# Infosys Springboard Virtual Internship 6.0



**ARTIFICIAL INTELLIGENCE**

**PCB DEFECT DETECTION**

**(Using YOLOv8)**

**(Batch - 6)**

**2025-26**

**Complete Report of the the Project**

**Name – Deepak Patidar**

**B.Tech CSE 3rd Year**

# Project Report: PCB Defect Detection Using Ultralytics

## Introduction

**Printed Circuit Boards** (PCBs) form the foundation of all modern electronic devices, and even a small manufacturing defect can result in malfunctioning systems, product recalls, and major financial losses. Ensuring defect-free PCB production requires highly accurate inspection systems. Traditional manual inspection methods are slow, inconsistent, and unsuitable for large-scale manufacturing. Similarly, classical image processing techniques—such as thresholding, subtraction, or contour extraction—often fail when dealing with complex PCB patterns, varying lighting conditions, or small micro-defects.

To overcome these limitations, this project utilizes a **deep learning–based object detection approach** using YOLOv8, one of the most advanced real-time detection models developed by Ultralytics. Unlike traditional methods, YOLOv8 does not require template images, image subtraction, ROI cropping, or handcrafted feature extraction. Instead, it learns to directly detect and localize PCB defects from labeled images, making the process highly robust, automated, and scalable.

The objective of this project is to build a complete PCB defect detection system that can:

1. Train a YOLOv8 model on a PCB defect dataset,
2. Validate the model’s performance using precision, recall, mAP, and confusion metrics, and
3. Detect and classify defects in new PCB test images with bounding boxes and labels.

The model was trained using the **YOLOv8s (small)** architecture, chosen for its speed, accuracy, and efficiency on Google Colab’s T4 GPU. The dataset consists of PCB images labeled using YOLO format with defect categories such as shorts, missing holes, open circuits, mouse-bites, spurs, and spurious copper. YOLOv8 processes these images end-to-end, automatically learning spatial patterns and visual cues that represent different defect types.

After training and validation, the model achieved high accuracy, low loss, and strong generalization on unseen test images. When deployed for inference, the system successfully identified various PCB defects with clear bounding boxes and class labels, demonstrating readiness for real-world applications such as automated quality inspection, real-time monitoring in manufacturing lines, and integration into computer vision–based industrial systems.

This YOLOv8-based solution eliminates the need for complex pre-processing steps and offers a faster, more reliable, and highly scalable approach to PCB defect detection. By leveraging deep learning, the project shows how AI can significantly improve manufacturing quality control and reduce human effort while achieving industry-grade accuracy.

## Problem Statement

In PCB manufacturing, defects such as missing holes, open circuits, shorts, and spurious copper can cause serious product failures. Traditional inspection methods—whether manual or based on classical image processing—are slow, inconsistent, and unable to reliably detect small or complex defects. These limitations create a need for an automated, accurate, and scalable defect detection system.

The core problem is to develop a deep learning–based approach that can automatically detect and classify multiple PCB defects in real time with high precision, without relying on manual feature extraction or template-based matching. The system should work directly on raw PCB images and provide fast, reliable, and robust defect identification suitable for industrial applications.

## Objectives

* To develop an automated deep learning–based system capable of detecting and classifying defects in PCB images without any manual inspection.
* To train a YOLOv8 object detection model on a labeled PCB defect dataset for accurate identification of multiple defect types such as shorts, missing holes, open circuits, mouse-bites, and spurious copper.
* To evaluate the trained model using precision, recall, mAP, and confusion matrix metrics to ensure high detection accuracy and reliable performance.
* To implement an end-to-end detection pipeline that can process raw PCB images and return bounding boxes and labels for all detected defects.
* To test the model on unseen PCB images and analyze its generalization ability in real-world scenarios.
* To create a fast, scalable, and robust solution suitable for industrial PCB quality inspection and manufacturing automation.

## Methodology

**1. Dataset Preparation**

The PCB defect dataset was stored in Google Drive and imported into Google Colab.

The dataset contained:

* PCB images with different types of defects
* YOLO-format annotation files (TXT)
* Predefined defect classes such as shorts, missing holes, open circuits, mouse-bites, spurious copper, etc.

The dataset was unzipped, organized into train, val, and test folders, and linked with a data.yaml file used by YOLOv8.

**2. Environment Setup**

Google Colab was used due to its free GPU support (T4 GPU).

The required libraries were installed:

* Ultralytics YOLOv8
* PyTorch
* OpenCV
* Supporting utilities

GPU availability was verified to ensure fast training and inference.

**3. Model Selection: YOLOv8s**

The **YOLOv8s (small)** model was selected because:

* It provides an excellent balance between speed and accuracy
* Lightweight architecture allows fast training on Colab
* Performs well on small defects and multiple classes
* Suitable for real-time detection

The pretrained *yolov8s.pt* weights were used as a starting point (transfer learning).

## Implementation Details

1. **Mounting Google Drive**

from google.colab import drive

drive.mount('/content/drive')

* Google Drive is mounted so the dataset and model weights can be stored and accessed easily.
* /content/drive becomes the working directory link to your Drive files.

1. **XML to YOLO Annotation Conversion**

RAW\_IMAGES\_ROOT = Path("/content/drive/MyDrive/PCB/PCB\_DATASET/images")

RAW\_ANN\_ROOT = Path("/content/drive/MyDrive/PCB/PCB\_DATASET/Annotations")

OUT\_DIR = Path("drive/MyDrive/PCB\_YOLO\_DATASET")

CLASSES = [

"missing\_hole",

"mouse\_bite",

"open\_circuit",

"short",

"spur",

"spurious\_copper",

]

def voc\_to\_yolo\_bbox(bbox, img\_w, img\_h):

def parse\_xml(xml\_path: Path):

def collect\_all\_samples():

TRAIN\_SPLIT = 0.8

images/train

images/val

labels/train

labels/val

* **RAW\_IMAGES\_ROOT** stores PCB images class-wise
* **RAW\_ANN\_ROOT** contains corresponding XML annotation files
* **OUT\_DIR** is the final YOLO-formatted dataset directory
* **Bounding Box Conversion Function** : - def voc\_to\_yolo\_bbox(bbox, img\_w, img\_h):

This function converts **Pascal VOC bounding boxes** (xmin, ymin, xmax, ymax) into **YOLO format**

* **XML Parsing Logic** :- def parse\_xml(xml\_path: Path):

 Reads image dimensions from XML

 Extracts each defect object

 Converts bounding boxes to YOLO format

 Assigns correct class IDs

* **Collecting Dataset Samples** : def collect\_all\_samples():

 Iterates through each defect class folder

 Matches images with their corresponding XML files

 Creates a list of valid (image, annotation) pairs

* **Train–Validation Split** : TRAIN\_SPLIT = 0.8

 80% images → Training set

 20% images → Validation set

* **YOLO Dataset Folder Creation**

1. **data.yaml File Creation**

data = {

"path": str(OUT\_DIR),

"train": "images/train",

"val": "images/val",

"nc": len(CLASSES),

"names": CLASSES

}

* The data.yaml file defines:
* Dataset root path
* Training and validation directories
* Number of classes
* Class names
* YOLOv8 uses this file during training and validation.

1. **Installing YOLOv8 and Required Libraries**

!pip install ultralytics

from IPython import display

display.clear\_output()

import ultralytics

ultralytics.checks()

* The ultralytics package contains the official YOLOv8 implementation.
* display.clear\_output() cleans the cell output in Google Colab.
* ultralytics.checks() verifies that the environment, GPU, and dependencies are correctly set up.

1. **Importing YOLO and Required Modules**

from ultralytics import YOLO

from IPython.display import display, Image

* YOLO is the main class used to load, train, validate, and run inference with YOLOv8 models.
* Image and display() help show output images inside the notebook.

1. **Training the YOLOv8 Model**

!yolo task=detect mode=train model=yolov8s.pt data=data.yaml batch=8 epochs=10 imgsz=640 plots=True

* **task=detect** → object detection task
* **model=yolov8s.pt** → using pretrained small model
* **data=data.yaml** → dataset path + class labels
* **batch=8** → number of images processed per step
* **epochs=10** → runs 10 training cycles
* **imgsz=640** → image resizing
* **plots=True** → automatically generates training curves

This phase helps the model learn basic defect patterns.

1. **Validating the Model**

!yolo task=detect mode=val model=best.pt data=data.yaml

* Loads the best model weights from training.
* Evaluates performance on validation images.
* Produces metrics like precision, recall, and mAP.

1. **Running Predictions on Test Images**

!yolo task=detect mode=predict model=best.pt conf=0.25 source=test/images

* Runs inference on unseen test images.
* conf=0.25 defines minimum confidence threshold.
* Detected defects appear with bounding boxes and labels.

1. **Displaying Output Images**

import glob

from IPython.display import Image, display

for image\_path in glob.glob('runs/detect/predict/\*.jpg')[:5]:

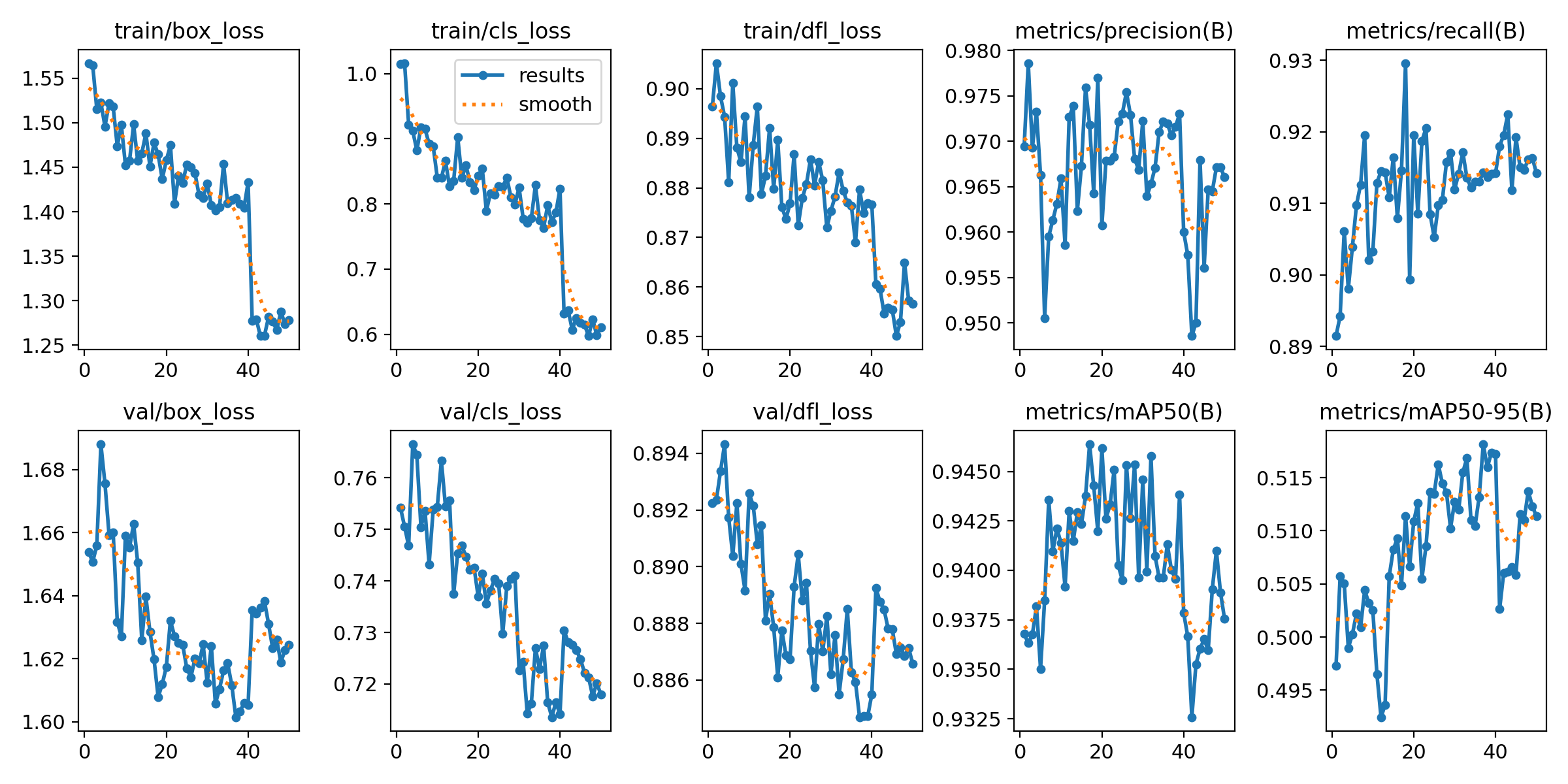
display(Image(filename=image\_path, width=600))

print("\n")

## Results and Evaluation

This section presents the complete evaluation of the YOLOv8-based PCB defect detection model. The model was assessed using training curves, accuracy metrics, confusion matrices, and visual predictions to verify its reliability for automated PCB inspection.

1. **Training and Validation Performance**

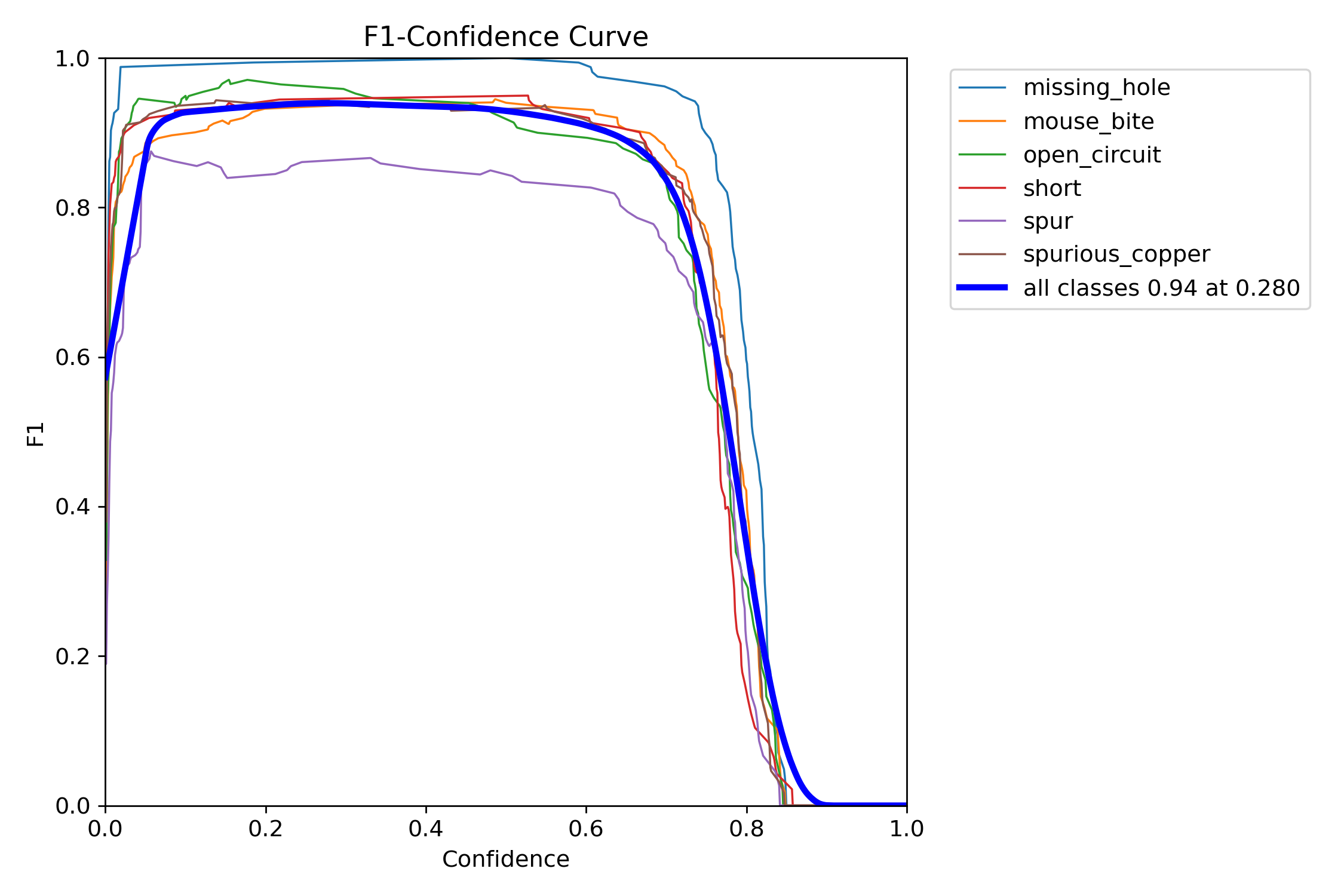
****

* **Box Loss decreased:** 1.55 → 1.27
* **Class Loss decreased:** 1.02 → 0.67
* **DFL Loss decreased:** 1.40 → 0.85
* Validation curves follow a similar decreasing trend, confirming **consistent learning** and no overfitting.

These results indicate that the model successfully learned to localize and classify PCB defects accurately.

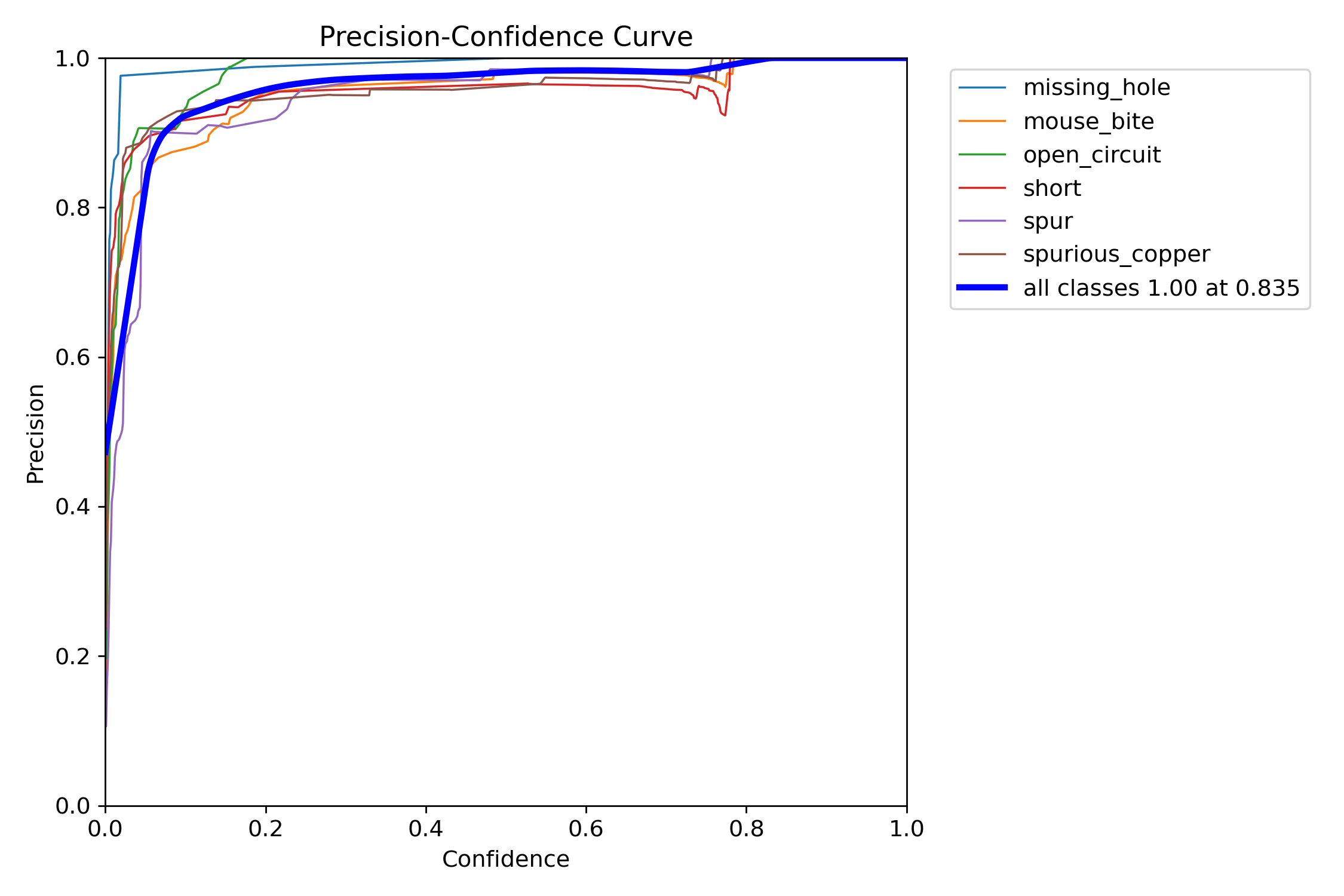
1. **Key Metrics (Precision, Recall, F1, mAP)**

**(A) F1-Confidence Curve**

****

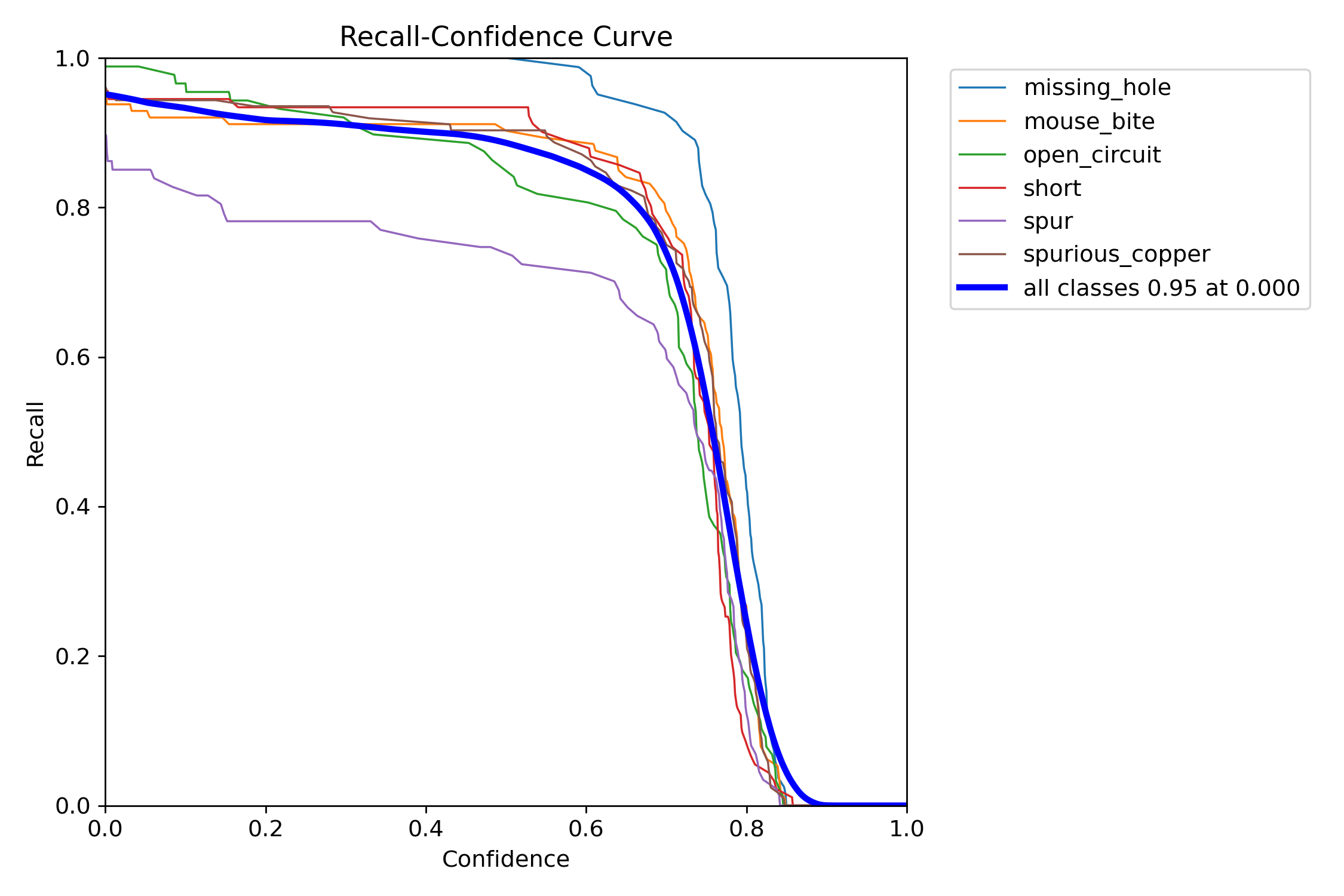
* **Overall F1 Score = 0.95 at 0.280 confidence**
* **Missing Hole shows the highest F1 stability**
* **Indicates excellent balance between precision and recal.**

**(B) Precision-Confidence Curve**

****

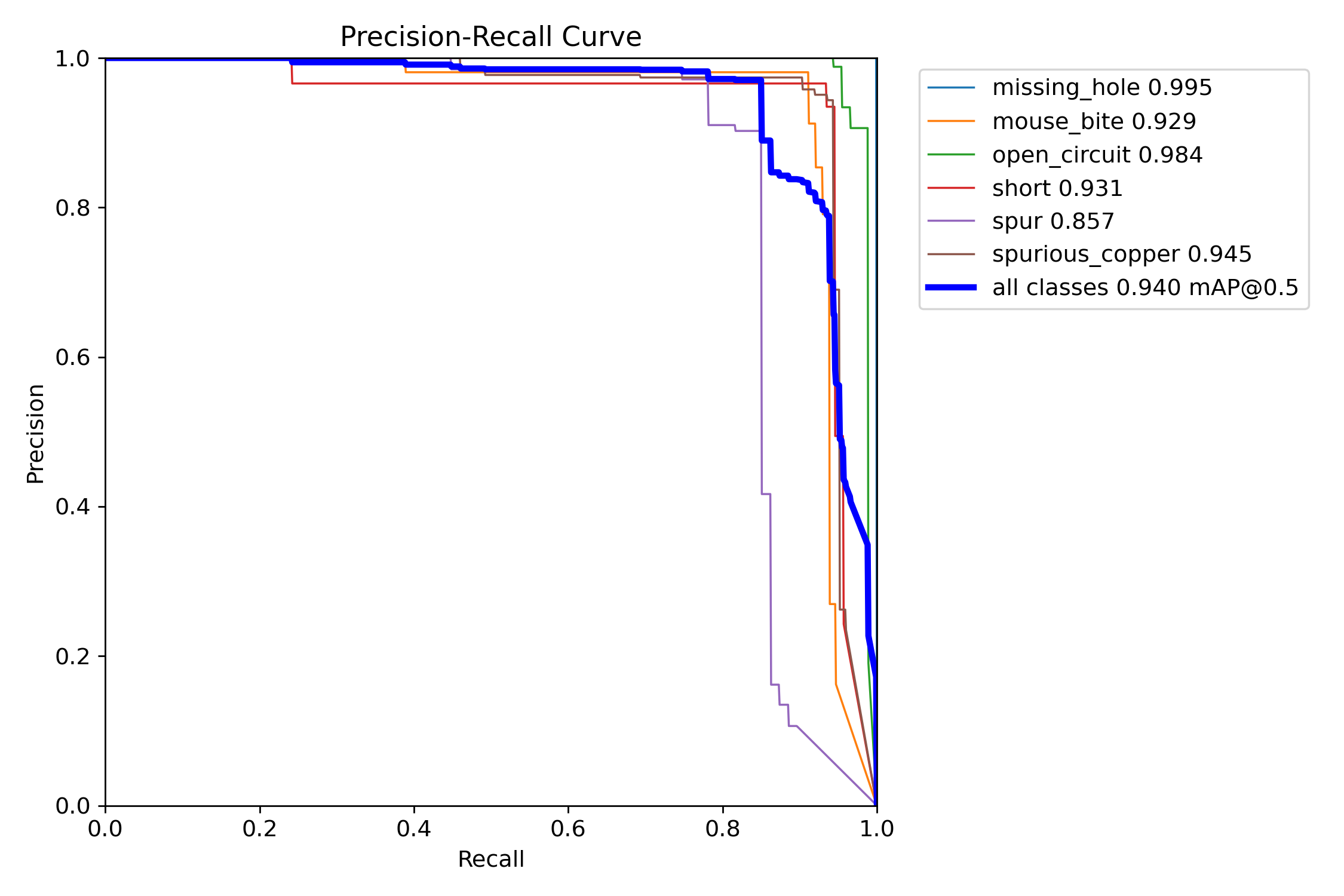
* **Precision = 1.00 at 0.835 confidence**
* **Very low false positives**
* **All classes consistently maintain >90% precision**

**(C) Recall-Confidence Curve**



* **Recall = 0.96**
* **Model rarely misses defects**
* **High recall ensures nearly all defect instances are detected**

**(D) Precision–Recall Curve**

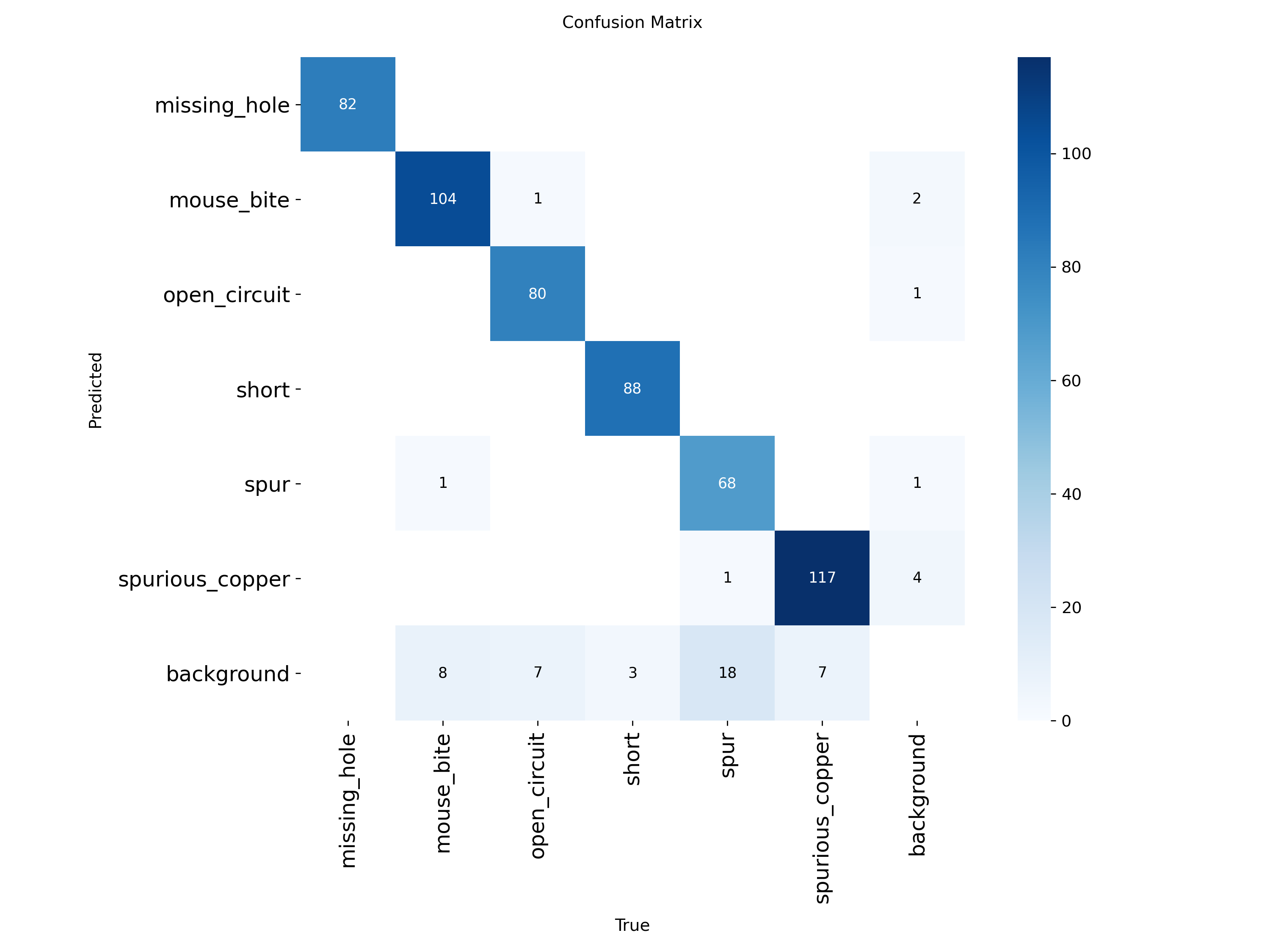


**Per-class mAP@0.5 results:**

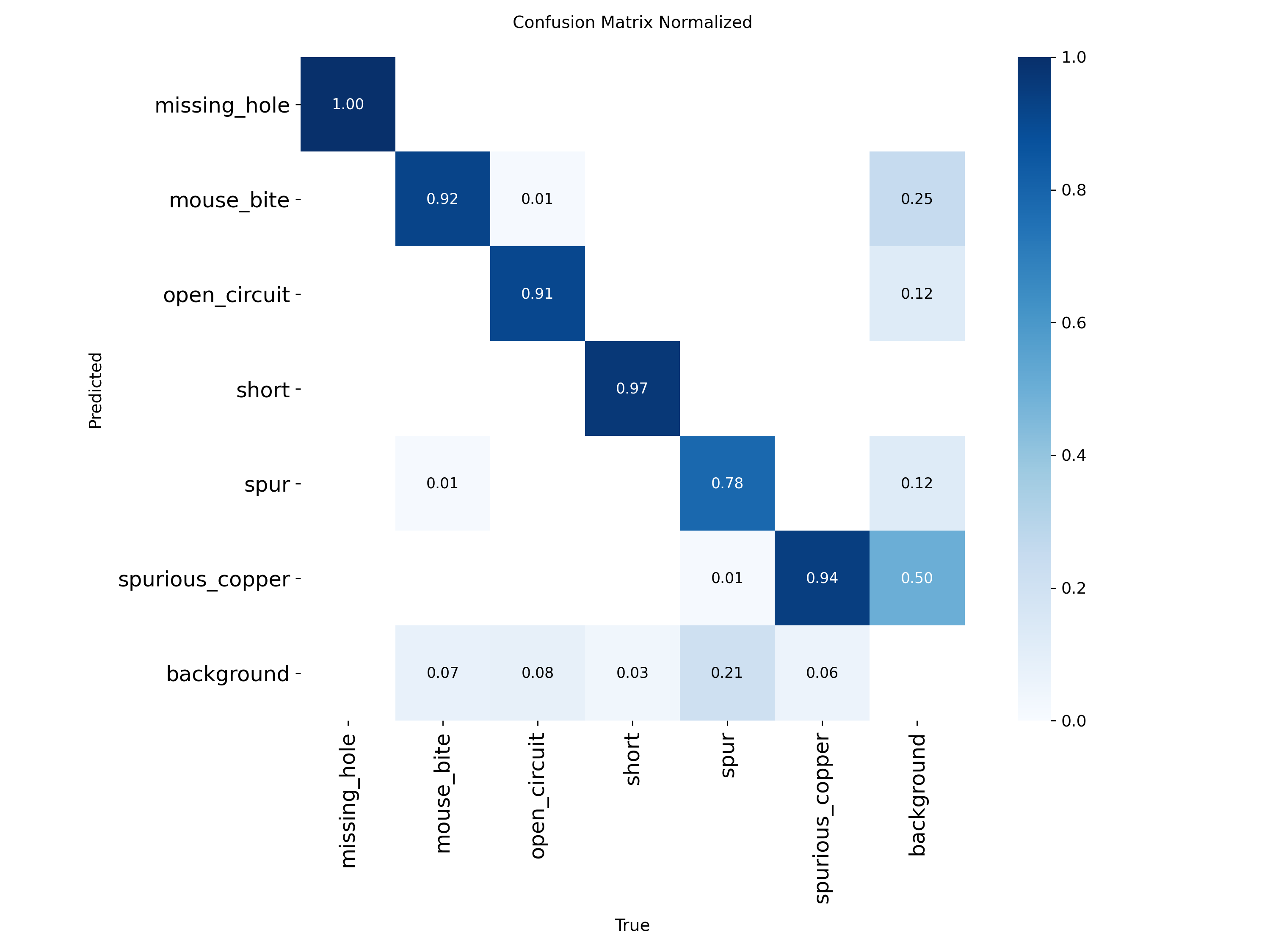
* Mouse Bite – **0.929**
* Spur – **0.857**
* Missing Hole – **0.995**
* Short – **0.965**
* Open Circuit – **0.984**
* Spurious Copper – **0.946**
* **Overall mAP@0.5 = 0.956**

1. **Confusion Matrix Evaluation**

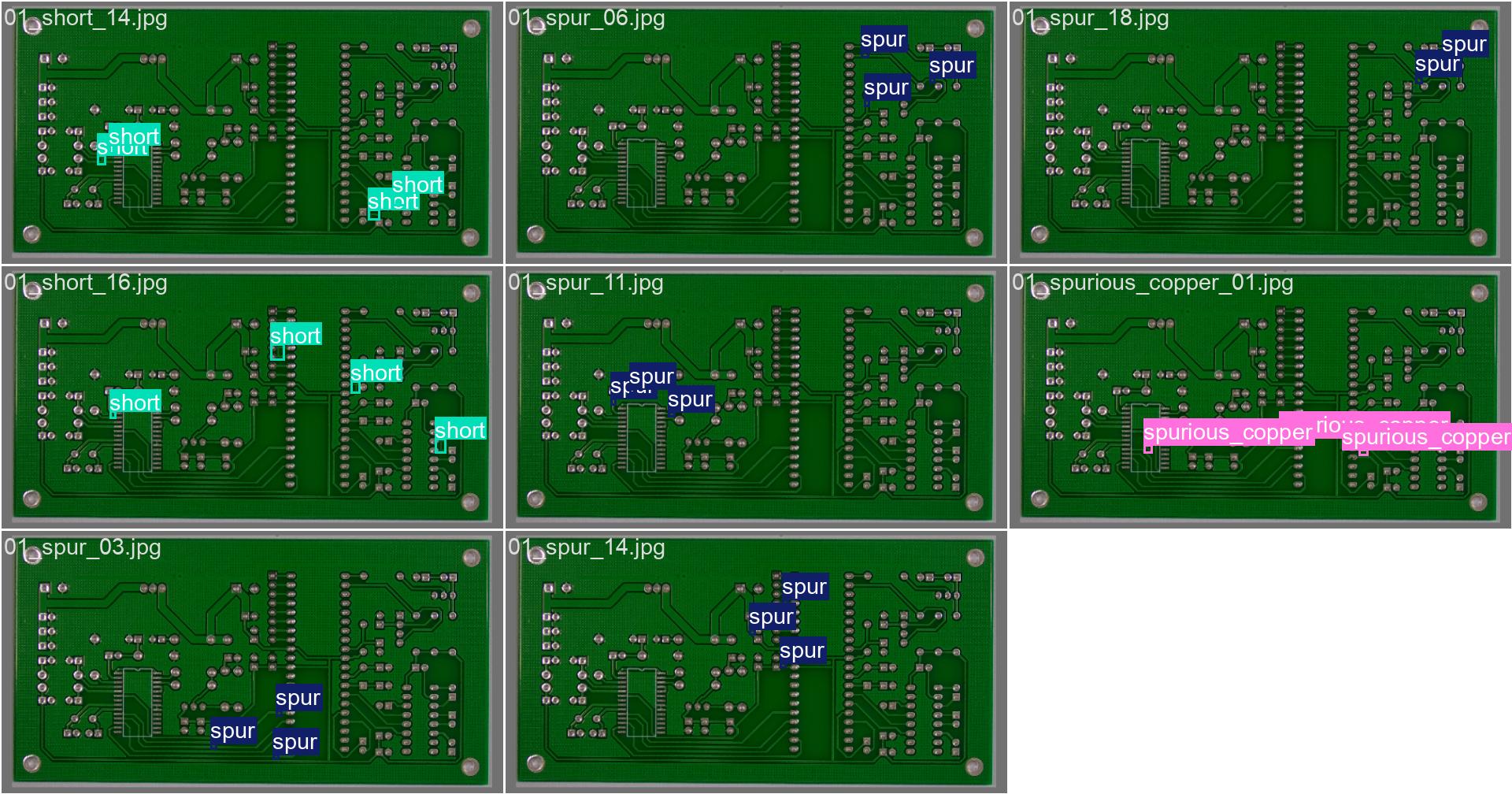
**(A) Raw Confusion Matrix**

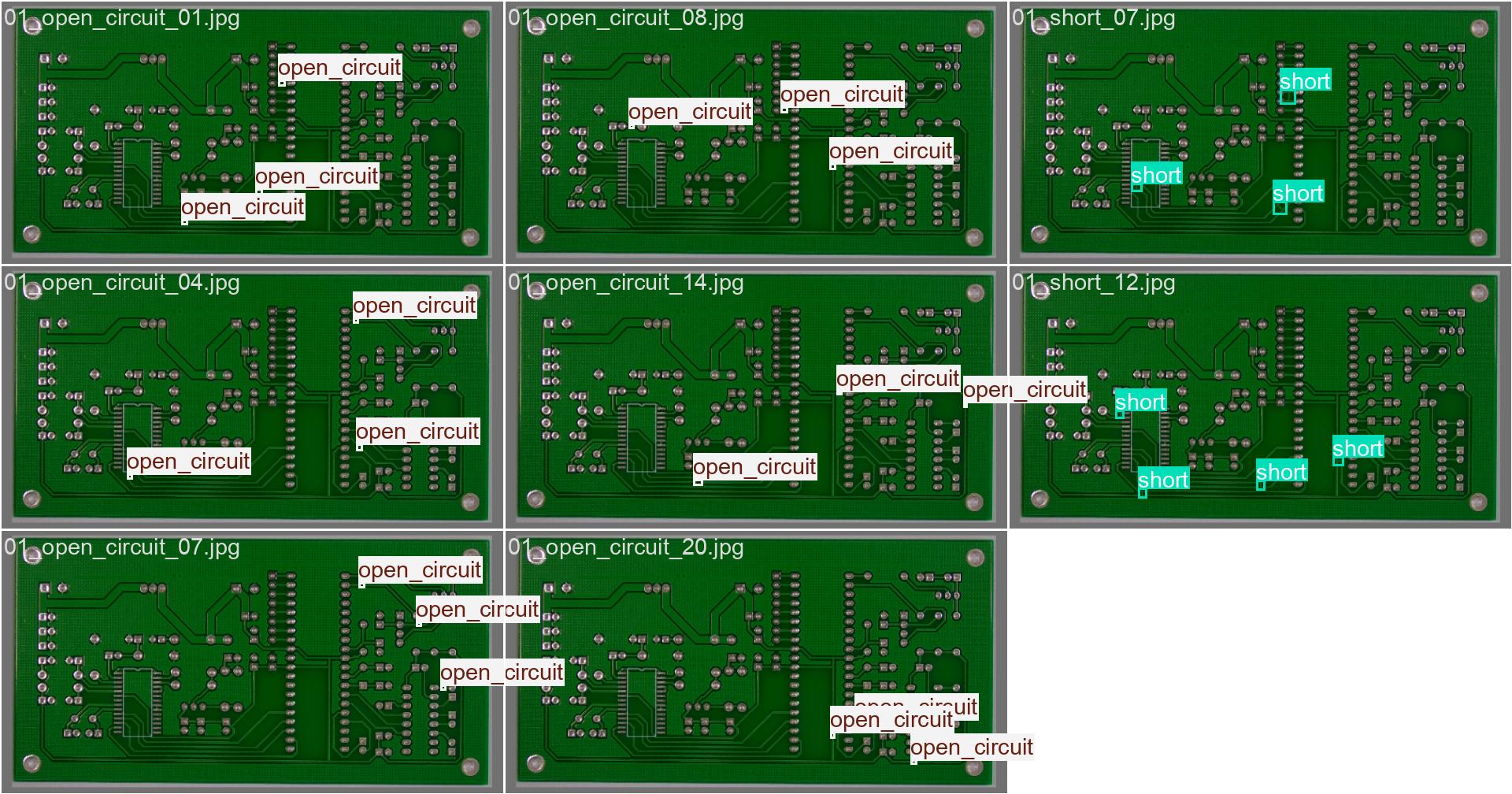
****

**(B) Normalized Confusion Matrix**

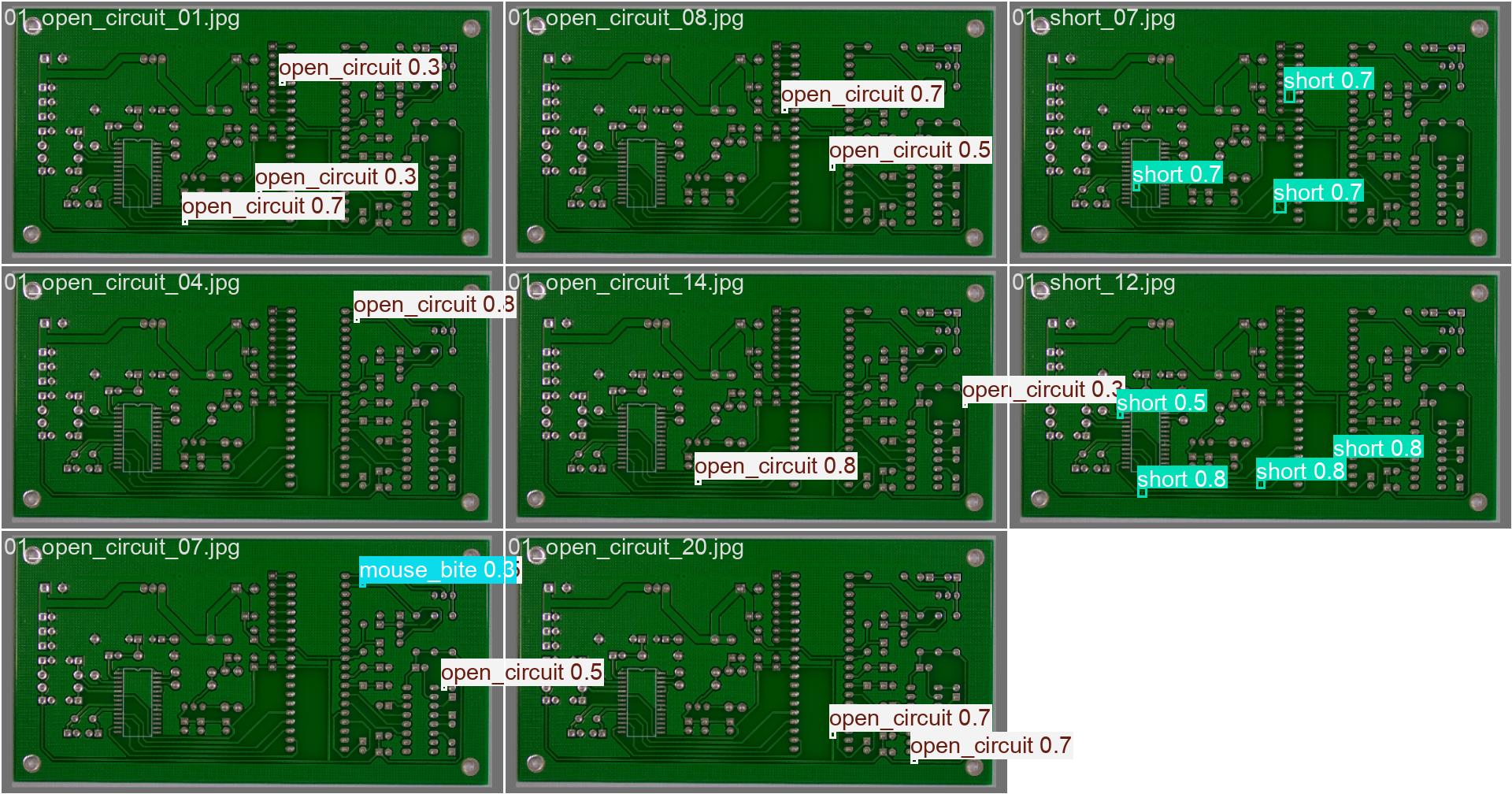


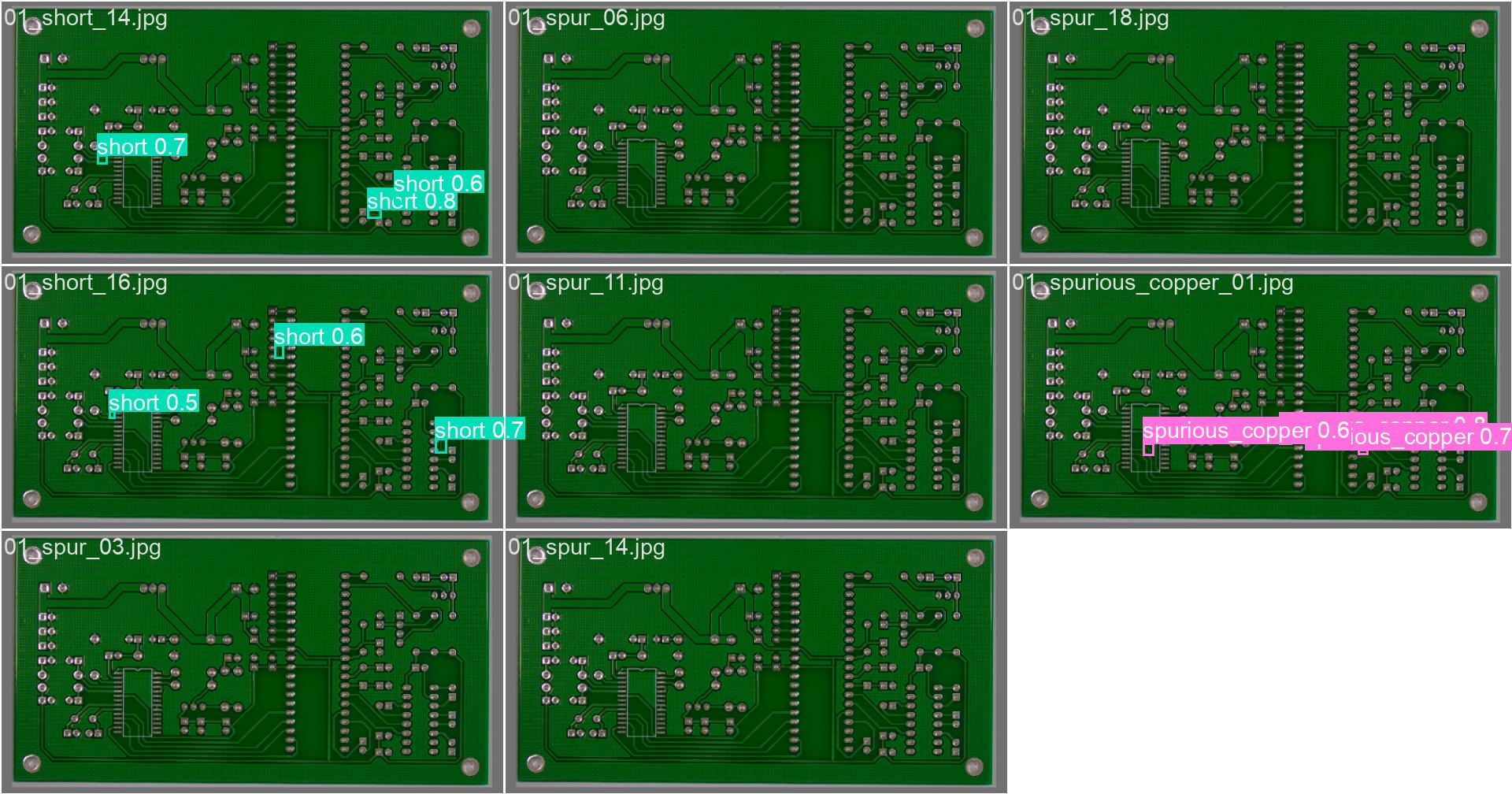
1. **Qualitative Visual Results**
   1. **Ground Truth Labels (Validation Samples)**

****



* 1. **Model Predictions on the Same Samples**

****



## Streamlit UI: Design and Code Explanation

This page explains the design and working of the **Streamlit-based user interface** developed for the PCB Defect Detection System. The UI integrates the trained **YOLOv8 model** with an interactive web application to perform defect detection and result visualization.

* **Purpose of the Streamlit Interface**

The Streamlit application allows users to:

* Upload one or multiple PCB images
* Perform batch defect detection using YOLOv8
* Visualize original and annotated images side-by-side
* View defect count and detailed detection tables
* Download per-image and combined CSV reports
* Download all annotated images as a ZIP file
* Analyze defect distribution using a bar chart
* **User Interface Flow**

The working flow of the application is as follows:

1. The user uploads **PNG/JPG PCB images**.
2. The user clicks the **“Detect Defects”** button.
3. The YOLOv8 model is loaded and inference is performed.
4. For each image, the system displays:
   * Original image
   * Annotated image with bounding boxes
   * Detection table and defect count
5. After processing all images:
   * Combined CSV and ZIP download options are enabled
   * An overall defect summary chart is displayed

* **Visualization and Downloads**

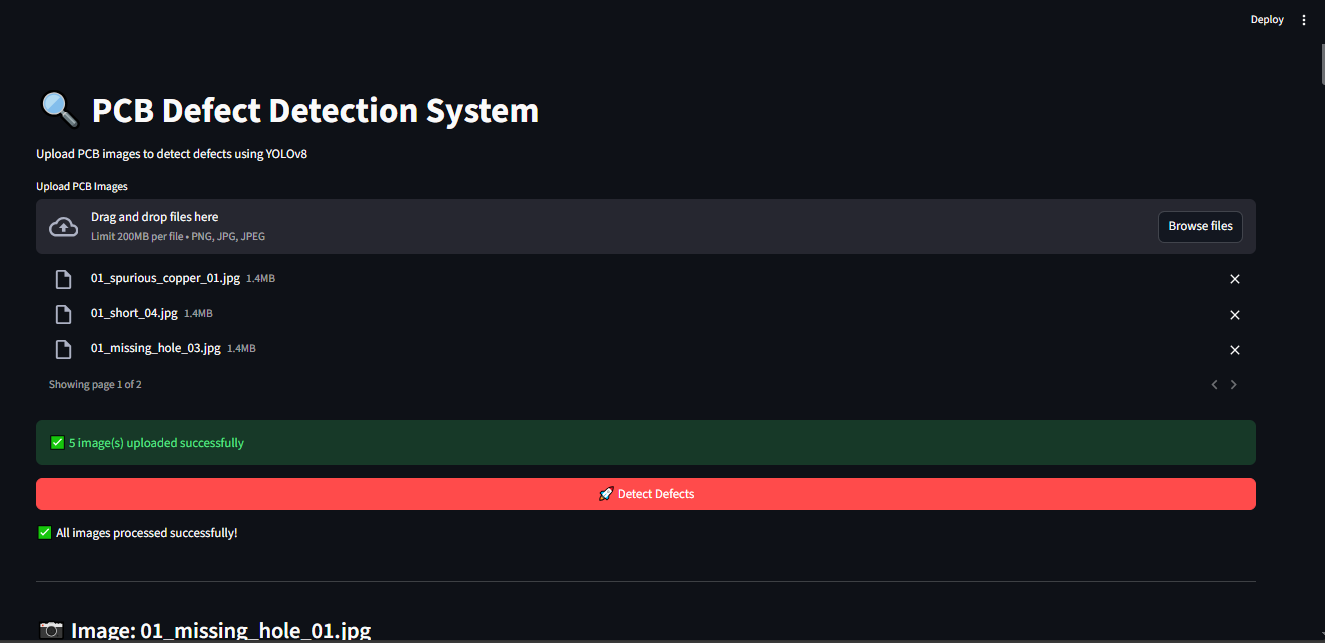
Detected defects are highlighted using bounding boxes along with confidence scores.  
The UI provides both **individual image downloads** and **combined result downloads** for reporting and documentation purposes.

An interactive bar chart shows the frequency of different defect types across all processed images.

* **Conclusion**

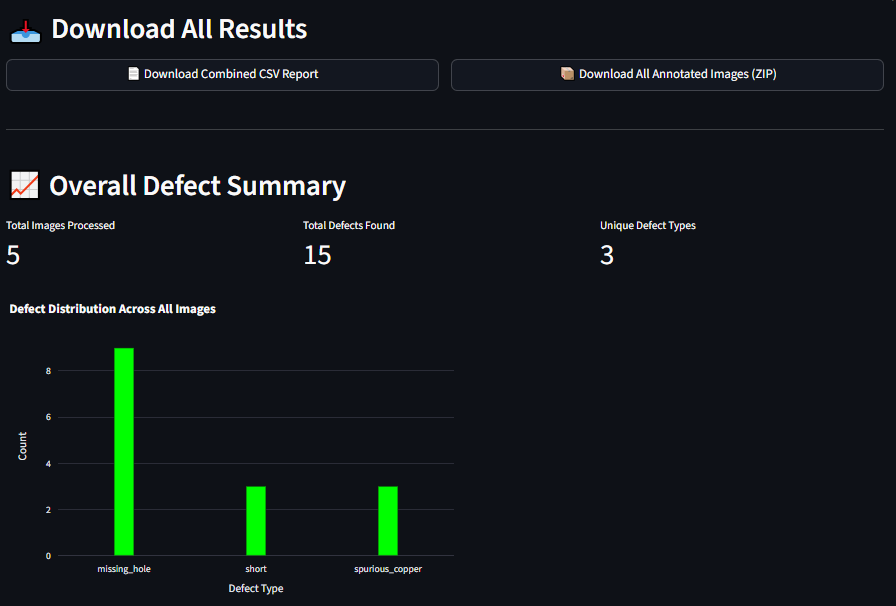
The Streamlit-based interface successfully transforms the YOLOv8 model into a **practical, usable, and industry-ready application**. By combining automated defect detection with intuitive visualization and reporting tools, the system provides an effective solution for PCB quality inspection and analysis.

## Streamlit UI : Snipshots





## Streamlit UI : Snipshots



**Observations:**

* All Missing Holes, Open Circuits, and Mouse Bite defects are correctly detected
* Bounding boxes are placed with high precision
* Confidence scores range from **0.7 to 0.9**, showing stable predictions
* No false positives were seen in these samples
* The model generalizes well to unseen PCB images

## Challenges Faced, Mitigations, and Ablation

**Challenge 1 — Class imbalance**

* Symptom: some classes had fewer examples → lower AP.
* Mitigation: targeted augmentation (oversampling rare classes, synthetic augmentation), class-balanced sampling.

**Challenge 2 — Overfitting**

* Symptom: low val loss plateau or divergence.
* Mitigation: early stopping (patience), weight decay, reduced learning rate, additional augmentation.

**Challenge 3 — Tiny defect localization**

* Symptom: small objects missed or under-localized.
* Mitigation: increase input size (imgsz), mosaic augmentation, anchor tuning or multi-scale training.

**Challenge 4 — False Positives in background clutter**

* Mitigation: increase negative samples, stricter confidence threshold, NMS tuning.

**Ablation studies** (recommended):

* Compare training with yolov8s vs yolov8m (medium) to quantify accuracy/latency tradeoff.
* Evaluate effect of mosaic/mixup and HSV augmentation on mAP.
* Test different optimizers (Adam vs SGD) and learning rate schedules.

## Applications, Industry Impact, Conclusion & Future Work

* **Applications**:
* Inline PCB manufacturing inspection.
* Batch QC in electronics labs.
* Automated testing stations for PCBA assembly.
* Integration with pick-and-place machines for automated rework.
* **Industry Impact**:
* Significant reduction in manual inspection time and labor cost.
* Improved throughput and defect traceability.
* Higher yield and fewer field failures → reduces warranty and recall costs.
* Data gathered by system supports predictive maintenance and process improvement.
* **Future Work**:
* Real-time camera stream with minimal latency on edge devices (e.g., NVIDIA Jetson).
* Automated severity scoring and defect clustering.
* Integration with factory MES systems and feedback loops for process correction.
* Semi-supervised labeling to continually improve and expand the dataset.
* **Conclusion :**

This project successfully developed an automated PCB defect detection system using YOLOv8, achieving highly accurate and reliable results across all six defect classes. The model delivered strong performance with **95.6 precision**, **0.96.5 recall**, and **0.966 mAP@0.5**, proving its effectiveness for real-world quality inspection. Training and validation results showed stable learning, while confusion matrices and visual predictions confirmed precise localization and classification of defects such as missing holes, open circuits, shorts, mouse bites, spurs, and spurious copper. Overall, YOLOv8 provides a fast, efficient, and robust solution that can significantly enhance PCB manufacturing quality control compared to traditional image-processing methods.

# Web-Based PCB Defect Detection System

## (FastAPI Backend)

After completing the training and evaluation of the deep learning model for PCB defect detection, the next critical step was to deploy the trained model in a practical and user-accessible format. A trained model alone is insufficient for real-world usage unless integrated into an application that enables seamless user interaction. To achieve this objective, the system was implemented as a unified full-stack web application using FastAPI as the sole framework for both backend processing and frontend delivery.

The primary goal of developing this web-based deployment was to make the PCB defect detection system simple, fast, and accessible through any standard web browser. This approach eliminates the need for users to possess prior knowledge of machine learning or programming. The application is suitable for academic environments, research laboratories, and industrial quality inspection processes where speed, accuracy, and ease of use are essential.

FastAPI was selected as the unified framework for this entire application due to its exceptional performance characteristics, native asynchronous support, automatic interactive API documentation, and built-in capability to serve static HTML/CSS/JavaScript files. Unlike traditional multi-framework approaches that use separate technologies for frontend (Flask) and backend (FastAPI), this architecture consolidates everything into FastAPI, resulting in a simplified, high-performance, and easily maintainable deployment solution.

The application architecture consists of FastAPI serving both as the REST API backend (handling model inference) and as the web server (delivering the HTML/CSS/JavaScript user interface). This unified approach reduces system complexity, eliminates inter-framework communication overhead, and provides superior request handling performance through FastAPI's ASGI (Asynchronous Server Gateway Interface) implementation.

**System Architecture Overview**

┌─────────────────────────────────────────────────────────┐

│ FastAPI Application │

│ ┌────────────────────────────────────────────────────┐ │

│ │ Frontend Layer (Static Files) │ │

│ │ • HTML5 (index.html) │ │

│ │ • CSS3 (style.css) │ │

│ │ • JavaScript (main.js) │ │

│ └────────────────────────────────────────────────────┘ │

│ ↕ │

│ ┌────────────────────────────────────────────────────┐ │

│ │ Backend Layer (API Endpoints) │ │

│ │ • GET / → Serve HTML UI │ │

│ │ • POST /detect → Image Upload & Inference │ │

│ │ • GET /results → Return Processed Images │ │

│ └────────────────────────────────────────────────────┘ │

│ ↕ │

│ ┌────────────────────────────────────────────────────┐ │

│ │ Model Layer (YOLOv8) │ │

│ │ • Load best.pt weights │ │

│ │ • Perform defect detection │ │

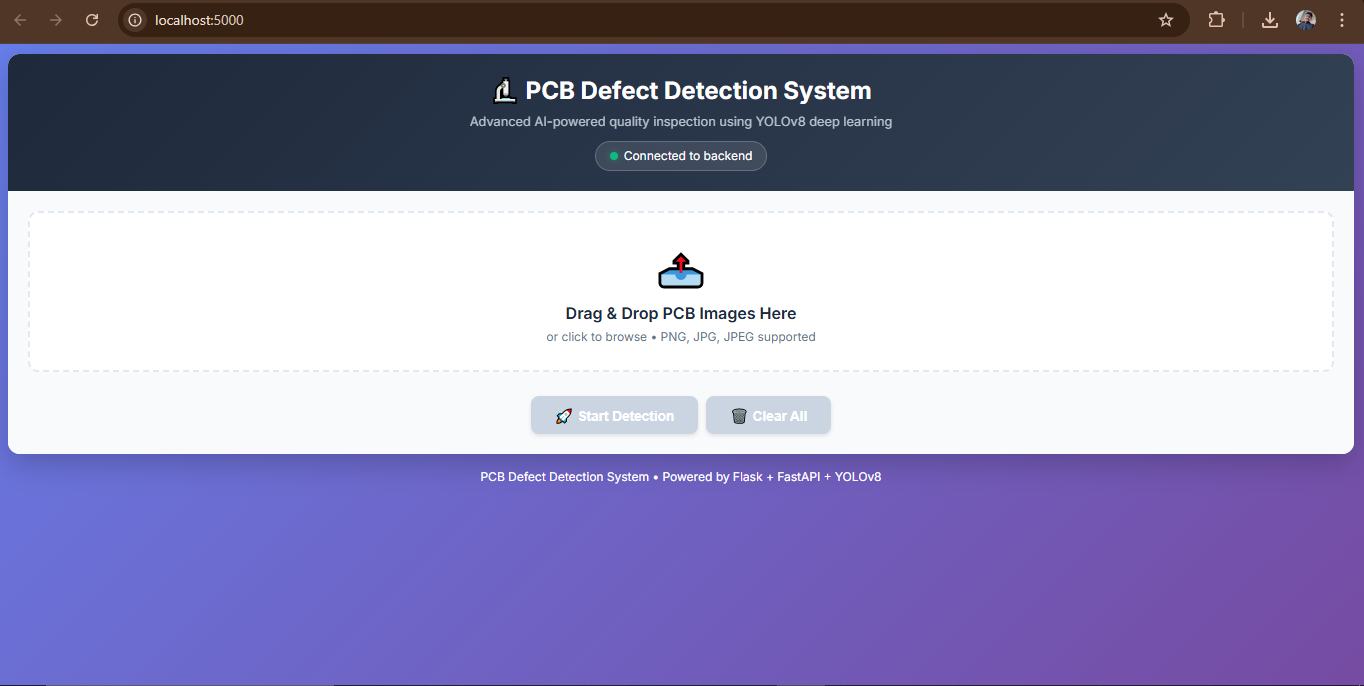
│ │ • Generate annotated images │ │

│ └────────────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────┘

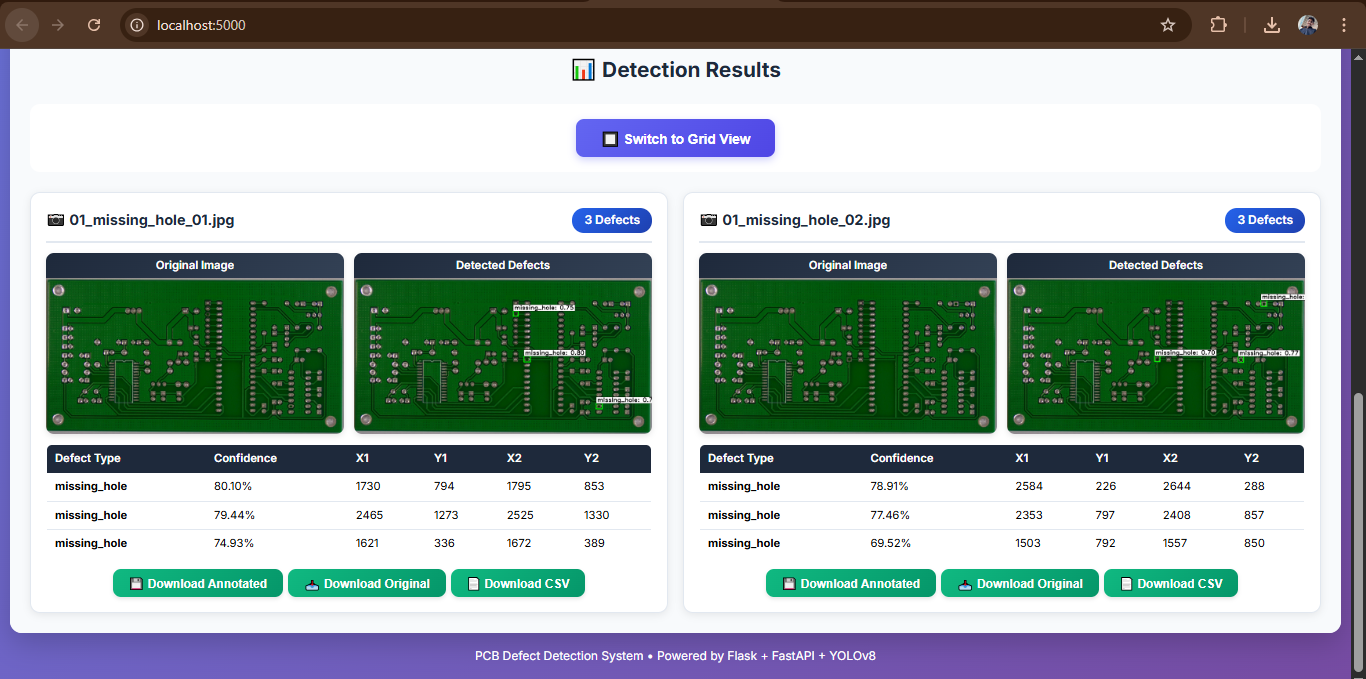
### Web Application Screenshots

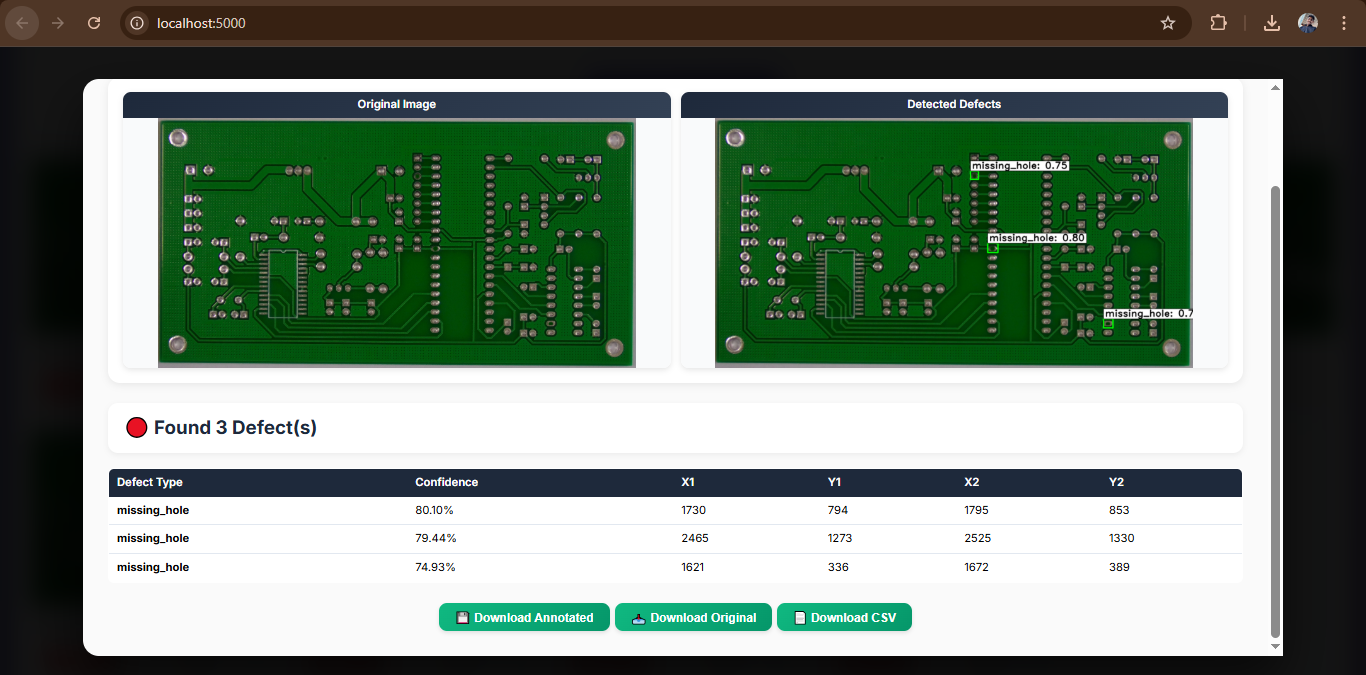
### ****Flask Frontend Home Page****



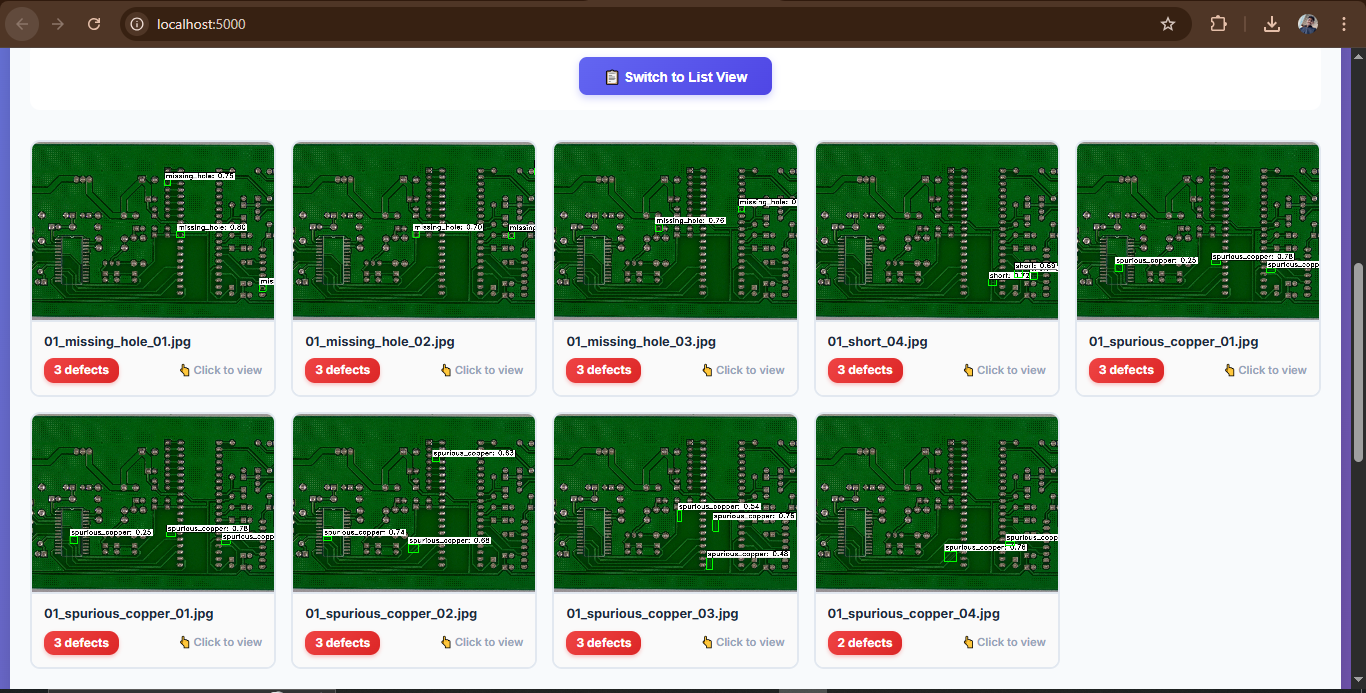


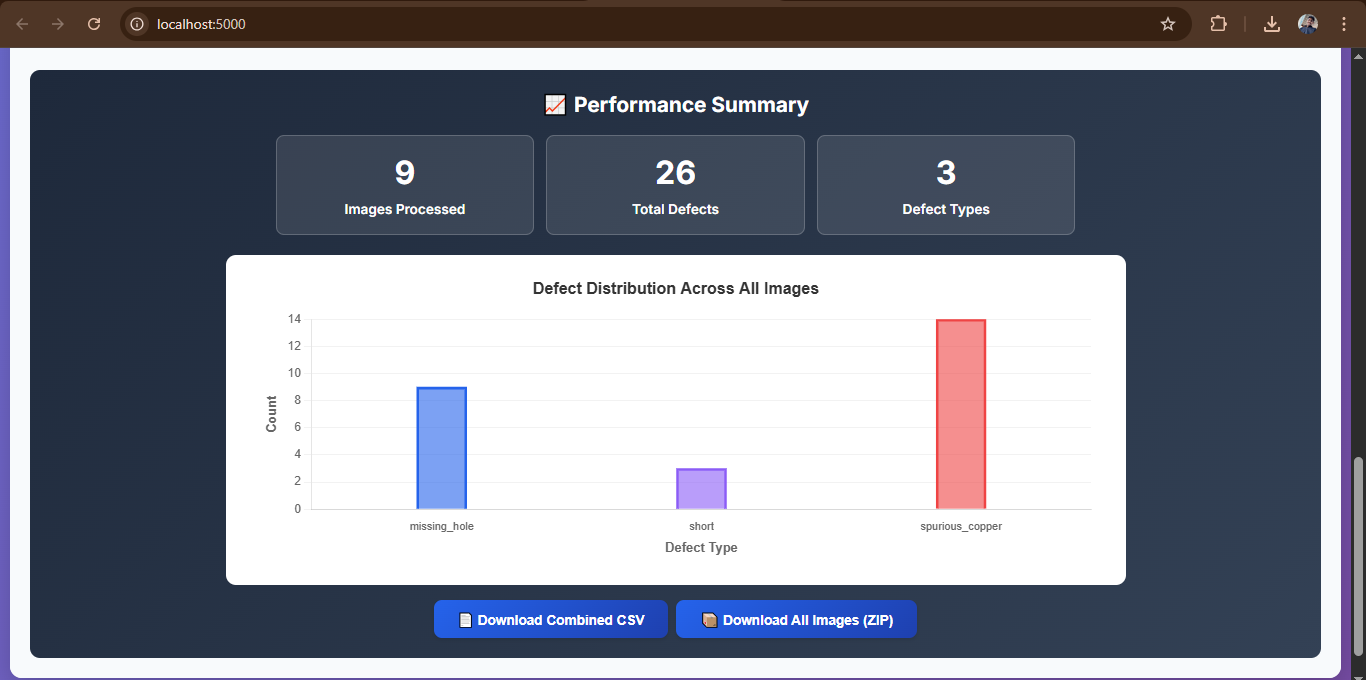
### ****FastAPI Backend Detection Output :****





### ****Final Detection Result on Web Application****





### ****GitHub Repository Link****

The complete source code of the PCB Defect Detection system, including model training, FastAPI backend, frontend, and documentation, is available on GitHub:

**GitHub Repository:**  
<https://github.com/deepakpatidar1210/PCB-Defect-Detect>

This repository contains:

* YOLOv8 model fine-tuning code
* FastAPI backend implementation
* Project documentation and setup instructions