CIRCUITGUARD – PCB DEFECT DETECTION SYSTEM

Report of Milestone 1, 2, 3 & 4

By: Prachi Khatri

---

1. Introduction

Printed Circuit Boards (PCBs) play a critical role in the functioning of electronic devices, and even the smallest defects can lead to operational failure, reduced lifespan, or complete malfunction. Manual PCB inspection is slow, labor-intensive, expensive, and prone to errors. CircuitGuard addresses this by automating PCB defect detection using deep learning.

This project uses the **YOLO11m** model to detect six types of PCB defects:

- Missing Hole
- Mouse Bite
- Open Circuit
- Short Circuit
- Spur
- Spurious Copper

Across three milestones, the system evolved from dataset preparation and model training to a fully deployed web application with automated reporting capabilities. The final model demonstrates strong performance and robust generalization, making CircuitGuard suitable for real-world PCB quality inspection.

---

2. Dataset Setup

A labeled PCB defect dataset provided by the mentor was used for training. It contained:

- High-resolution PCB images
- Bounding-box annotations in Pascal VOC (XML) format
- Six defect categories, representing both structural and surface-related PCB defects

This dataset enabled precise supervised training of the detection model.

---

3. Data Understanding and Exploration

Before training, the dataset was studied to evaluate its preparation needs.

★ PCB Layout Variation

The dataset consisted of PCB images with **different board layouts**. Track structures, hole placements, and copper patterns varied between images. This required the model to generalize defect detection across different circuit designs.

★ Annotation Quality

Annotations in XML were properly aligned with defects but required careful conversion to YOLO's normalized bounding-box format.

★ Consistent Imaging Conditions

The dataset did **not** show major variability in lighting, angle, resolution, or backgrounds.
Images were captured under consistent conditions, which simplifies initial training but can reduce generalization if augmentation is disabled.

These insights influenced the preprocessing and training decisions in later stages.

---

4. Data Preparation

4.1 Annotation Conversion

Annotations were converted from VOC XML → YOLO format:

<class_id> <x_center> <y_center> <width> <height>

Coordinates were normalized relative to image size.
A custom converter script was created to ensure accuracy.

4.2 Train–Validation Split

The dataset was divided into:

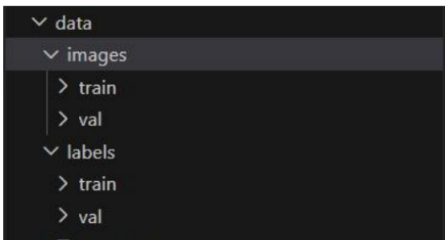
- **80% training**
- **20% validation/testing**

This split ensured sufficient training data while preserving evaluation integrity.

4.3 Directory Structure

Data was arranged in YOLO's expected structure:

/images/train
/images/val
/labels/train
/labels/val
Data.yaml



---

5. Model Selection

Multiple YOLO architectures were considered, including YOLOv8 and the YOLO11 family.
YOLO11m was selected because it offers:

- High accuracy
- Strong small-object detection
- Reasonable computational requirements
- Faster training and inference

Its performance on fine-grained PCB defects made it an ideal choice.

---

6. Training Strategy

Training occurred in **two phases**:

6.1 Phase 1 – Base Training

The YOLO11m model was trained without freezing layers.
This allowed the model to learn general defect patterns quickly.

6.2 Phase 2 – Fine-Tuning

To enhance detection of small defects:

- Initial backbone layers were **frozen(20 layers)** to retain pretrained visual features
- Later, the entire model was **unfrozen** for full fine-tuning

This two-step approach improved model stability and recall.

6.3 Data Augmentation

During model training, the dataset was processed using a **deterministic preprocessing pipeline**. Images were resized to a standard resolution of **1024×1024**, normalized, and formatted for YOLO input. No additional stochastic augmentations such as mosaic, mixup, rotations, or random brightness adjustments were applied in this training cycle.

This approach ensured consistent, reproducible training results and allowed the model to learn defect characteristics directly from the original dataset. Although the absence of augmentation may limit exposure to varied imaging conditions, the model compensated effectively through architectural strength and fine-tuning strategies, achieving high accuracy on the validation set.

6.4 Model Weights Saving

The trained weights were stored at:

runs/detect/train10/weights/best.pt
runs/detect/train10/weights/last.pt

best.pt was selected for inference and deployment.

---

7. Model Evaluation

The model performed strongly on the validation set.

7.1 Final Performance Metrics

Metric	Score
mAP@50	0.9823
mAP@50–95	0.5598
Precision	0.9714
Recall	0.9765

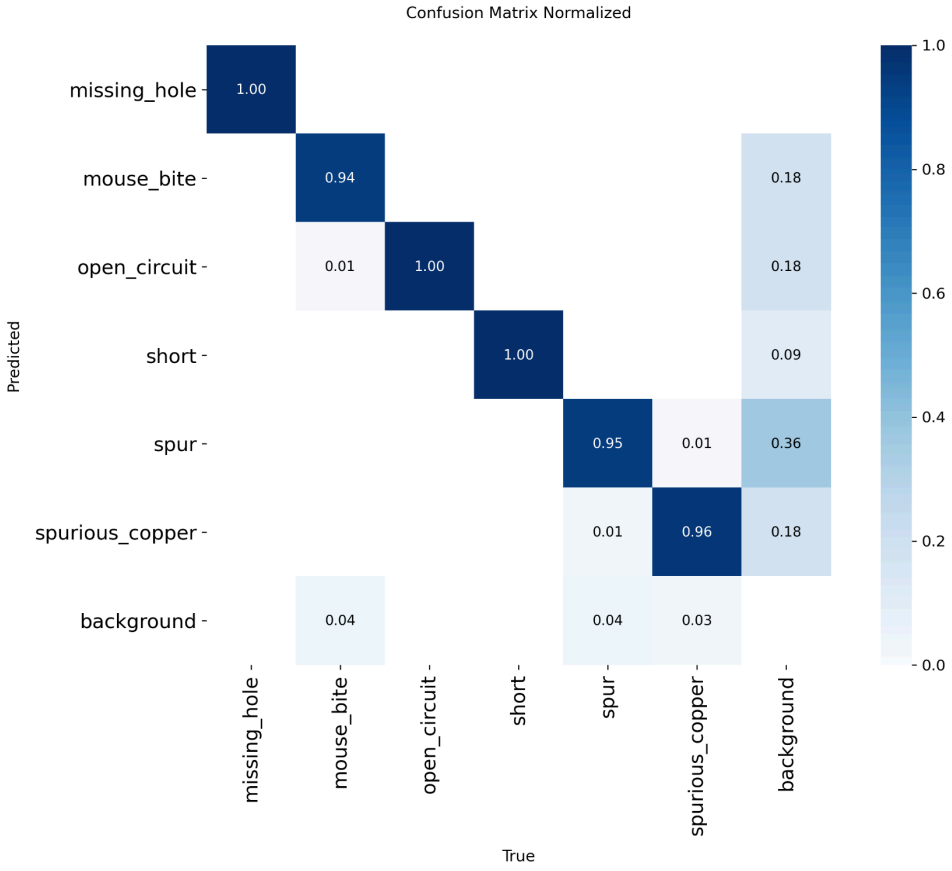
These values indicate excellent detection performance across classes.

7.2 Class-Wise Metrics

Class	Images	Instances	Precision	Recall	mAP50	mAP(50-95)
All	139	577	0.968	0.975	0.982	0.558
missing_hole	33	137	0.995	1	0.995	0.655
mouse_bite	21	89	0.965	0.941	0.962	0.521
open_circuit	23	88	0.953	0.989	0.991	0.573
short	19	81	0.979	1	0.995	0.545
spur	20	82	0.952	0.968	0.972	0.503
spurious_copper	23	100	0.96	0.95	0.977	0.55

7.3 Confusion Matrix

Shows how well the model distinguishes between the six defect categories.
Most predictions fall along the diagonal, indicating strong class-wise performance.



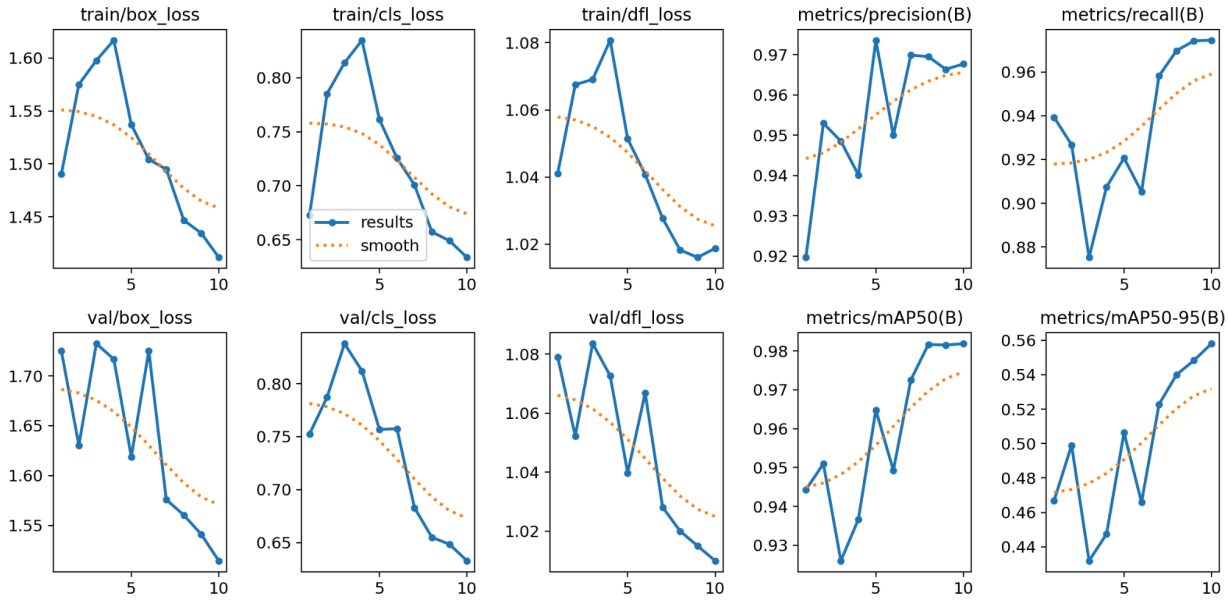
7.4 Training Graphs

1. Overall Training Summary

This combined plot displays the major performance trends including:

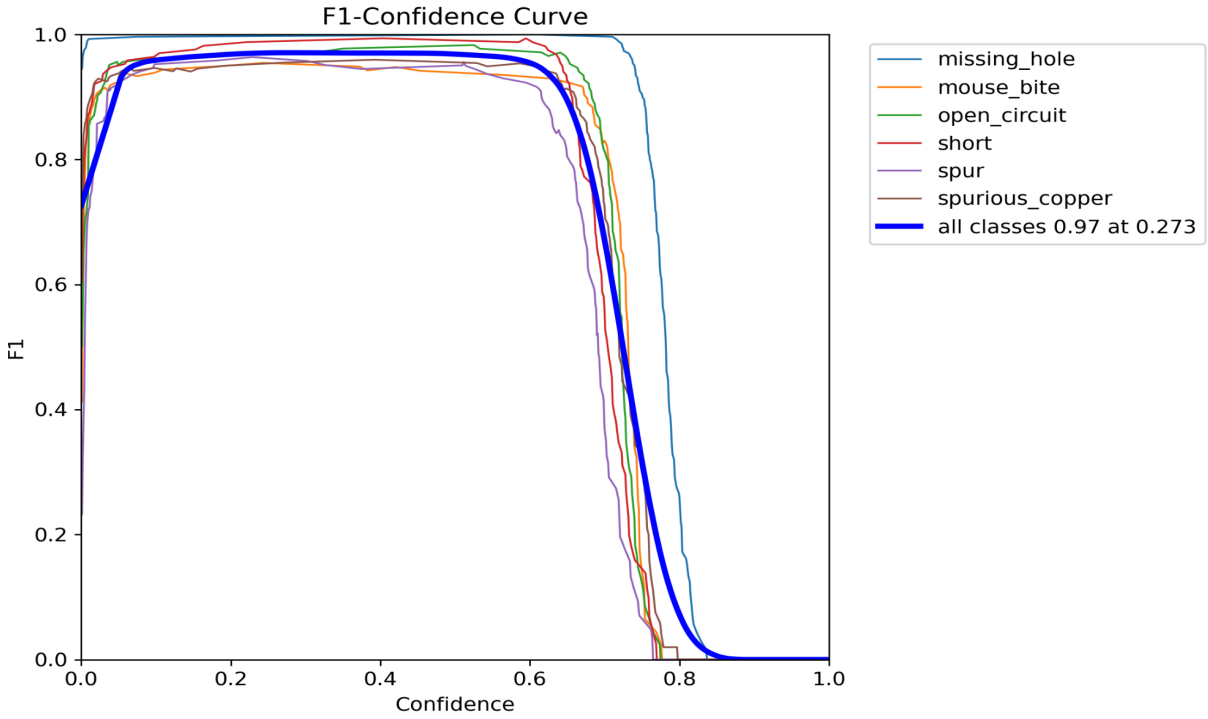
- Training and validation losses
- Precision and recall progression
- mAP@50 and mAP@50–95 improvements

It offers a complete overview of how the model converged during training.



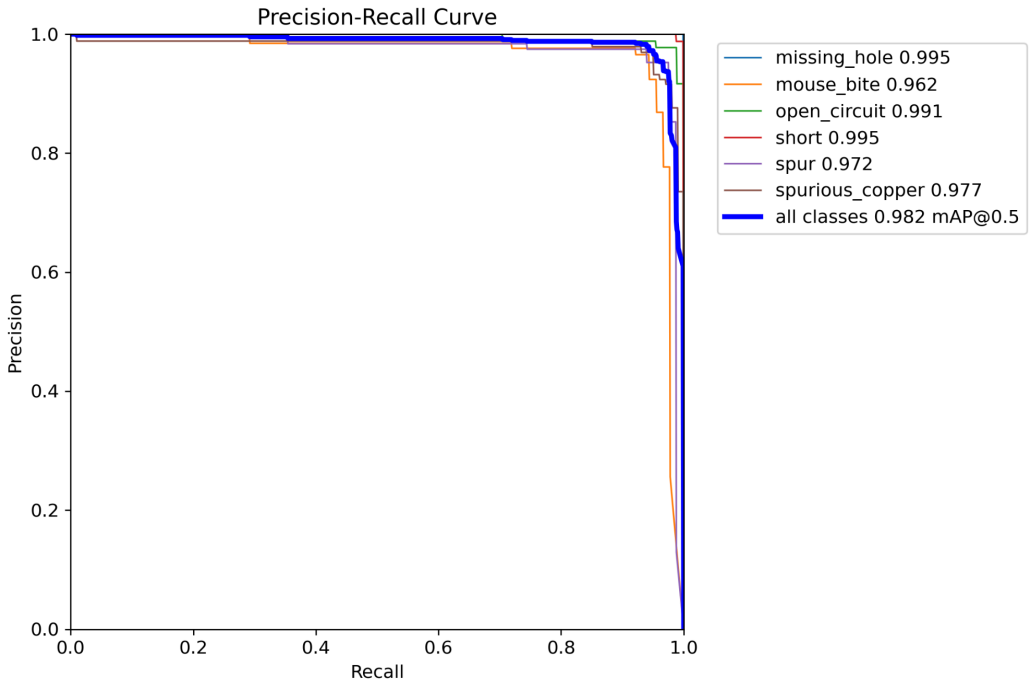
2. F1 Curve

The F1 curve shows the balance between precision and recall across confidence thresholds.
A strong F1 peak indicates the model maintains high accuracy while minimizing missed defects.



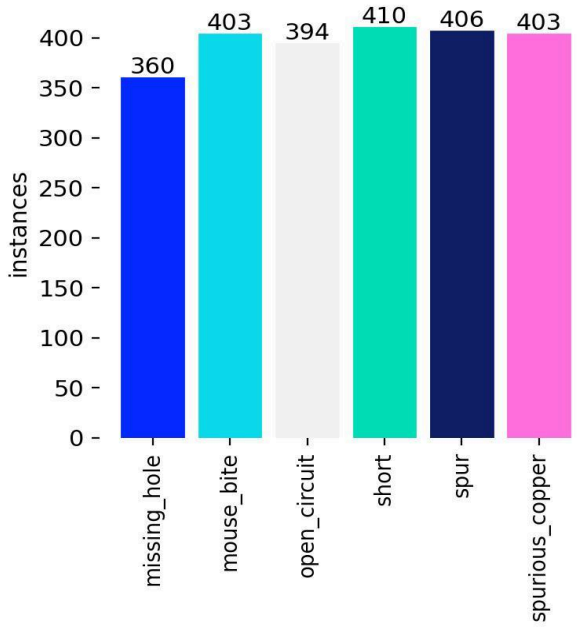
3. Precision–Recall Curve

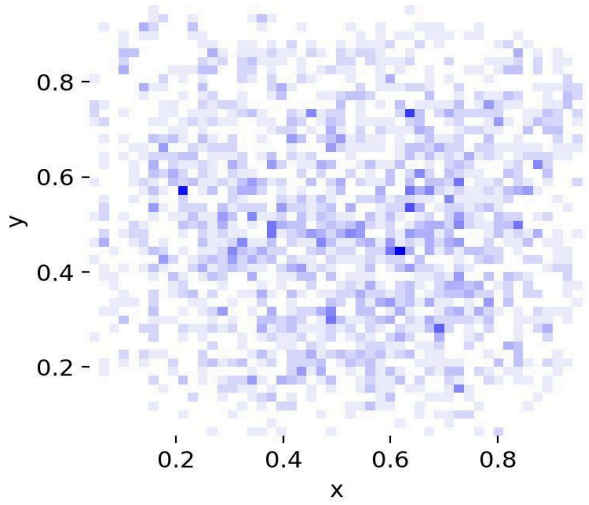
This curve visualizes the trade-off between precision and recall.
A curve that stays near the top-right region confirms robust performance, especially on subtle PCB defects.

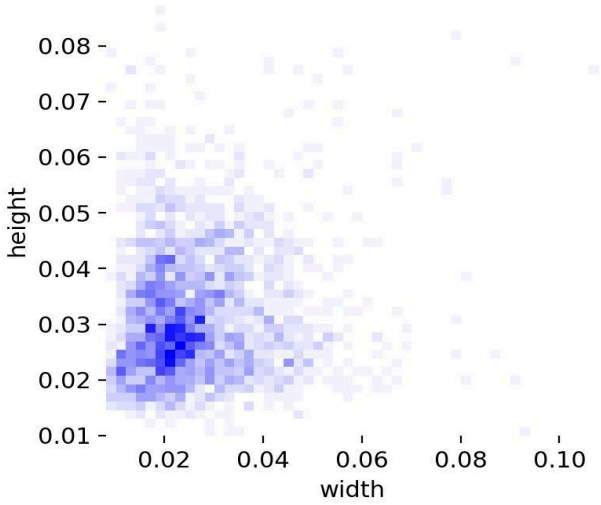


4. Label Distribution

Provides insight into dataset balance, bounding-box sizes, and defect frequency.
This helps explain training behavior and class imbalance effects.







8. Inference and Results

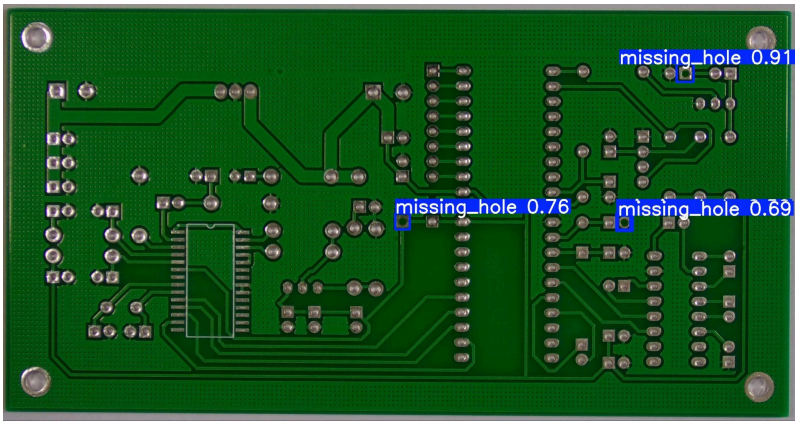
Model inference was tested on:

- Single PCB images
- Batch folders
- Multiple defect scenarios

Each output image contained:

- Bounding boxes
- Class labels
- Confidence values
- Annotated visualization

The model successfully detected both major and subtle defects with high accuracy.



---

9. Milestone 3 – Deployment, UI Development & Reporting System

Milestone 3 focused on transforming the trained YOLO11m model into a usable inspection application by separating concerns between the user interface and the inference logic.

Instead of embedding all processing directly into the frontend, CircuitGuard was redesigned with a **FastAPI backend** responsible for model inference and data processing, and a **Streamlit frontend** for user interaction and visualization.

9.1 Streamlit Frontend

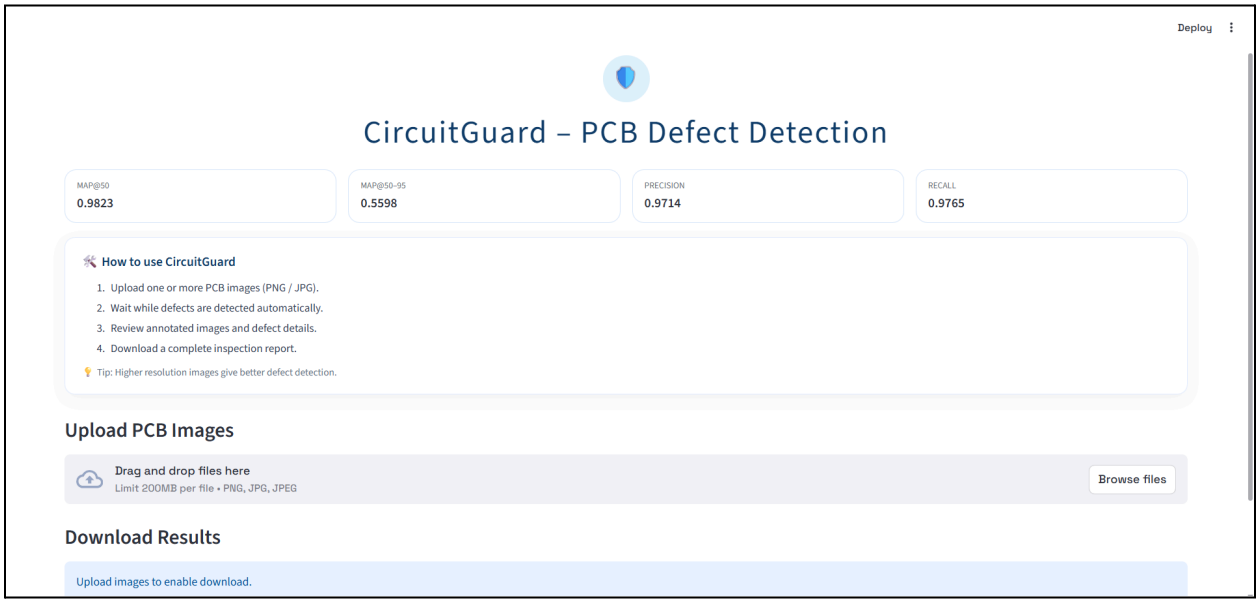
The Streamlit-based frontend provides a clean and responsive interface for PCB inspection with the following features:

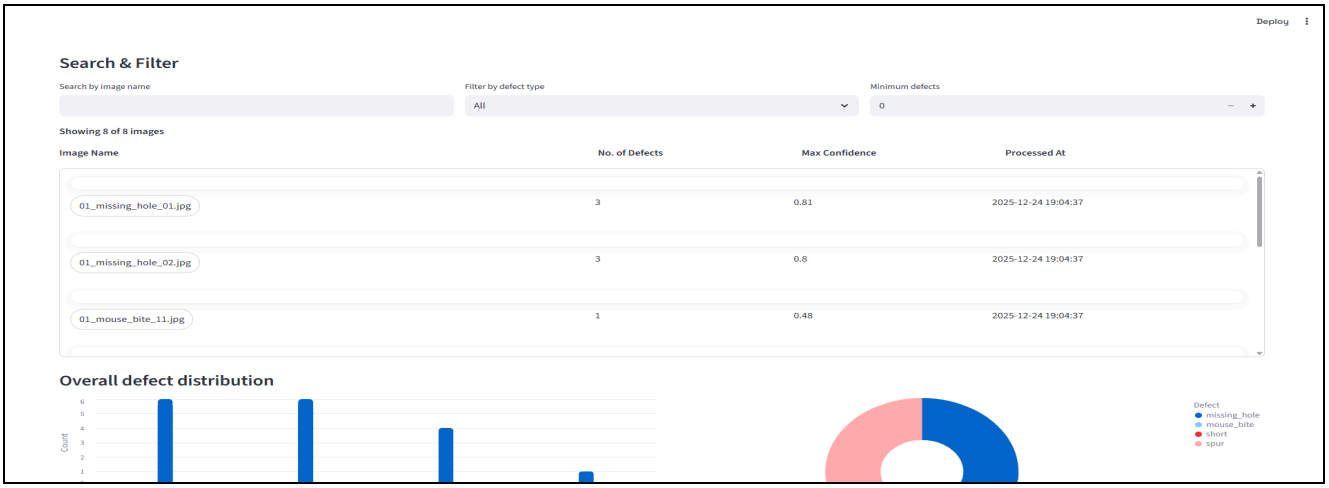
✔ Image and batch upload support
✔ Expandable image-wise result sections
✔ Original and annotated image visualization
✔ Defect summary tables
✔ Interactive charts for defect distribution
✔ Action-based export workflow

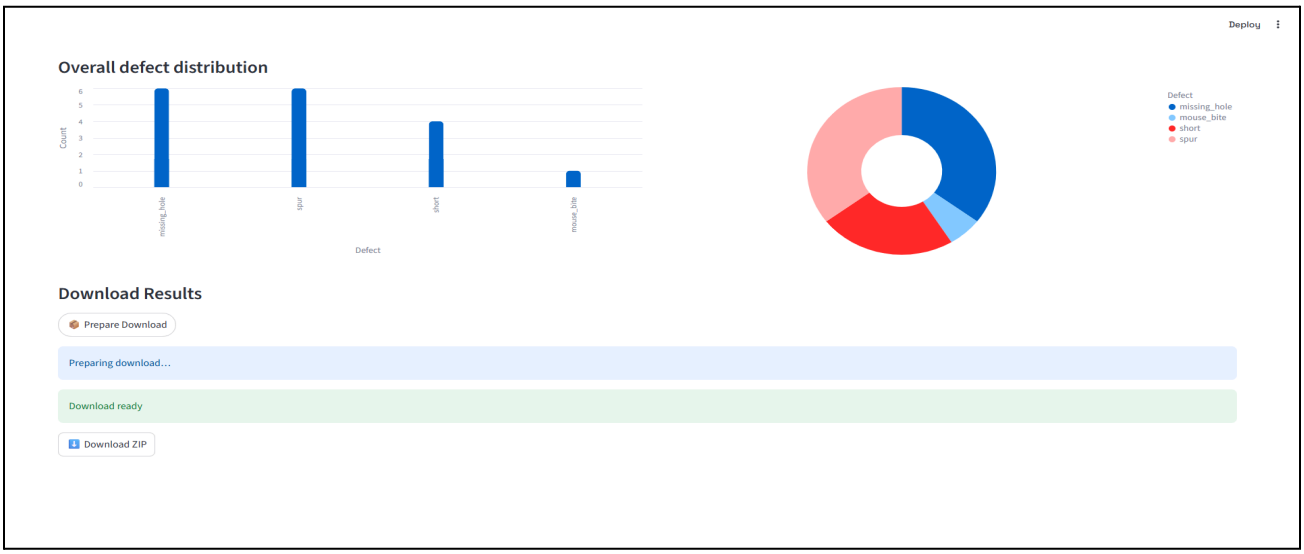
The frontend acts purely as a **presentation and interaction layer**, ensuring that UI complexity does not interfere with inference performance.

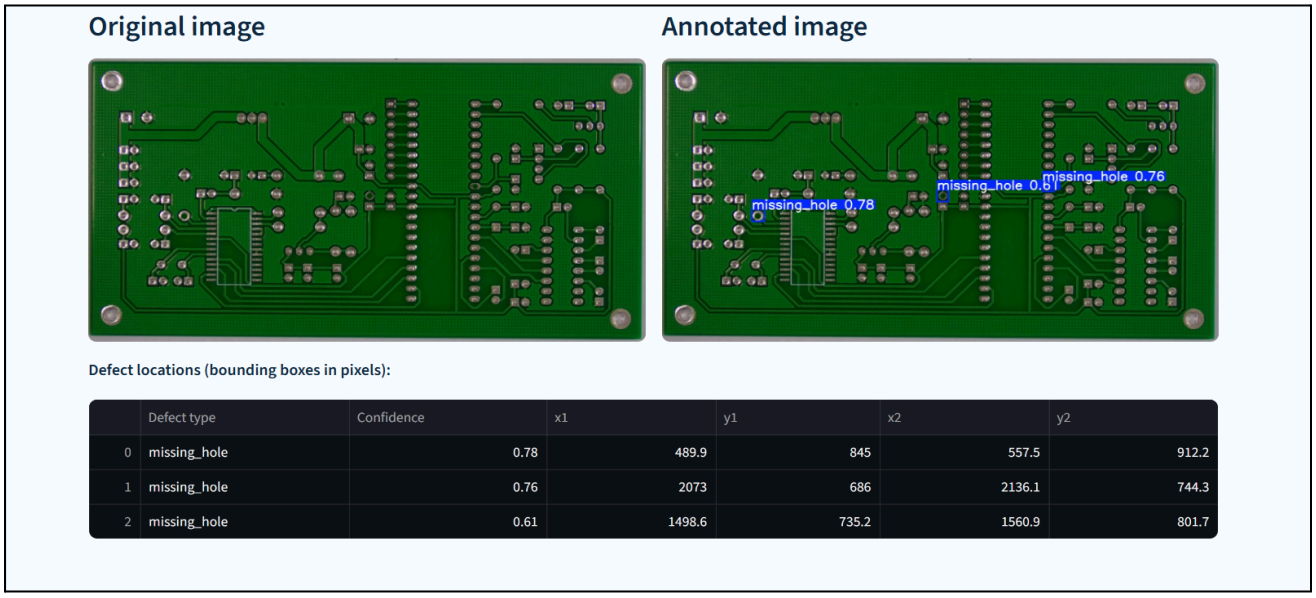
Users can click on any image row to expand details:

- Original image
- Annotated image
- Defect locations table









	Defect type	Confidence	x1	y1	x2	y2
0	missing_hole	0.78	489.9	845	557.5	912.2
1	missing_hole	0.76	2073	686	2136.1	744.3
2	missing_hole	0.61	1498.6	735.2	1560.9	801.7

9.2 FastAPI Backend for Inference

A FastAPI backend was implemented to handle:

- Model loading (YOLO11m)
- Image preprocessing
- Defect detection and bounding-box extraction
- Confidence scoring
- Structured response generation

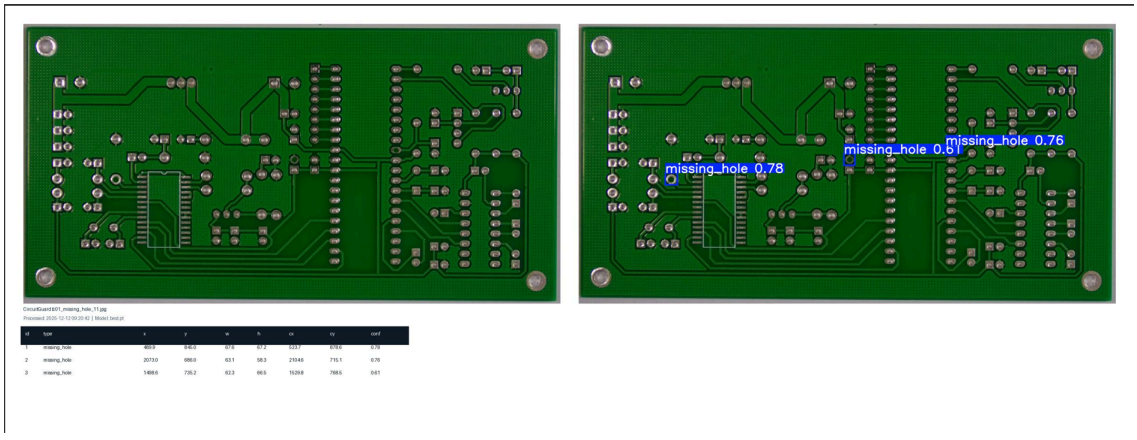
This backend runs locally and exposes REST endpoints that the frontend communicates with during inference. This design enables:

- Faster inference
- Better modularity
- Easier debugging and testing
- Future scalability (cloud or edge deployment)

9.3 PDF Generation per Image

Each inspected image generates a comprehensive PDF report containing:

- Original PCB image
- Annotated detection image
- Tabulated defect coordinates
- Confidence scores
- Timestamp of processing

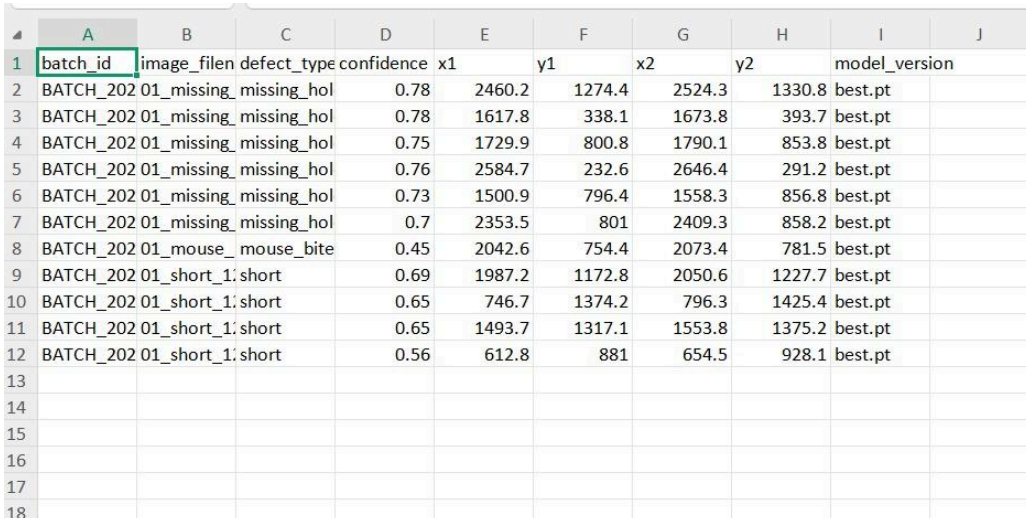


9.4 Combined CSV Export

A consolidated CSV file contains all defects detected across all images:

| Image | Defect Type | Confidence | x1 | y1 | x2 | y2 |

This enables manufacturing analytics and automated logging.



	A	B	C	D	E	F	G	H	I	J
1	batch_id	image_filen	defect_type	confidence	x1	y1	x2	y2	model_version	
2	BATCH_202	01_missing_	missing_hol	0.78	2460.2	1274.4	2524.3	1330.8	best.pt	
3	BATCH_202	01_missing_	missing_hol	0.78	1617.8	338.1	1673.8	393.7	best.pt	
4	BATCH_202	01_missing_	missing_hol	0.75	1729.9	800.8	1790.1	853.8	best.pt	
5	BATCH_202	01_missing_	missing_hol	0.76	2584.7	232.6	2646.4	291.2	best.pt	
6	BATCH_202	01_missing_	missing_hol	0.73	1500.9	796.4	1558.3	856.8	best.pt	
7	BATCH_202	01_missing_	missing_hol	0.7	2353.5	801	2409.3	858.2	best.pt	
8	BATCH_202	01_mouse_	mouse_bite	0.45	2042.6	754.4	2073.4	781.5	best.pt	
9	BATCH_202	01_short_1	short	0.69	1987.2	1172.8	2050.6	1227.7	best.pt	
10	BATCH_202	01_short_1	short	0.65	746.7	1374.2	796.3	1425.4	best.pt	
11	BATCH_202	01_short_1	short	0.65	1493.7	1317.1	1553.8	1375.2	best.pt	
12	BATCH_202	01_short_1	short	0.56	612.8	881	654.5	928.1	best.pt	
13										
14										
15										
16										
17										
18										

9.5 ZIP Export System

When users click **Finish Defect Detection**, the following are bundled in a ZIP file:

- All per-image PDFs (having original image, annotated image, and table consisting of defect location)
- Combined CSV file (defect location of all the images)

This provides a complete inspection package for industry workflows.

---

10. Milestone 4: System Finalization, Backend Optimization & Documentation

Milestone 4 focused on finalizing CircuitGuard as a stable, modular, and demonstration-ready PCB defect inspection system. While earlier milestones emphasized model training, evaluation, and UI development, this phase prioritized **system robustness, architectural clarity, backend optimization, and complete documentation**.

The goal of this milestone was to ensure that CircuitGuard operates reliably in a real-world inspection workflow while remaining easy to use, extend, and maintain.

10.1 Final System Architecture

In this milestone, CircuitGuard was finalized as a **two-tier local application** consisting of:

- **Frontend:** Streamlit-based user interface
- **Backend:** FastAPI-based inference and processing service

The FastAPI backend handles all computation-intensive tasks such as model loading, image preprocessing, defect detection, and result formatting. The Streamlit frontend interacts with the backend through REST APIs running on localhost, focusing purely on visualization and user interaction.

Architectural Benefits:

- Clear separation of concerns
- Improved scalability and maintainability
- Faster inference using local CPU/GPU resources
- No dependency on cloud services
- Enhanced data privacy for sensitive PCB images

This design aligns with industry-standard AI application architectures.

10.2 Backend Finalization and Optimization

The FastAPI backend was refined to ensure consistent and efficient inference across different usage scenarios.

Backend Capabilities:

- Single-image PCB inspection
- Batch processing of multiple PCB images
- Support for multiple defect instances per image
- Structured JSON responses for frontend rendering

Each inference request follows a standardized pipeline:

1. Image validation and preprocessing
2. YOLO11m model inference
3. Bounding box and class extraction
4. Confidence score computation
5. Result serialization

This optimized backend ensures stable performance even when processing large image batches.

10.3 Robust Batch Processing and Result Aggregation

Milestone 4 emphasized robustness during batch inspection, a key requirement for industrial quality control.

The system was tested with:

- Multiple PCB layouts
- Images containing multiple defects

Results from all images are aggregated in memory and presented through:

- Image-wise expandable result sections
- Global defect distribution summaries
- Combined statistical tables

This ensures that users can efficiently analyze inspection results at both micro (image-level) and macro (batch-level) scales.

10.5 System Testing and Validation

The finalized system was tested extensively to ensure reliability and usability.

Validation Observations:

- Stable inference across repeated runs
- No memory leaks during batch processing
- Accurate and consistent defect localization

- Responsive UI during large uploads

End-to-end processing time remained within acceptable limits for real-time or near-real-time inspection scenarios.

10.6 Documentation and Project Readiness

As part of Milestone 4, comprehensive documentation was prepared, including:

- Technical README with setup instructions
- Frontend and backend usage guide
- Project structure documentation
- Screenshots and workflow explanations

This ensures that the project can be:

- Easily reproduced by other users
- Demonstrated during evaluations
- Extended for future research or deployment

10.7 Milestone 4 Outcome Summary

By the completion of Milestone 4, CircuitGuard achieved:
✔ A modular FastAPI + Streamlit architecture
✔ Stable and optimized backend inference
✔ Robust batch inspection capability
✔ Industry-style reporting and export system
✔ Complete documentation and demo readiness

Milestone 4 successfully transformed CircuitGuard from a functional prototype into a **polished, reliable, and extensible PCB defect inspection system**.

11. Challenges Faced & Solutions

➔ Class Imbalance

Some defect types appeared less frequently.
Solution: Managed through careful evaluation + fine-tuning rather than augmentation.

➔ Subtle & Small Defect Regions

Tiny features like spurs or micro shorts were hard to detect.
Solution: Used YOLO11m (strong small-object detection) + fine-tuning.

➔ Annotation Format Conversion

XML → YOLO conversion required accurate scaling.
Solution: Custom converter and manual validation.

➔ Overfitting in Early Training

Initial training showed signs of overfitting.
Solution: Two-phase training (freeze → unfreeze).

➔ Frontend–Backend Integration Complexity

Separating the application into a Streamlit frontend and a FastAPI backend introduced challenges in coordinating data flow and response formats.
Solution: Designed standardized REST endpoints and consistent JSON response schemas, ensuring smooth communication between frontend and backend.

➔ Inference Speed and Performance

During initial implementation of the frontend–backend architecture, the system experienced slower inference times, particularly while processing multiple high-resolution PCB images in a batch. This affected the overall responsiveness of the application.
Solution: The backend was optimized by loading the YOLO11m model once at startup, streamlining preprocessing steps, and avoiding redundant computations. Batch processing logic was refined to improve throughput while keeping the UI responsive.

➔ Latency Due to Image Transfer Between Frontend and Backend

Transferring large image files from the Streamlit frontend to the FastAPI backend introduced additional latency, especially for batch uploads.

Solution: Image handling was optimized by reducing unnecessary image copies and ensuring efficient serialization and deserialization between frontend and backend.

---

12. Potential Applications

CircuitGuard can be used for:

✔ **Manufacturing QA Automation**

Detect defects instantly during production.

✔ **Electronics Assembly Verification**

Identify issues before soldering or component placement.

✔ **Predictive Maintenance**

Use defect patterns to forecast failure-prone PCB designs.

✔ **Research & Benchmarking**

Benchmark new defect datasets or testing deep learning techniques.

---

13. Industry Impact

CircuitGuard demonstrates the practicality of AI in manufacturing:

- **Significantly faster inspection cycles**
- **Reduced labor dependency**
- **Consistent detection quality**
- **Lower defective shipment rates**
- **Scalable integration into Industry 4.0 systems**

With reliable reporting (PDF + CSV), CircuitGuard supports digital traceability and compliance.

---

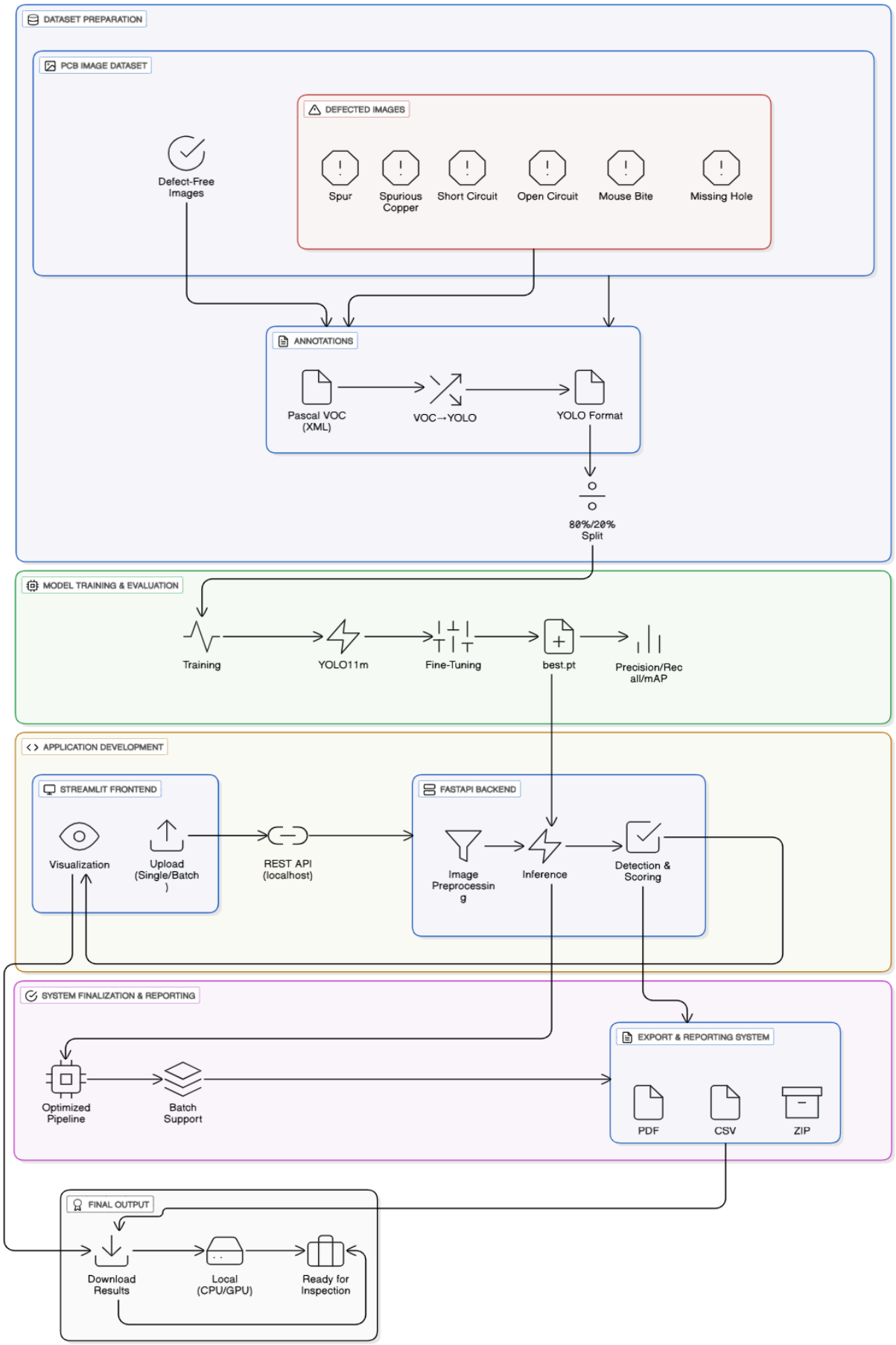
14. Conclusion

CircuitGuard successfully demonstrates an end-to-end PCB defect detection system using deep learning, achieving high accuracy and reliable inspection performance. The project evolved from dataset preparation and model training to a fully functional **local inspection system** with a modular architecture, combining a FastAPI backend for inference and a Streamlit frontend for user interaction.

The finalized system supports batch inspection, structured defect reporting, and exportable analysis outputs, making it suitable for academic evaluation and industrial-style quality assurance workflows. The separation of frontend and backend improves scalability, maintainability, and performance while ensuring data privacy through local execution.

CircuitGuard provides a strong foundation for future extensions such as larger and more diverse datasets, advanced data augmentation, cloud or edge deployment, and integration with automated manufacturing hardware.

★ Complete Project update in a glance:


DATASET PREPARATION
PCB IMAGE DATASET
Defect-Free Images
DEFECTED IMAGES
Spur
Spurious Copper
Short Circuit
Open Circuit
Mouse Bite
Missing Hole
ANNOTATIONS
Pascal VOC (XML)
VOC→YOLO
YOLO Format
80%/20% Split
MODEL TRAINING & EVALUATION
Training
YOLO11m
Fine-Tuning
best.pt
Precision/Recall/mAP
APPLICATION DEVELOPMENT
STREAMLIT FRONTEND
Visualization
Upload (Single/Batch)
REST API (localhost)
FASTAPI BACKEND
Image Preprocessing
Inference
Detection & Scoring
SYSTEM FINALIZATION & REPORTING
Optimized Pipeline
Batch Support
EXPORT & REPORTING SYSTEM
PDF
CSV
ZIP
FINAL OUTPUT
Download Results
Local (CPU/GPU)
Ready for Inspection