Milestone-4 Report

Flask-Based Web Application for PCB Defect Classification

Duration: Week 7 – Week 8

---

1. Milestone Title & Duration

Milestone-4 focuses on **upgrading the existing Streamlit-based prototype into a production-oriented Flask web application**. This milestone emphasizes better control over routing, scalability, deployment flexibility, and structured backend handling while retaining all functional features implemented in Milestone-3.

The Flask-based application provides a more robust architecture suitable for real-world deployment scenarios, API extensions, and integration with external systems.

---

2. Objective of Milestone-4

The key objectives of this milestone were:

- To migrate the Streamlit interface to a **Flask-based web application**.
- To design a **modular backend architecture** separating frontend views, inference logic, and file handling.
- To enable **multi-image upload and batch processing** using HTTP-based form submissions.
- To allow users to:
  - Download **individual annotated images**
  - Download **annotation files in YOLO format** (`.txt`)
  - Download a **ZIP archive** containing all processed images and annotations
- To prepare the application for **scalable deployment** on cloud or on-premise servers.

---

3. Tasks Completed

During Milestone-4, the following tasks were completed:

- Designed a Flask application structure with separate folders for templates, static files, uploads, and outputs.
- Converted Streamlit-based logic into Flask route handlers.

- Implemented routes for:
    - Image upload
    - Batch inference execution
    - Individual result visualization
    - File and ZIP downloads
- Integrated the trained YOLO model into the Flask backend with **single-time model loading** to improve efficiency.
- Implemented annotation export in **YOLO text format** alongside annotated images.
- Added backend validation for supported file types and file size limits.
- Tested the application for batch uploads and repeated inference cycles.

---

4. Technologies Used

Category	Tools / Frameworks
Web Framework	Flask
Frontend	HTML, CSS, JS
Backend Model	YOLOv8 (Ultralytics, PyTorch)
Image Processing	OpenCV
File Handling	Python, Zipfile, OS, Tempfile
Deployment Ready	Local Flask Server

Flask was chosen due to its lightweight nature, flexibility, and suitability for production deployment.

---

5. System Workflow & Implementation Summary

Step 1: User Uploads Images

- Users upload one or more PCB images through an HTML form.
- Uploaded files are validated and saved temporarily on the server.

Step 2: Model Inference

- The YOLO model processes each image sequentially.
- For each image:
    - Defect regions are localized
    - Defect classes are predicted
    - Bounding boxes and confidence scores are generated

Step 3: Annotation Generation

- Annotated images are saved with bounding boxes and labels.
- Corresponding annotation files are generated in **YOLO format**, containing:
  - Class ID
  - Normalized bounding box coordinates

Step 4: Result Visualization

- Processed images are displayed on result pages using Flask templates.
- Users can inspect results individually.

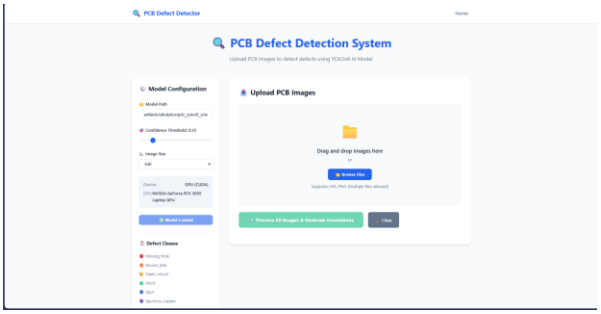
Step 5: Download Options

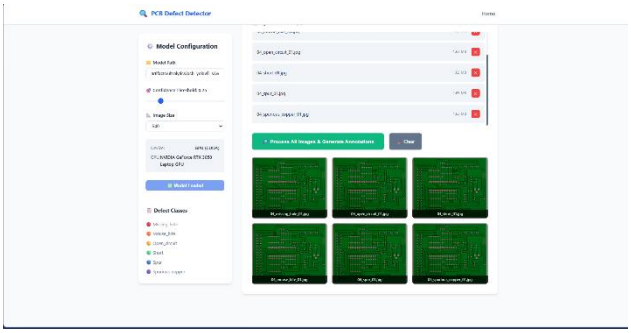
- Individual download options:
  - Annotated image
  - YOLO annotation `.txt` file
- Batch download option:
  - ZIP file containing all annotated images and annotation files

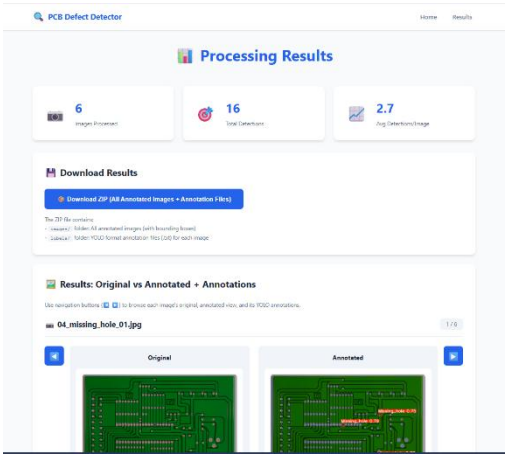
---

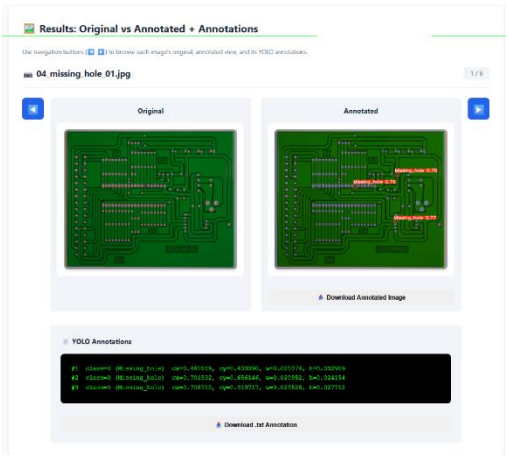
6. Model Outputs (Flask Interface)

The Flask interface provides a cleaner separation between UI and backend logic, enabling easier customization and future expansion.









7. Challenges & How I Solved Them

Challenge	Observation	Solution
Migration from Streamlit to Flask	Streamlit abstractions not directly reusable	Rewrote logic using Flask routes and templates
YOLO annotation export	Bounding boxes needed normalization	Converted absolute coordinates to YOLO normalized format
ZIP generation reliability	Large batch outputs caused memory issues	Used temporary directories and streamed ZIP creation

These solutions ensured stability, correctness, and scalability of the application.

Summary of Milestone-4

Milestone-4 successfully upgraded the project from a prototype-level Streamlit interface to a **production-ready Flask web application**. The system now supports:

- Multi-image upload and processing
- Accurate PCB defect classification
- Individual and batch downloads
- YOLO-format annotation exports

This milestone significantly improves deployment readiness and makes the application suitable for industrial inspection pipelines, cloud hosting, and API-based extensions.

With the completion of Milestone-4, the project now covers the full lifecycle, dataset preparation, model training, interactive UI, and deployable web application.