PCB DEFECT DETECTION PROJECT

Name :Namile Sharanya Gajanan

Model: yolov8n

Milestone 1: Image Processing

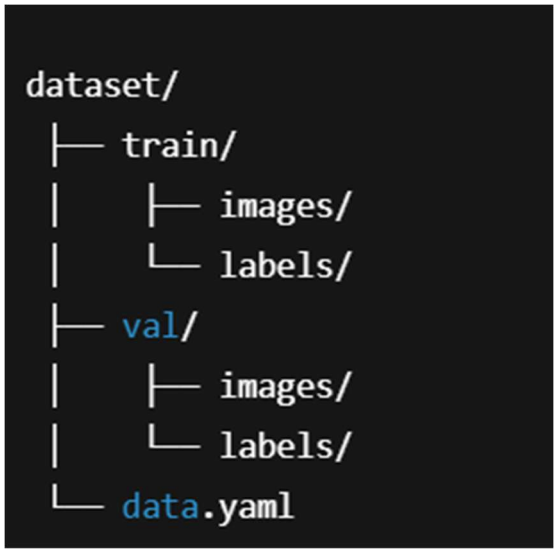
Objective

The primary objective of this milestone was to prepare a high-quality dataset of PCB images with clear and accurate defect annotations. This ensures that the model can correctly learn and detect all key defect regions.

Dataset Preparation

The PCB dataset consists of real-world printed circuit board images containing common manufacturing defects. The dataset was structured following the YOLO format.

- Images and labels were organized into training and validation sets.
- All file formats were standardized (.jpg / .png).
- Proper directory hierarchy was maintained for seamless training.



Annotation Strategy

Each PCB image was annotated using bounding boxes representing visible defects. The annotations follow the YOLO format with normalized coordinates.

- Class ID mapping for six PCB defect categories.
- Normalized bounding box coordinates for consistency.
- Manual verification to minimize annotation errors.

Data Quality Assurance

- Checked for missing or corrupt images.
- Ensured every image had a corresponding label file.
- Visual inspection of bounding boxes to verify coverage.

Outcome: A clean, reliable, and well-annotated dataset ready for deep learning model training.

Milestone 2: Model Performance

Objective

This milestone focused on training an efficient and accurate deep learning model to automatically detect PCB manufacturing defects using CPU-only resources.

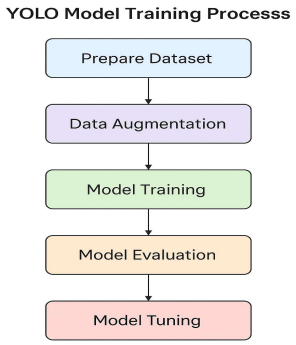
Model Selection and Justification

YOLOv8n was selected due to its lightweight architecture, fast inference speed, and ability to perform well even on limited hardware.

- Pretrained weights used for faster convergence.
- Optimized for real-time detection scenarios.
- Balanced trade-off between speed and accuracy.

Training Process

- Images resized to 640×640 for uniform input.
- Multiple epochs used to allow stable learning.
- Validation set used to monitor overfitting.



Fine-Tuning Strategy

Instead of retraining the model from scratch, fine-tuning was performed using the best checkpoint obtained during the initial training phase.

- Improved detection for complex defect patterns.
- Reduced training time.
- Enhanced model generalization.

```
Your best.pt (already PCB-aware)
↓
More training on same PCB data
↓
Fix leftovers (missed spur, confused classes, sharp boxes)
```

Evaluation Metrics

- Precision – evaluates detection correctness.
- Recall – measures model completeness.
- mAP@0.5 and mAP@0.5:0.95 – overall detection accuracy.

Outcome: A well-performing PCB defect detection model capable of identifying multiple defect types with strong accuracy.

Milestone 3: UI Integration

Objective

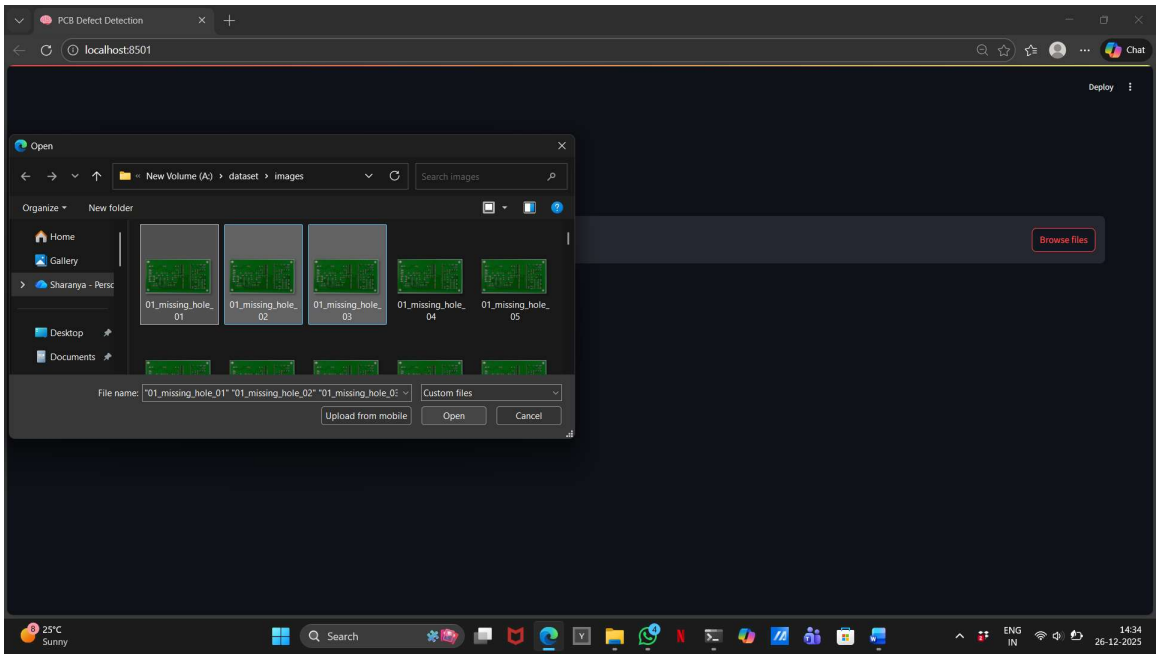
This milestone aimed to integrate the trained model into a professional and interactive user interface for real-world usability.

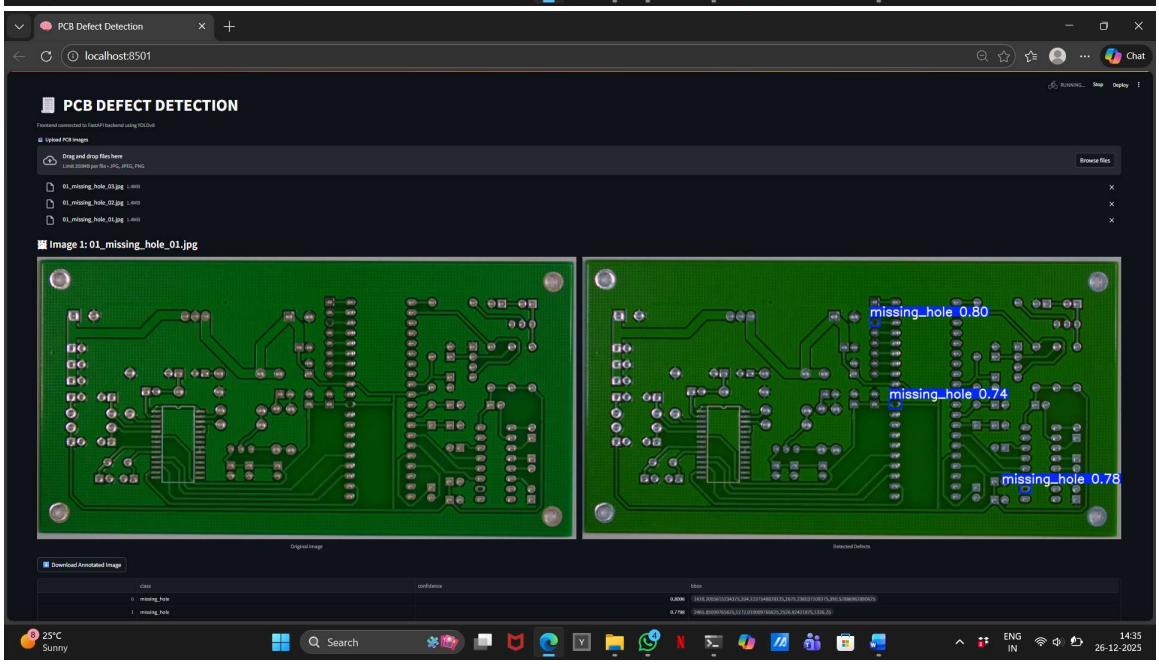
Technology Stack

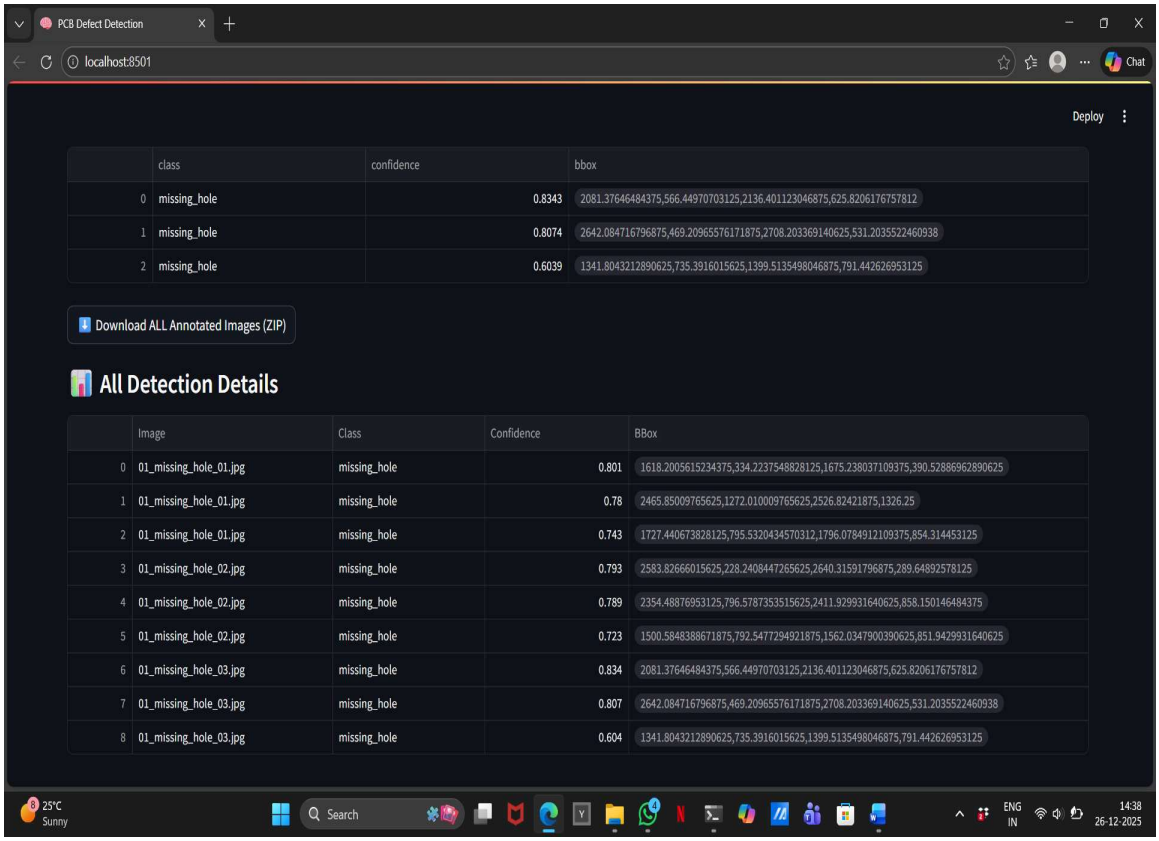
- Frontend: Streamlit
- Backend: Python
- Detection Engine: YOLOv8
- Data Handling: NumPy and Pandas.

Interface Features

- Multiple image upload capability.
- Side-by-side visualization of original and detected images.
- Adjustable confidence and IoU thresholds.
- Per-image and combined result tables.

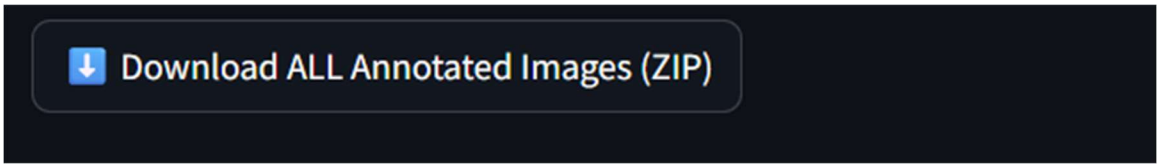






Download and Reporting Features

- Individual annotated image download.
- Bulk download of all results in ZIP format.
- Clear detection summary for analysis.



Performance Optimization

Model caching and optimized preprocessing ensure fast response times, meeting the requirement of under 5 seconds per image on CPU.

Outcome: A professional, user-friendly PCB defect detection application ready for demonstration and deployment.

Training & Validation Loss Curves

- **Box Loss**
  - **What it shows:** How accurately the model predicts bounding box locations.
  - **Expected behavior:**
    - Decreases steadily → model is learning correct object localization.

Class Loss

- **What it shows:** How well the model classifies defects (missing_hole, mouse_bite, etc.).
- **Expected behavior:**
  - Should decrease over epochs

  .

Precision Curve

- **What is Precision?**

Out of all predicted defects, how many are **actually correct**?

- **High precision = fewer false positives**
- **Your curve:**
  Precision increased close to **90%+**, meaning the model rarely predicts wrong defects.

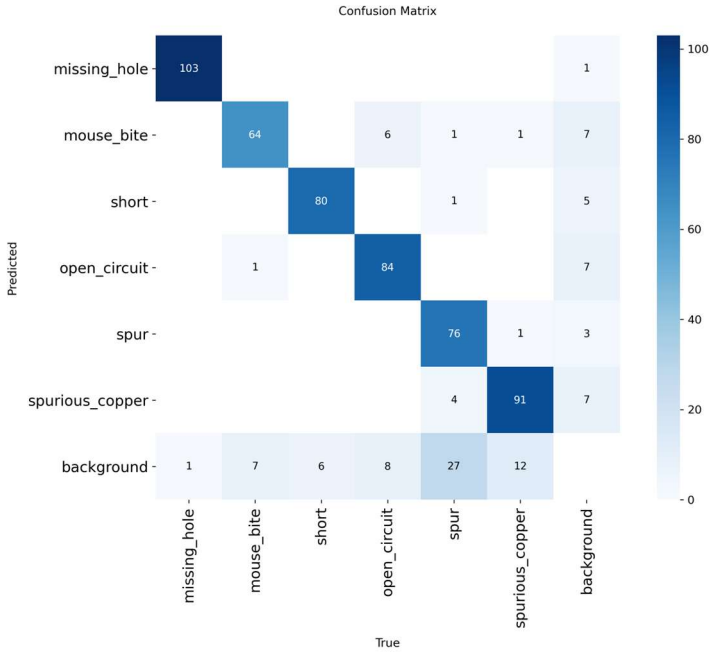
Recall Curve

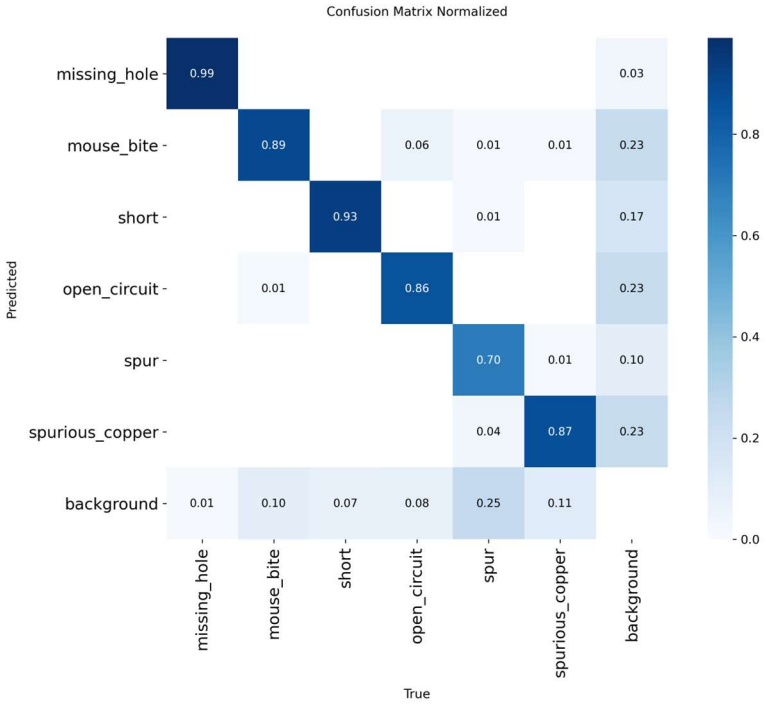
- **What is Recall?**

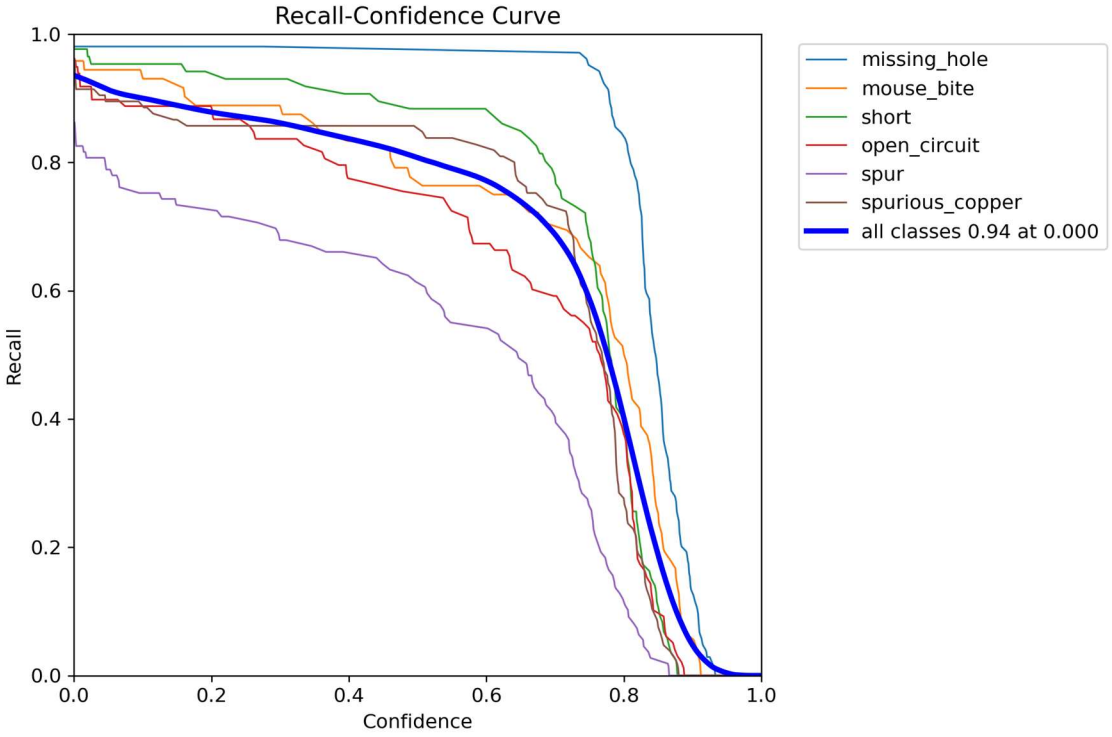
Out of all real defects, how many did the model **successfully detect**?

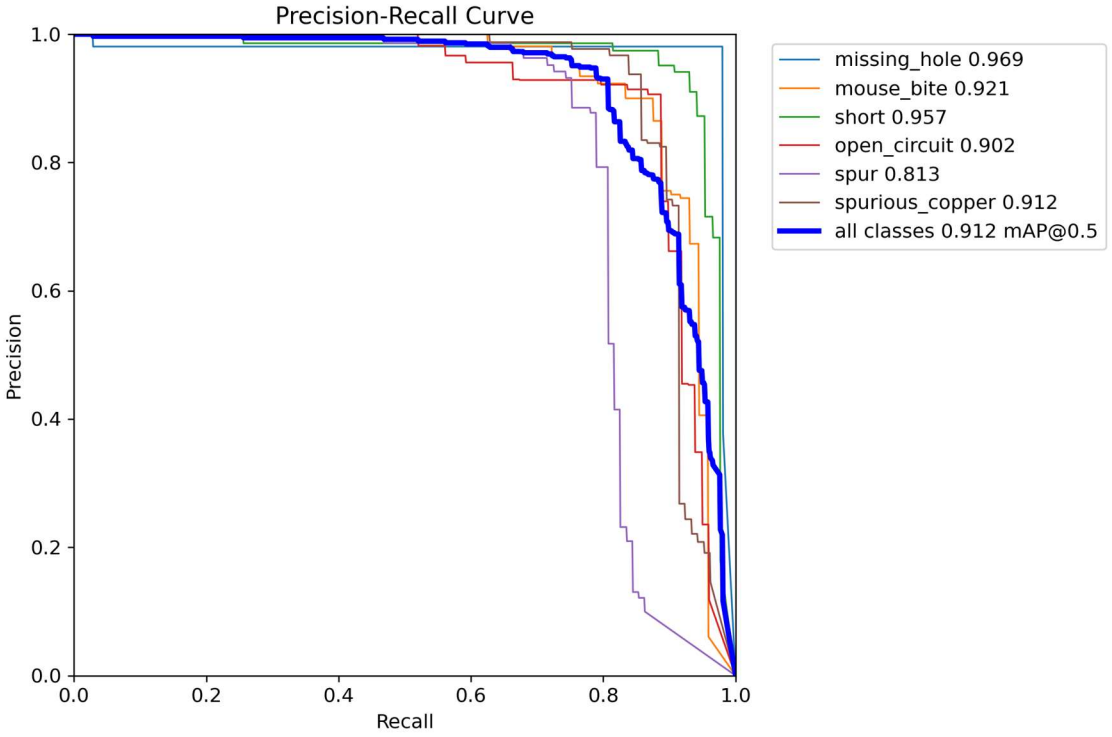
- **High recall = fewer missed defects**

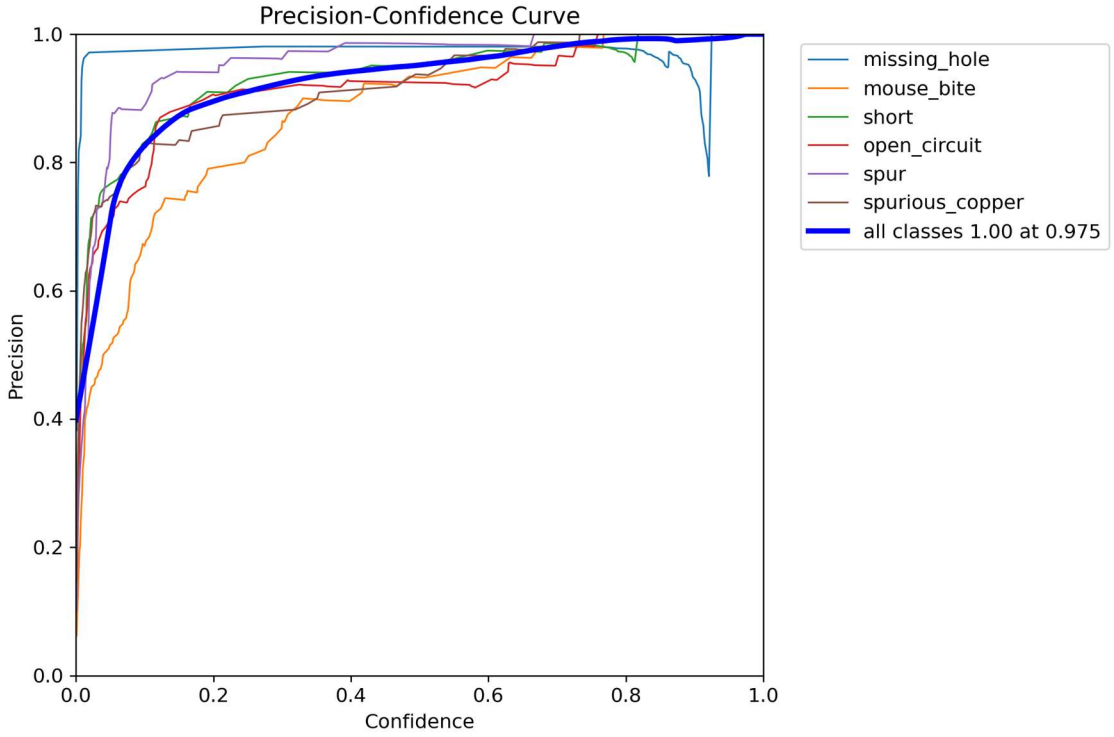
Your curve:
Recall steadily improved, showing the model detects most defects present on the PCB.

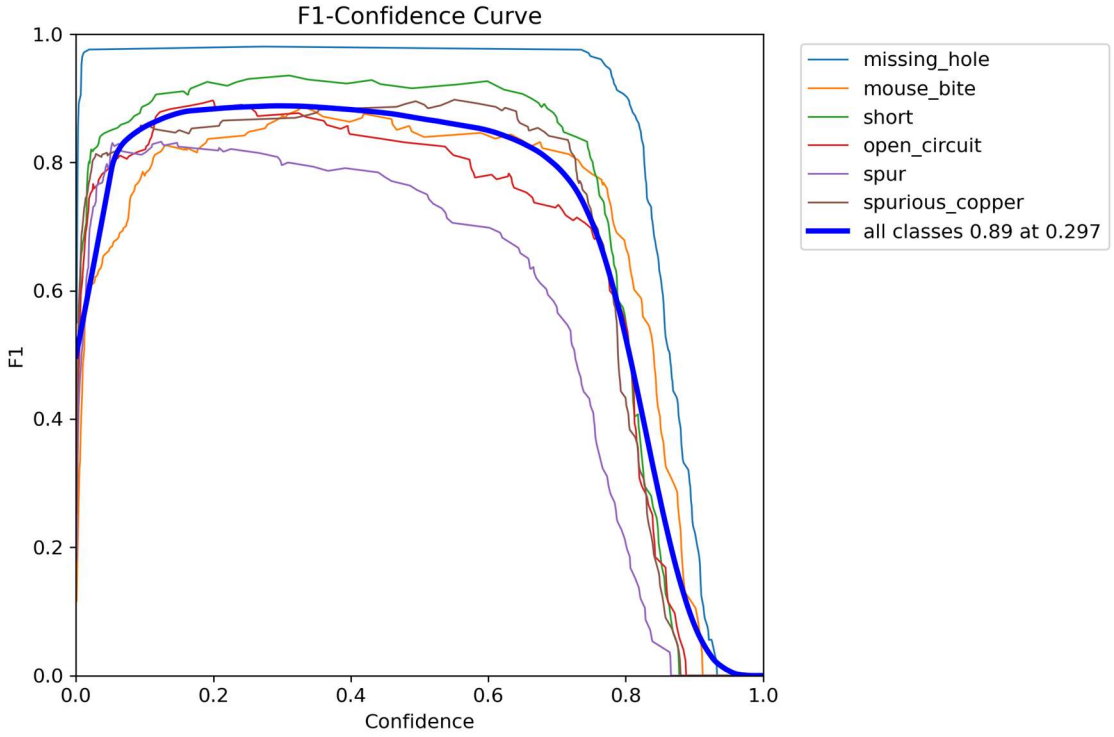





Recall-Confidence Curve
Recall
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
missing_hole
mouse_bite
short
open_circuit
spur
spurious_copper
all classes 0.94 at 0.000


Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
missing_hole 0.969
mouse_bite 0.921
short 0.957
open_circuit 0.902
spur 0.813
spurious_copper 0.912
all classes 0.912 mAP@0.5


Precision-Confidence Curve
Precision
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
missing_hole
mouse_bite
short
open_circuit
spur
spurious_copper
all classes 1.00 at 0.975


F1-Confidence Curve
F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
missing_hole
mouse_bite
short
open_circuit
spur
spurious_copper
all classes 0.89 at 0.297

Frontend Development

- **Purpose of Frontend**

The frontend provides a **user-friendly interface** for:

- Uploading PCB images
- Viewing defect detections
- Downloading annotated images
- Visualizing results

- **Technology Used**

- **Streamlit** – Rapid UI development
- **HTML + CSS (custom styling)** – Professional appearance
- **Requests library** – Communicates with FastAPI backend

Frontend–Backend Interaction

- Streamlit sends image to FastAPI using HTTP POST
- Backend returns detection results
- Frontend displays and formats the output

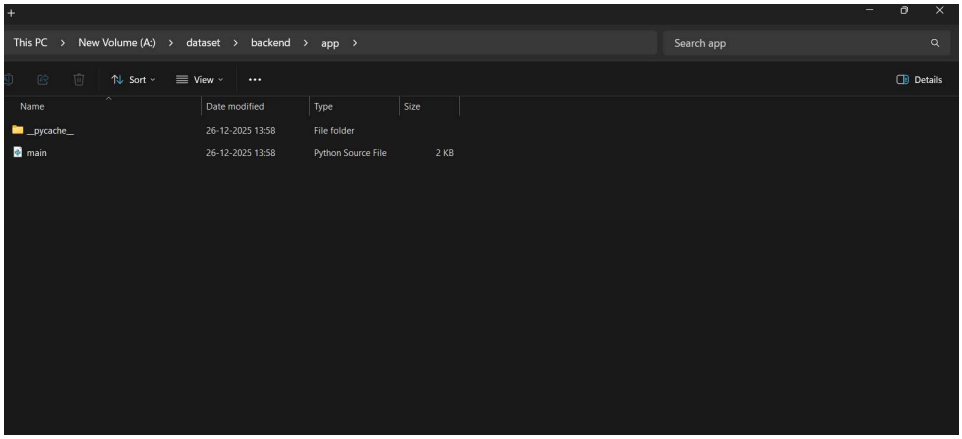
- **Why Streamlit?**

- Fast development
- Python-based
- Easy ML integration
- Ideal for academic & industrial demos

Backend Implementation

The backend of the **PCB Defect Detection System** is responsible for handling image uploads, running defect detection using a trained YOLOv8 model, and returning detection results to the user.
To achieve this, a **FastAPI-based backend** was developed due to its high performance, simplicity, and built-in API documentation support.

Backend Folder Structure



1. Purpose of the Backend

The backend of the PCB Defect Detection system is responsible for running the trained YOLOv8 model, processing uploaded PCB images, and returning detection results to the frontend in a structured manner.

The backend acts as a bridge between the AI model and the user interface, ensuring clean separation of logic and scalability.

---

2. Technology Stack Used

The backend is built using the following technologies:

- FastAPI – for building high-performance REST APIs

- YOLOv8 (Ultralytics) – for PCB defect detection
- Python – core programming language
- Uvicorn – ASGI server to run the FastAPI app
- Pillow (PIL) – image processing
- NumPy – array handling

---

3. Why FastAPI?

FastAPI was chosen because:

- It is very fast and lightweight
- Automatic Swagger UI documentation (/docs)
- Built-in support for file uploads
- Easy integration with ML models
- Clean separation of frontend and backend

4. Error Handling in Backend

The backend handles:

- Invalid image uploads
- Missing files
- Model inference errors

Returns proper HTTP status codes like:

- 200 OK – success
- 400 Bad Request – invalid input
- 500 Internal Server Error – processing failure