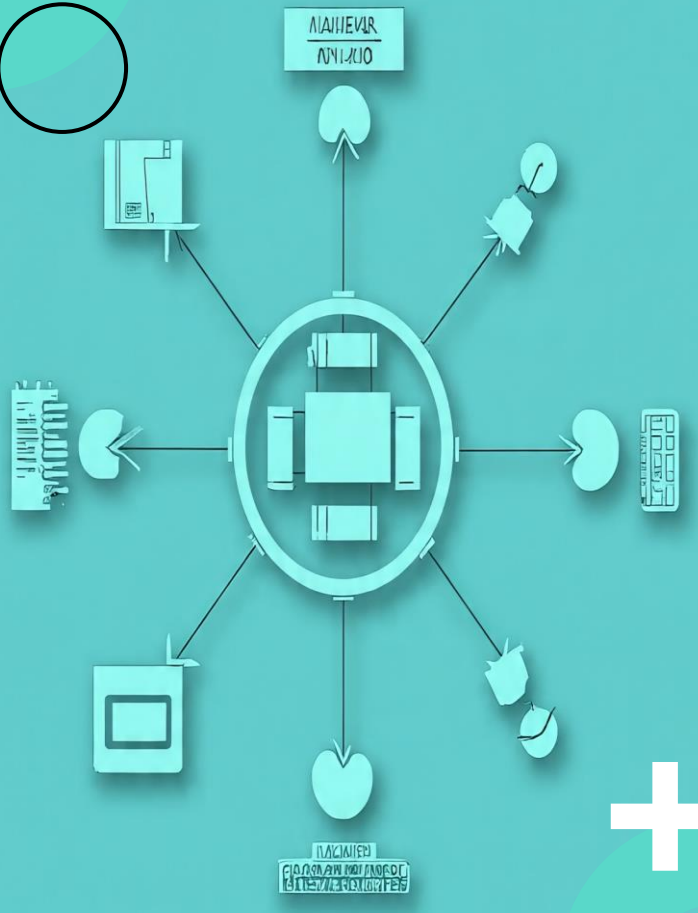
Save intermediate artifacts for debugging.

Project Development Life Cycle



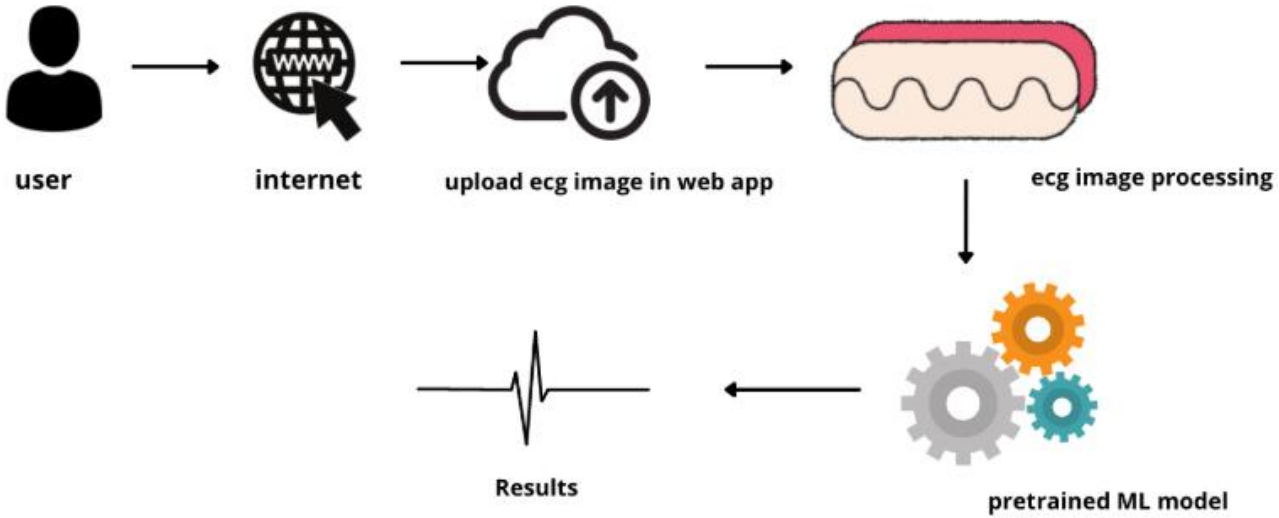
SCHEDULE

Task	Description	
Task 1	Planning	May 2025
Task 2	Design and Prototyping	June –July 2025
Task 3	Development	Aug – Sep 2025
Task 4	Testing	Oct 2025
Task 5	Deployment	Nov 2025



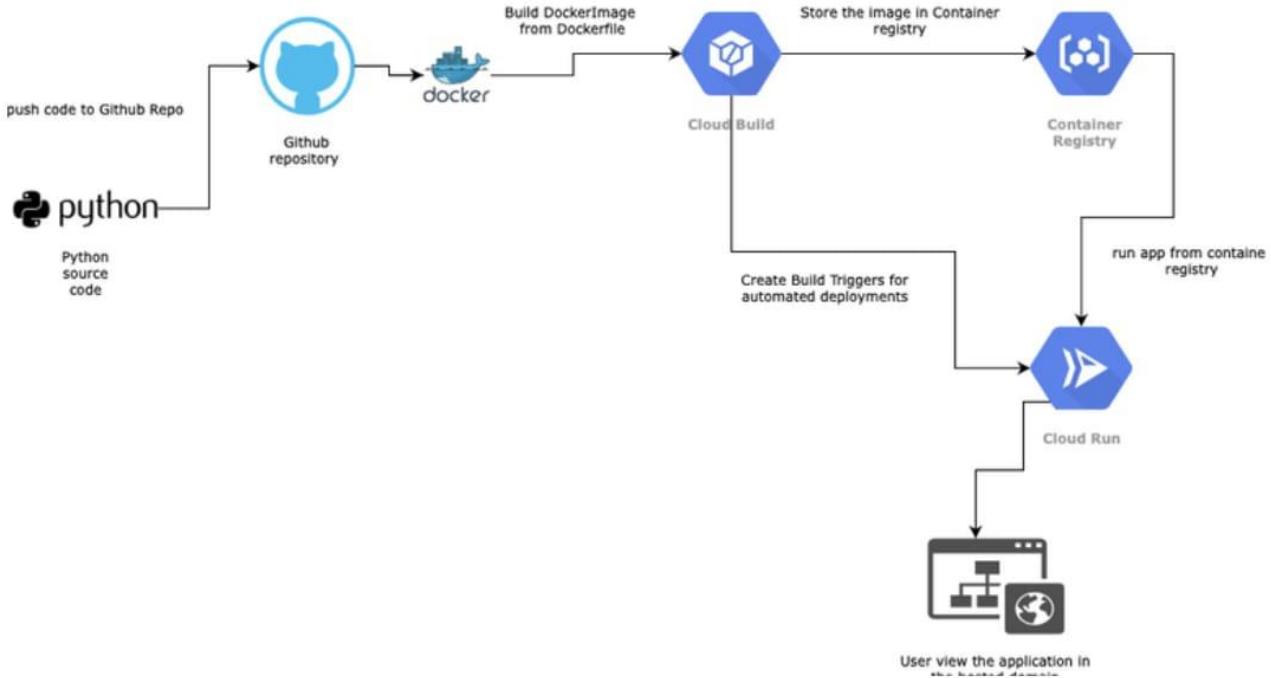
System Design

Project Architecture



Project Architecture

Our Deployment Architecture



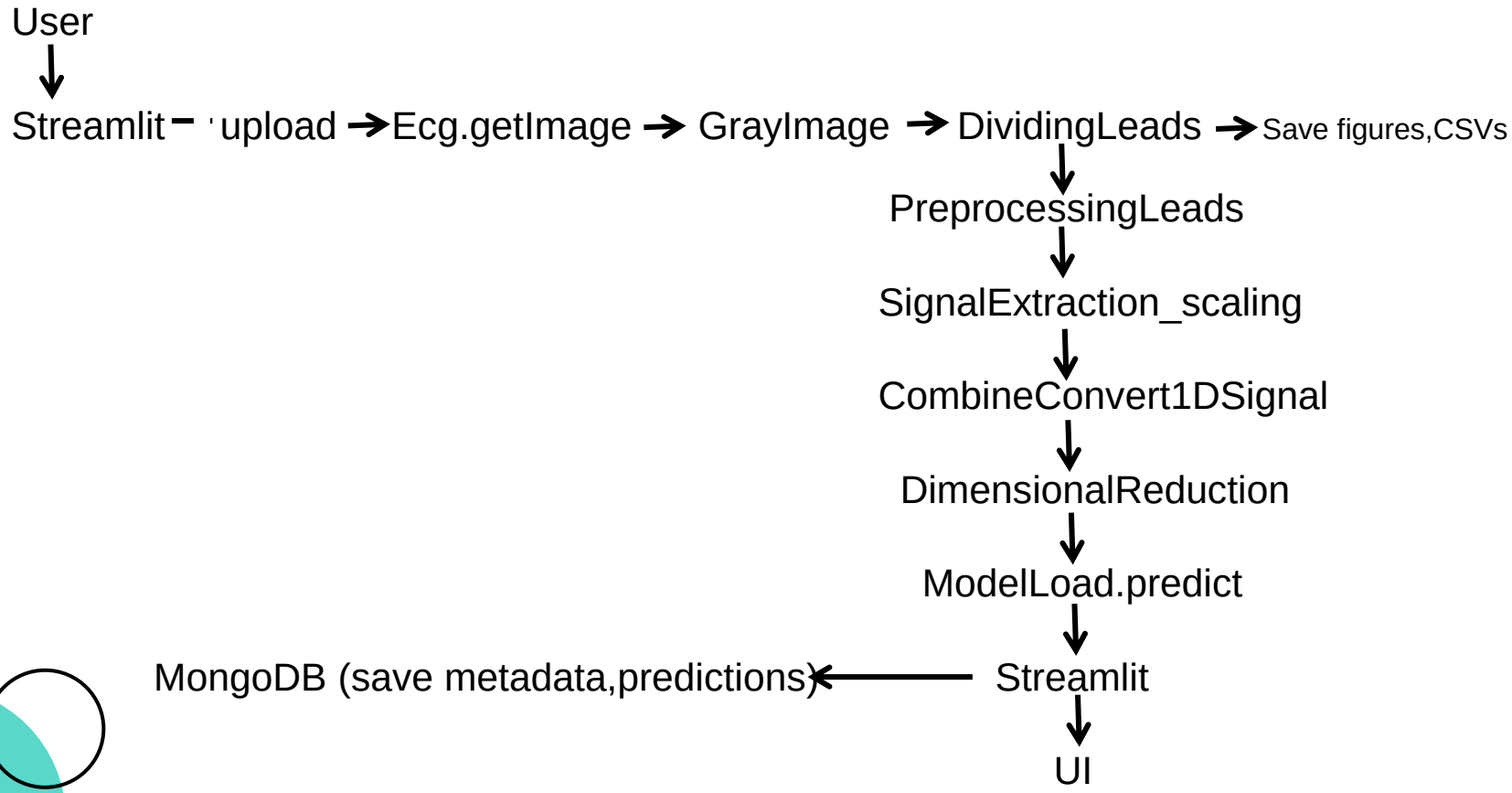
Project Architecture Description

Layer / Component	Description
Application Layer	Streamlit app (Deployment/final_app.py) serves as UI & controller; users upload ECG images via st.file_uploader; expanders display each pipeline stage.
Processing Layer	ECG pipeline in Deployment/Ecg.py with functions: getImage, GrayImage, DividingLeads, PreprocessingLeads, SignalExtraction_Scaling, CombineConvert1Dsignal, DimensionalReduciton, ModelLoad_predict.
Model Artifacts	Joblib pickles stored in Deployment/ and model_pkl/; loaded on demand; CPU-only scikit-learn models.
Intermediate Storage	Artifacts like *.png and Scaled_1DLead_*.csv saved in the working directory for debugging and traceability.
Frontend/UI	Built with Streamlit components (st.image, st.expander, st.write); no separate SPA or API—Python functions directly drive the interface.
Dependencies / Runtime	Python 3.9 recommended; uses NumPy, SciPy, scikit-image, scikit-learn with pinned versions; runnable in local venv or Docker container.
Deployment	Cloud Run container exposing port 8080; local dev on port 8501 with the same entrypoint.

Database Design

```
{
  "_id": ObjectId,
  "filename": "sample_ecg.png",
  "predicted_class": "MI",
  "timestamp": ISODate("2025-08-04T09:45:00Z")
}
```

Sequence Diagram



Dataflow Diagram

Context:

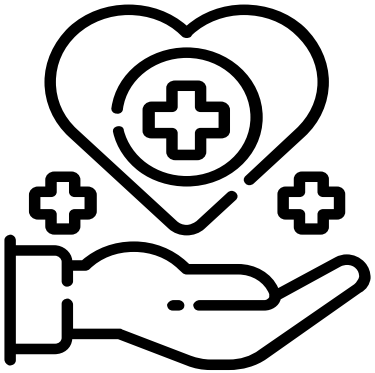
External Entity: User

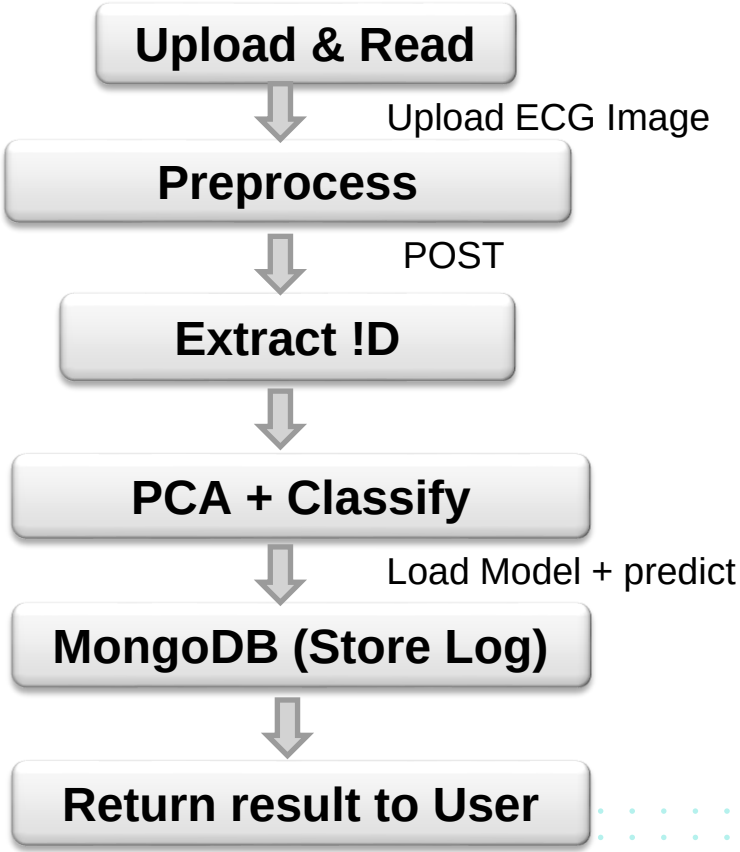
System: PulseAI Website

Database: MongoDB, Object Storage (images/artifacts)

Flows: Upload image, view results, stored artifacts/predictions

Dataflow Diagram

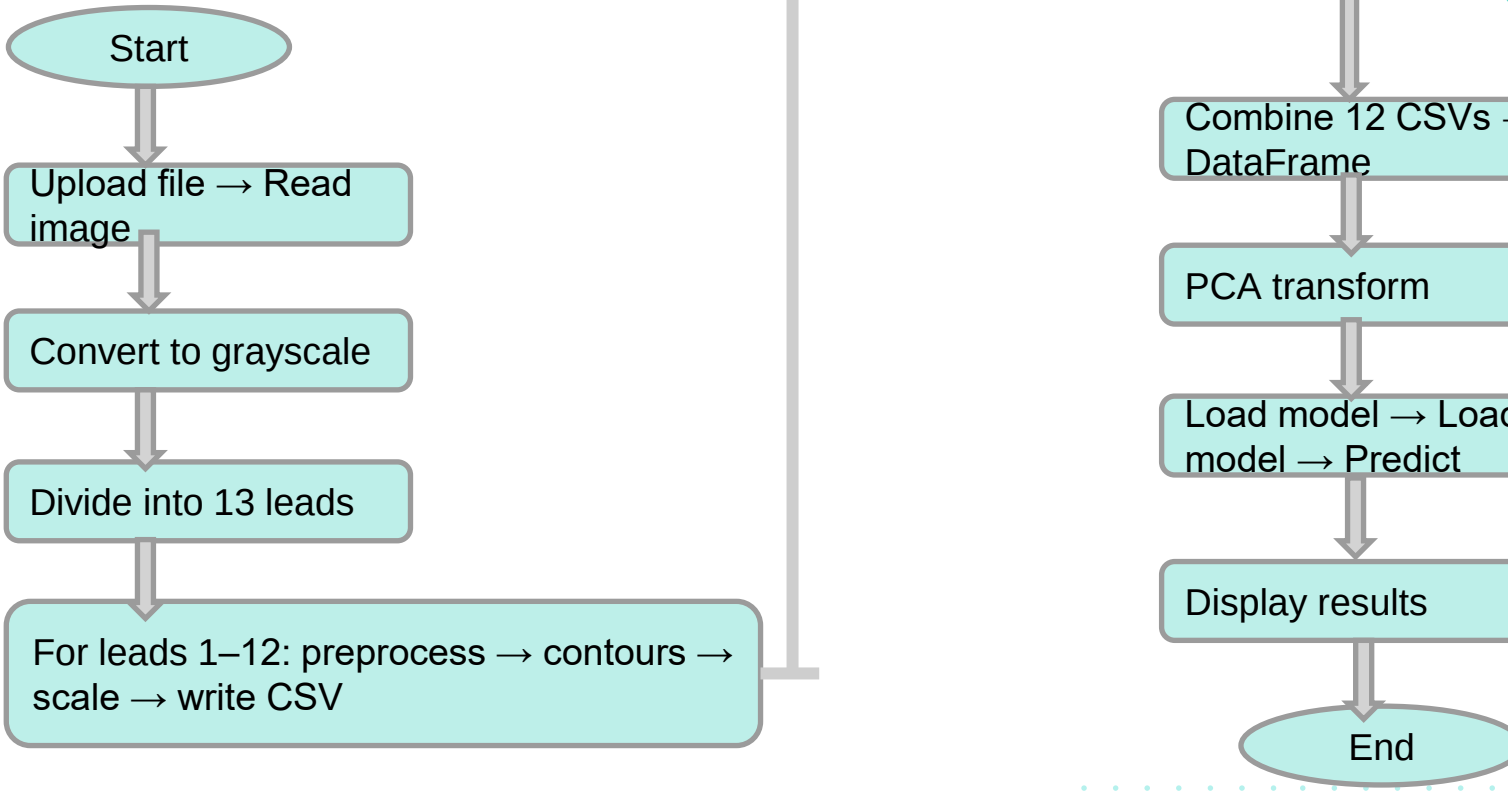




Database Table Structure

Field Name	Type	Description
_id	ObjectId	MongoDB's internal primary key
filename	String	Name of uploaded ECG image
predicted_class	String	One of: Normal, MI, AH, HMI
timestamp	DateTime	When the prediction was made

Flowchart


Start
Upload file → Read image
Convert to grayscale
Divide into 13 leads
For leads 1–12: preprocess → contours → scale → write CSV
Combine 12 CSVs → DataFrame
PCA transform
Load model → Load model → Predict
Display results
End

THANK YOU