PCB DEFECT DETECTION AND CLASSIFICATION: PROJECT REPORT

Project Title	**PCB Defect Detection and Classification**
Target Architecture	YOLOv11-M (Medium Variant)
Objective	Real-time detection and classification of six critical PCB defects
Key Performance	mAP50: 0.9863, Precision: 0.9780

---

1. Introduction

Printed Circuit Boards (PCBs) are indispensable components in virtually all modern electronic devices. The reliability and lifespan of these devices hinge critically on the quality of the PCB manufacturing process. Any flaws—such as missing holes, shorts, spurious copper, open circuits, mouse bites, or spurs—can lead to severe device malfunction or complete failure.

Traditionally, inspection relied on manual review or Automated Optical Inspection (AOI) systems. However, manual inspection is slow, inconsistent, highly labor-intensive, and requires expert knowledge. Traditional AOI systems, while faster, often struggle with the complexity of modern, dense PCB layouts and the detection of minute, low-contrast anomalies.

This project addresses these challenges by developing an automated defect detection system using a modern, high-performance object detection model, YOLOv11-M, fine-tuned on a custom PCB defect dataset. This solution aims to provide superior generalization, accuracy, and real-time processing capabilities for industrial quality control.

2. Comprehensive Explanation of Model Building (Methodology)

2.1. Dataset Description

The custom dataset used for training and validation consisted of annotated PCB images categorized into six critical defect classes:

Defect Class	Description
1. Missing Hole	Missing through-holes or vias
2. Mouse Bite	Irregular indentations or nicks on the edge of the board or trace
3. Open Circuit	A break in a conductive trace
4. Short	An unintended conductive path between two traces
5. Spur	A small, unintended projection extending from a trace
6. Spurious Copper	Unwanted patches of copper left on the board

The original dataset format involved images (JPG/PNG) paired with **XML annotations** (PASCAL VOC format). The images themselves contained a mix of horizontal and vertical orientations.

2.2. Data Preprocessing and Structuring

Since the chosen YOLO architecture requires specific label and directory formats, two main preprocessing steps were executed:

A. XML to YOLO TXT Label Conversion

1. **Tool Use:** The conversion was performed using the `XmlToTxt` Python tool.
2. **Setup:** Dependencies (`declxml`) were installed, and the six defect classes were explicitly defined in a `classes.txt` file.
3. **Output Format:** Each XML file was converted into a corresponding YOLO TXT file containing normalized bounding box coordinates in the format: `<class_id> <x_center> <y_center> <width> <height>`.

B. Dataset Structuring for Training

A custom Python script was created to organize the converted image-label pairs into the directory structure required by the Ultralytics framework:

- `train/images` and `train/labels`
- `val/images` and `val/labels`
- A `data.yaml` configuration file was created to link the dataset to the model.

2.3. Model Architecture: YOLOv11-M

The **YOLOv11-M (Medium variant)** model was selected as the base architecture. YOLOv11 is recognized as one of the latest iterations of the YOLO family, following the advancements made in YOLOv8 and YOLOv9.

The Medium variant ("M") was specifically chosen to balance the key performance trade-off in industrial applications:

- **Accuracy:** Medium models offer higher feature extraction capacity than Nano or Small models, providing the accuracy required to detect subtle, small-scale defects common on PCBs.
- **Speed:** It maintains a rapid inference speed (high Frames Per Second, FPS) suitable for real-time quality control on high-throughput production lines, unlike slower two-stage detectors like Faster R-CNN.

Key architectural features of the YOLO series utilized include:

- **Anchor-Free Detection:** YOLOv8/v11 replaced the anchor-based approach, simplifying the training process and improving localization.
- **Advanced Loss Functions:** Modern YOLO variants typically use DFL (Distribution Focal Loss) + CIoU/EIoU loss for regression and BCE Loss for classification, enhancing bounding box prediction accuracy.

2.4. Training and Hyperparameters

The model training was conducted on a **Kaggle GPU T4** instance using the following configuration:

Parameter	Value	Rationale
Base Model	yolo11m.pt	Pre-trained weights for faster convergence.
Epochs	20	Sufficient to demonstrate convergence for this fine-tuning task.
Image Size	640 * 640	Standard input resolution for a good balance of detail and speed.
Optimizer	Adam/SGD (auto-selected)	Ultralytics' default auto-selection optimizes convergence.
Confidence Threshold	0.25 (Typical default)	Controls the sensitivity of the detector.
IoU Threshold (NMS)	0.45 (Typical default)	Controls bounding box suppression for multiple detections of the same object.

```
model = YOLO('/content/yolo11m.pt')   # change if file name is different

Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11m.pt to '/content/yolo11m.pt': 100%

!cat /content/pcb-defect-dataset/data.yaml

names:
  0: mouse_bite
  1: spur
  2: missing_hole
  3: short
  4: open_circuit
  5: spurious_copper
path: ../pcb-defect-dataset
test: /content/pcb-defect-dataset/test/images
train: /content/pcb-defect-dataset/train/images
val: /content/pcb-defect-dataset/val/images
```

```
model.train(
    data="/content/pcb-defect-dataset/data.yaml",
    epochs=20,
    imgsz=640,
    batch=16
)

Ultralytics 8.3.235 🚀 Python-3.12.12 torch-2.9.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=16, bgr=0.0, box=7.5, cach
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf': 100% ━━━━━━━━━━━━ 755.1
Overriding model.yaml nc=80 with nc=6

                   from  n    params  module                                       arguments
  0                  -1  1      1856  ultralytics.nn.modules.conv.Conv             [3, 64, 3, 2]
  1                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  2                  -1  1    111872  ultralytics.nn.modules.block.C3k2            [128, 256, 1, True, 0.25]
  3                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
  4                  -1  1    444928  ultralytics.nn.modules.block.C3k2            [256, 512, 1, True, 0.25]
  5                  -1  1   2360320  ultralytics.nn.modules.conv.Conv             [512, 512, 3, 2]
  6                  -1  1   1380352  ultralytics.nn.modules.block.C3k2            [512, 512, 1, True]
```

3. Results and Detailed Analysis

The model was evaluated on the validation set, demonstrating a high degree of proficiency in detecting the six classes of PCB defects.

3.1. Quantitative Metrics

The primary evaluation metrics for object detection are Mean Average Precision (mAP), Precision, and Recall.

Metric	Definition	Value
mAP(50-95)	Mean Average Precision across IoU thresholds from 0.5 to 0.95	0.5763
mAP(50)	Mean Average Precision at IoU=0.50	0.9863
Precision	Ratio of correctly detected defects (True Positives) to all predictions	0.9780
Recall	Ratio of correctly detected defects to all actual defects (Ground Truth)	0.9858
F1-score	Harmonic mean of Precision and Recall	0.9815

```
task: 'detect'
```

[18]:

```
print(f"mAP50-95: {metrics.box.map:.4f}")
print(f"mAP50: {metrics.box.map50:.4f}")
print(f"Precision: {metrics.box.p.mean():.4f}")
print(f"Recall: {metrics.box.r.mean():.4f}")
print(f"F1-score: {metrics.box.f1.mean():.4f}")
```

```
mAP50-95: 0.5763
mAP50: 0.9863
Precision: 0.9780
Recall: 0.9850
F1-score: 0.9815
```

```
import cv2
```

Discussion of Performance:

The results indicate an outstanding performance in identifying and localizing the defects:

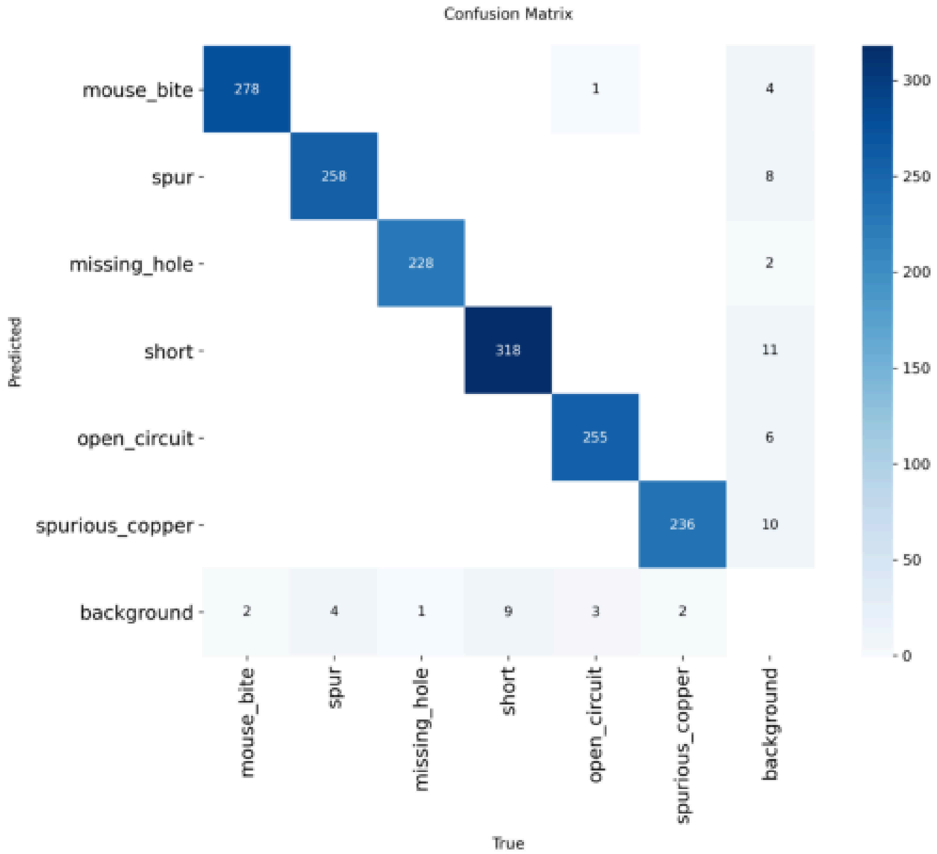
- **High mAP (50) (0.9863):** This score is near-perfect, confirming that 98.63% of the time, the model correctly identified the defect type and placed a bounding box that overlapped the ground truth box by at least 50% (IoU greater than or equal to 0.5).
- **High F1-score (0.9815):** The F1-score, supported by the balance between Precision and Recall, demonstrates that the model is both highly **accurate (low false positives)** and **robust (low false negatives)**. In quality control, minimizing False Negatives (missed defects) is critical, and the high Recall value confirms this robustness.
- **mAP (50-95) (0.5763):** While respectable, this metric is lower than the mAP(50), which is common for challenging datasets. It suggests that while the model finds defects easily, the precise bounding box localization (at higher IoU thresholds like 0.75 or 0.95) could be further optimized.

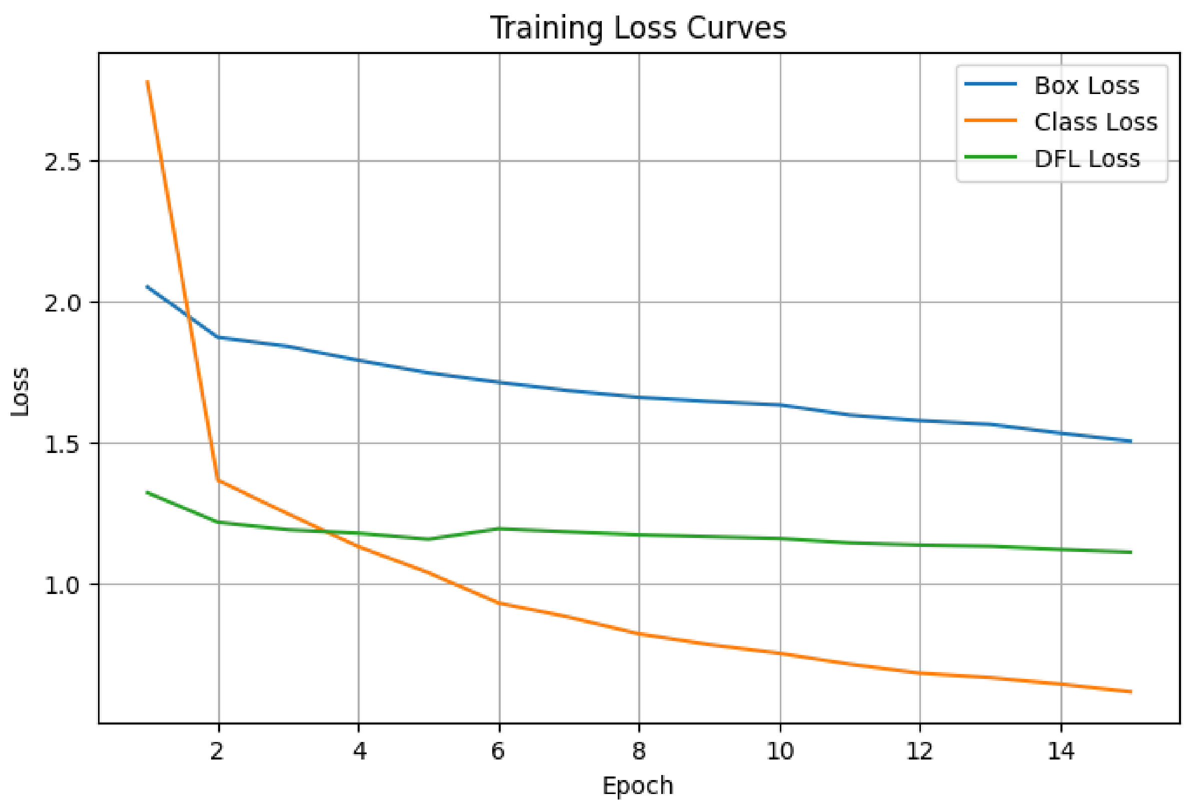
3.2. Visual Results and Model Convergence

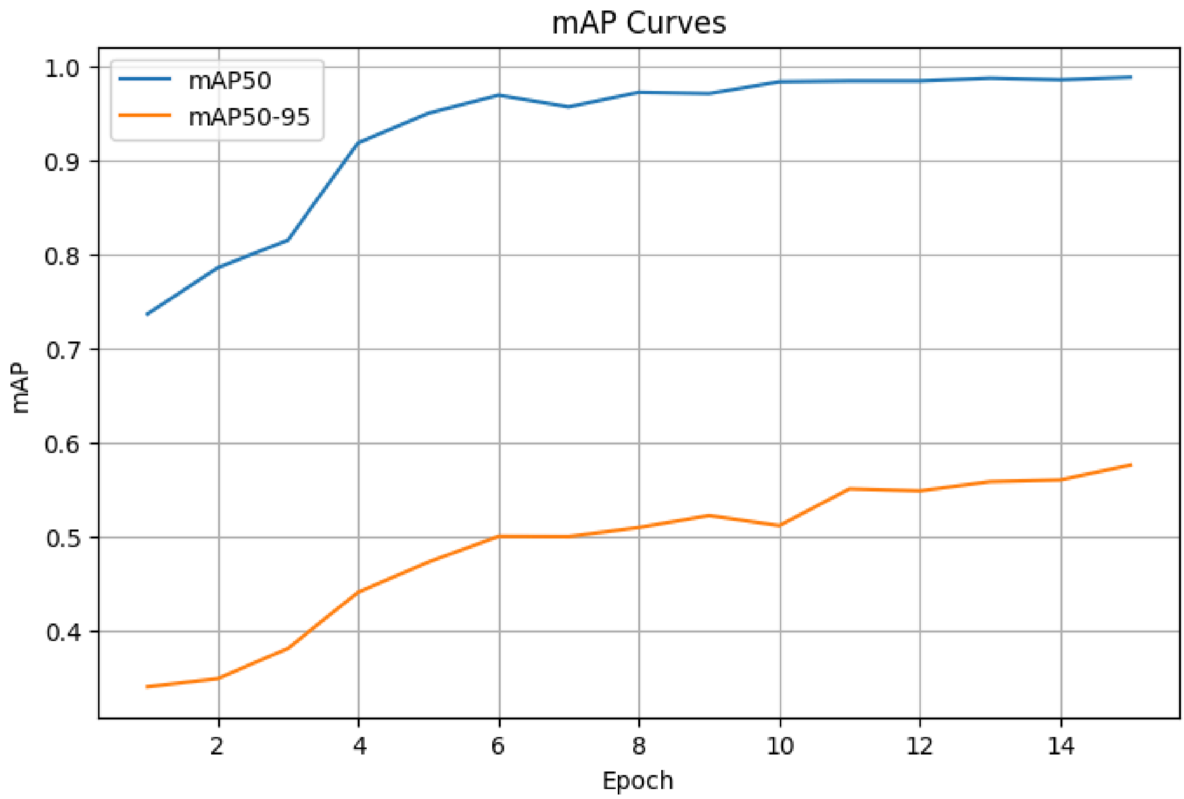
Visual inspection of the results, such as the prediction of a 'missing hole' with confidence scores of 0.73 and 0.71, confirmed the model's ability to accurately classify and localize different defect types.

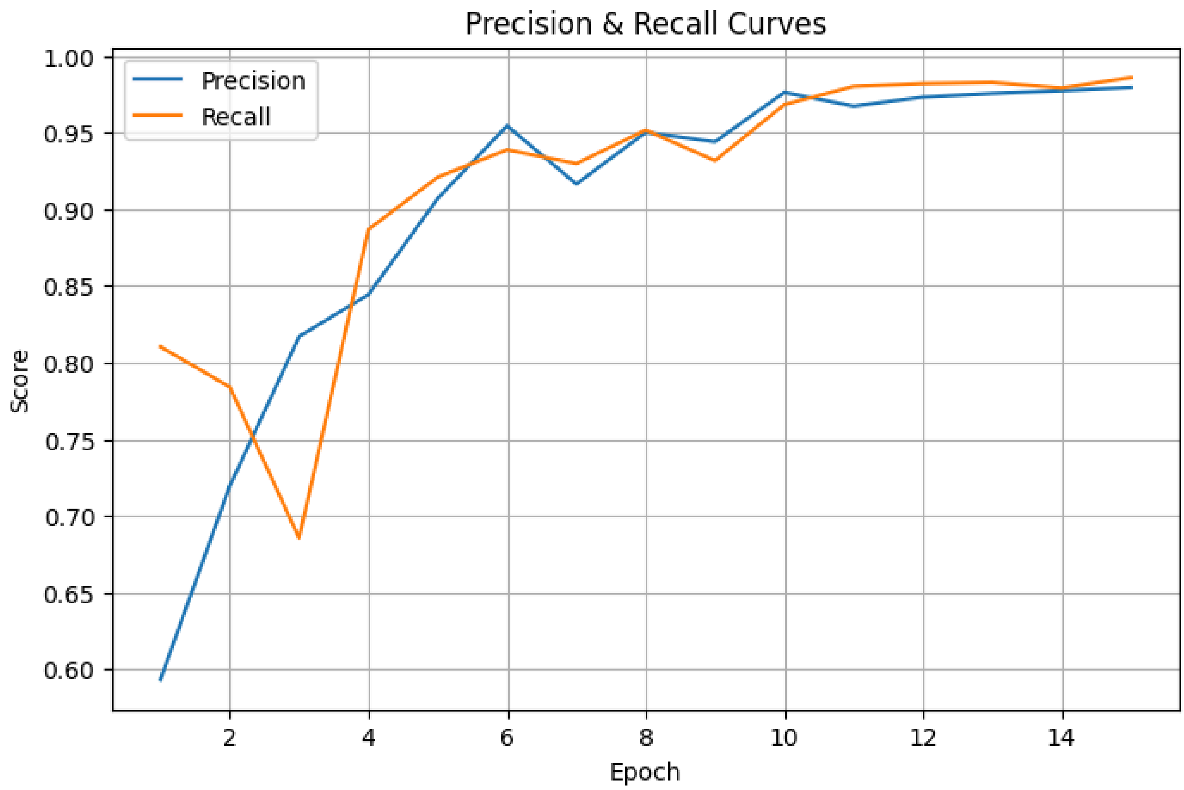
The comprehensive evaluation of model performance typically relies on visual graphs showing training stability and class-wise performance:

- **Training and Validation Graphs:**
  - Plotting training and validation loss over 20 epochs is essential to confirm that the model converged and did not overfit the training data. * **Confusion Matrix:**
  - This matrix is required to understand the model's performance on each specific defect class, identifying any classes that may be confused with one another (e.g., distinguishing a 'spur' from 'spurious copper').


Confusion Matrix
mouse_bite
spur
missing_hole
short
open_circuit
spurious_copper
background
Predicted
True
278
1
4
258
8
228
2
318
11
255
6
236
10
2
4
1
9
3
2
300
250
200
150
100
50
0


Training Loss Curves
Box Loss
Class Loss
DFL Loss
Loss
Epoch
2.5
2.0
1.5
1.0
2
4
6
8
10
12
14


mAP Curves
mAP50
mAP50-95
1.0
0.9
0.8
0.7
0.6
0.5
0.4
mAP
2
4
6
8
10
12
14
Epoch


Precision & Recall Curves
Precision
Recall
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
Score
2
4
6
8
10
12
14
Epoch

4. Challenges Faced and Solutions Implemented

This project encountered several challenges inherent to high-precision machine vision in manufacturing:

Challenge	Solution Implemented / Approach
Sub-millimeter/Minute Defects	Selection of **YOLOv11-M** over lighter variants and training at 640 * 640 resolution to enhance feature extraction for objects occupying less than 2% of the image area.
Low-Contrast Signatures	Defects often have low contrast, making them hard to distinguish from normal board texture. The depth and complexity of the YOLO network backbone are leveraged to automatically learn these subtle, fine-grained features.
Data Imbalance	Real-world data often has many more 'GOOD' images than defect images. Data augmentation (such as rotation, scaling, and brightness adjustments) was employed during training to artificially increase the effective size and diversity of the defect samples, improving generalization.
Real-time Requirement	Industrial lines demand fast processing. Using a single-stage detector like YOLO is crucial, as it maintains high throughput (FPS) while traditional two-stage models are too slow.

5. Potential Applications and Industry Impact

The successful development and validation of this YOLOv11-based PCB defect detection model demonstrate significant potential for both direct application and broad industry impact.

5.1. Potential Applications of Your Model

This type of object detection model is highly transferable to other quality control and inspection tasks beyond PCBs:

- **Semiconductor Wafer Inspection:** Detecting micro-scratches, particle contamination, or misaligned features on semiconductor chips.
- **Assembly Quality Check:** Verifying the presence, correct placement, and orientation of components (e.g., resistors, ICs) on a final circuit board (Component-level AOI).
- **Surface Defect Detection:** General inspection of manufactured goods, such as finding scratches, dents, or paint flaws on automotive parts or electronic casings.
- **Food and Drug Inspection:** Automated identification of foreign objects, packaging defects, or product inconsistencies.

5.2. Industry Impact

The integration of this deep learning solution can fundamentally change the economics and reliability of PCB manufacturing:

1. **Reduced Operational Costs:** By replacing slow, tedious, and error-prone manual inspection, the model significantly reduces labor costs.
2. **Enhanced Quality Standards:** The model provides objective, consistent, and superior accuracy (Precision 0.9780) compared to human inspectors, leading to fewer faulty products reaching the market.
3. **Real-Time Production Feedback:** The YOLO framework allows for inference speeds that meet stringent throughput requirements. This enables real-time, in-line quality checks, allowing manufacturers to identify and fix production line issues instantly, rather than waiting for batch inspections.
4. **Enabling Automation (Industry 4.0):** The model can be deployed on edge devices (low-latency IoT systems) to provide localized, instant defect identification, which is a key component of fully automated, 'smart' factory environments.

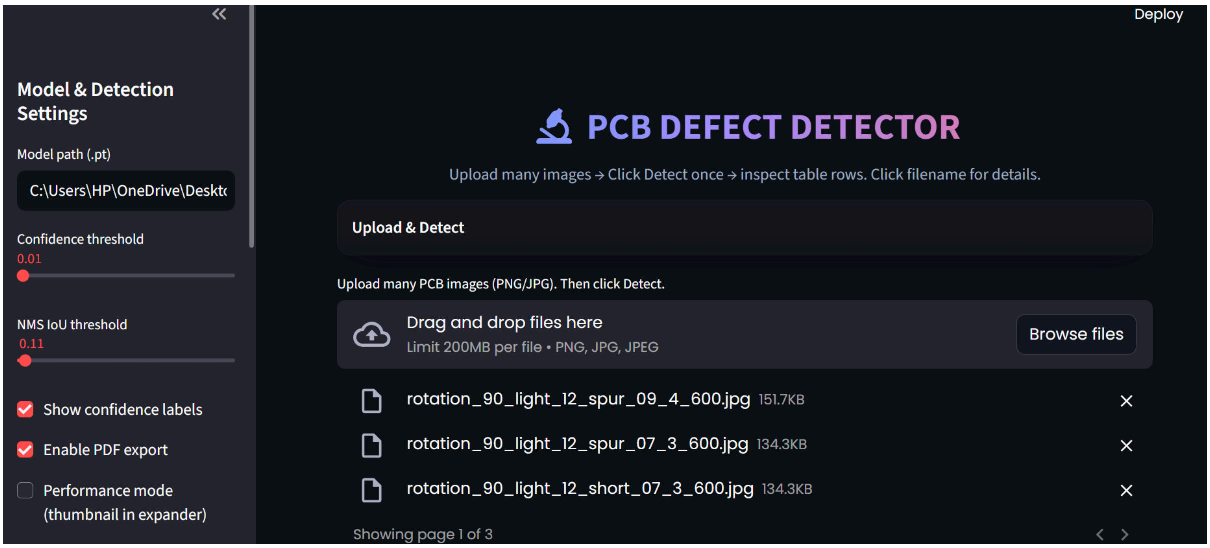
6. Deployment and User Interface (Streamlit)

The final stage of the project involved transitioning the trained YOLOv11-M model from a proof-of-concept script into a functional, user-friendly industrial tool using the **Streamlit** Python framework.

6.1. UI Technical Stack and Purpose

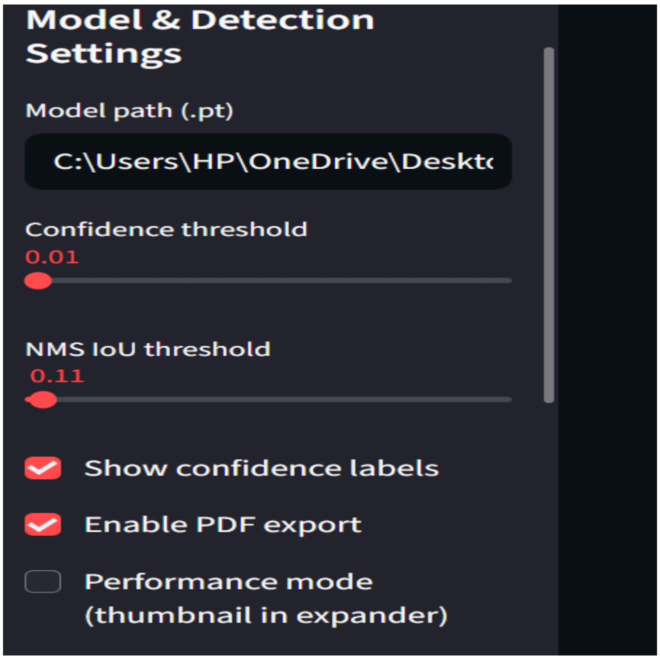
Component	Technology	Rationale
Web Interface	Streamlit	Rapid development of a data-centric, interactive web application using pure Python.
Model Backend	Ultralytics YOLOv11	Integrated directly for efficient inference and image plotting.
Report Generation	fpdf, zipfile, pandas	Enable the export of multi-format, consolidated reports.

The Streamlit application transforms a raw image detection process into an accessible quality control station.



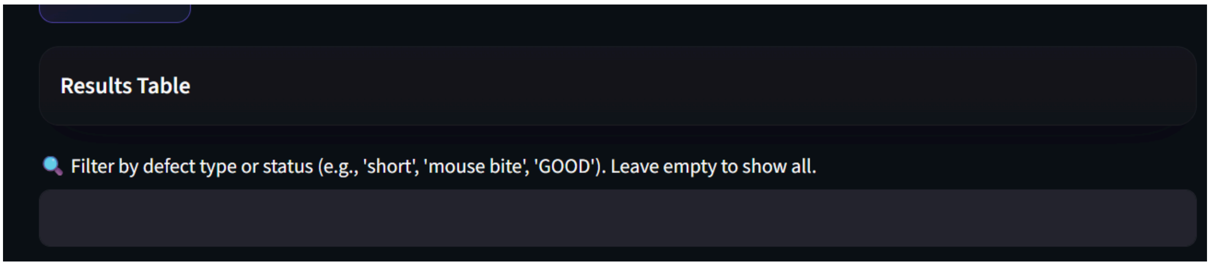
6.2. Key Application Features

1. **Bulk Upload & Detection:** Users can upload multiple PCB images simultaneously and run the entire detection process with a single "Detect" button click, making batch processing simple.
2. **Configurable Parameters:** A dedicated **Sidebar** allows quality engineers to adjust crucial detection hyperparameters, namely **Confidence Threshold** and **IoU (NMS) Threshold**, providing flexibility to tune the system for specific manufacturing conditions.

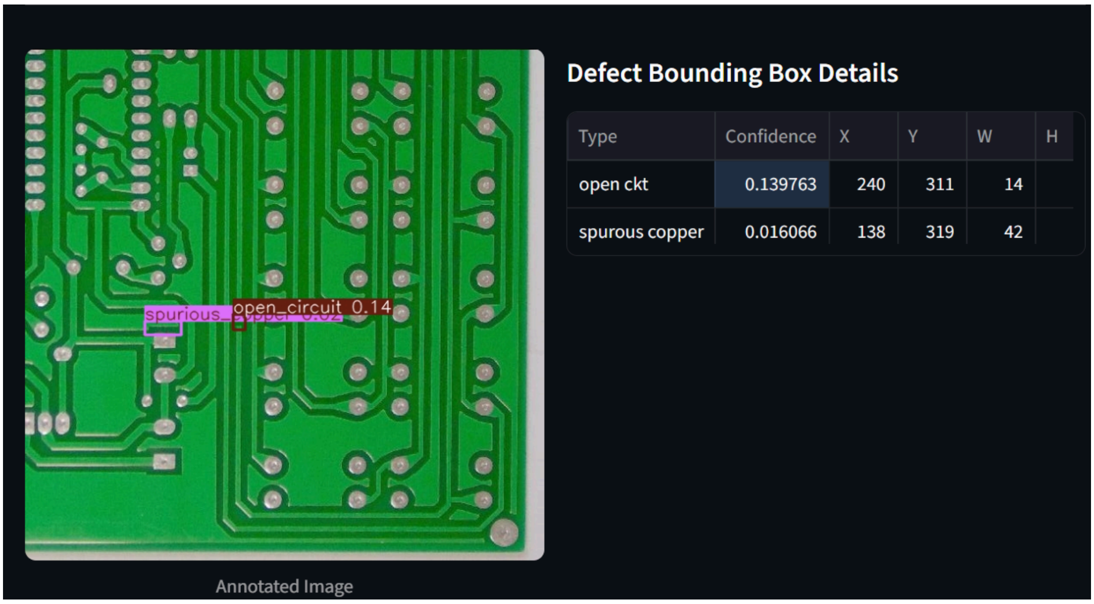


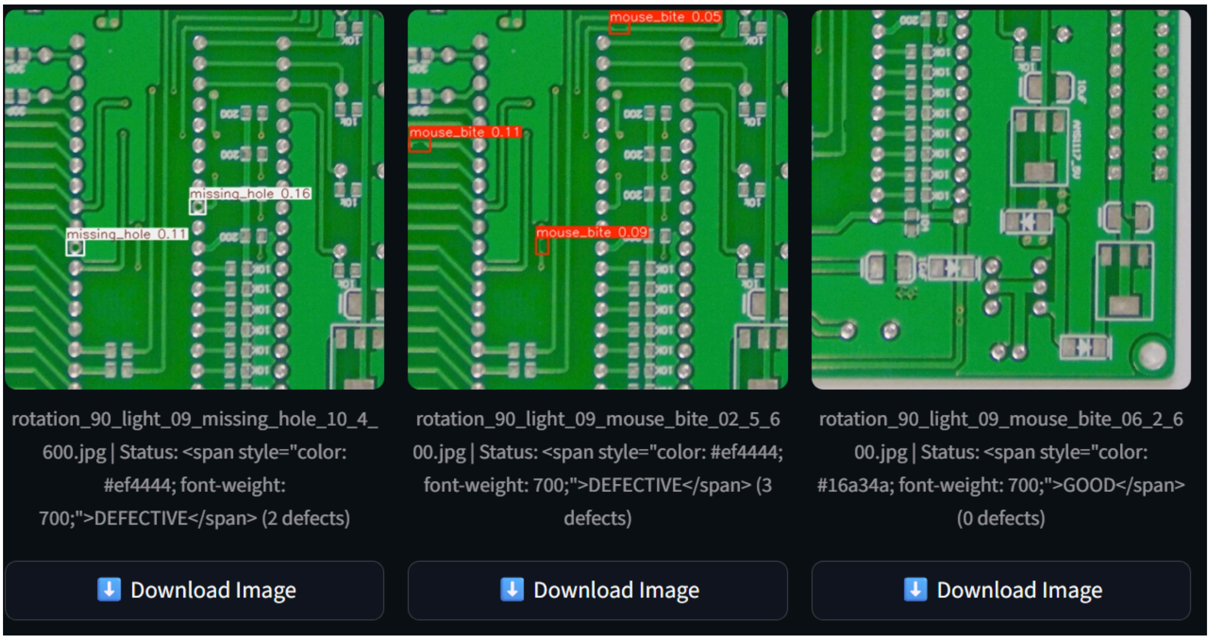
3. **Interactive Results Table:** This is the core output interface, displaying:
    - File Name(Clickable), no of defects and download option.

4. **Defect Search and Filtering:** A dedicated **Search Box** allows users to filter the table in real-time by defect type (e.g., "short") or by status ("GOOD" / "DEFECTIVE"), dramatically speeding up the inspection review process.

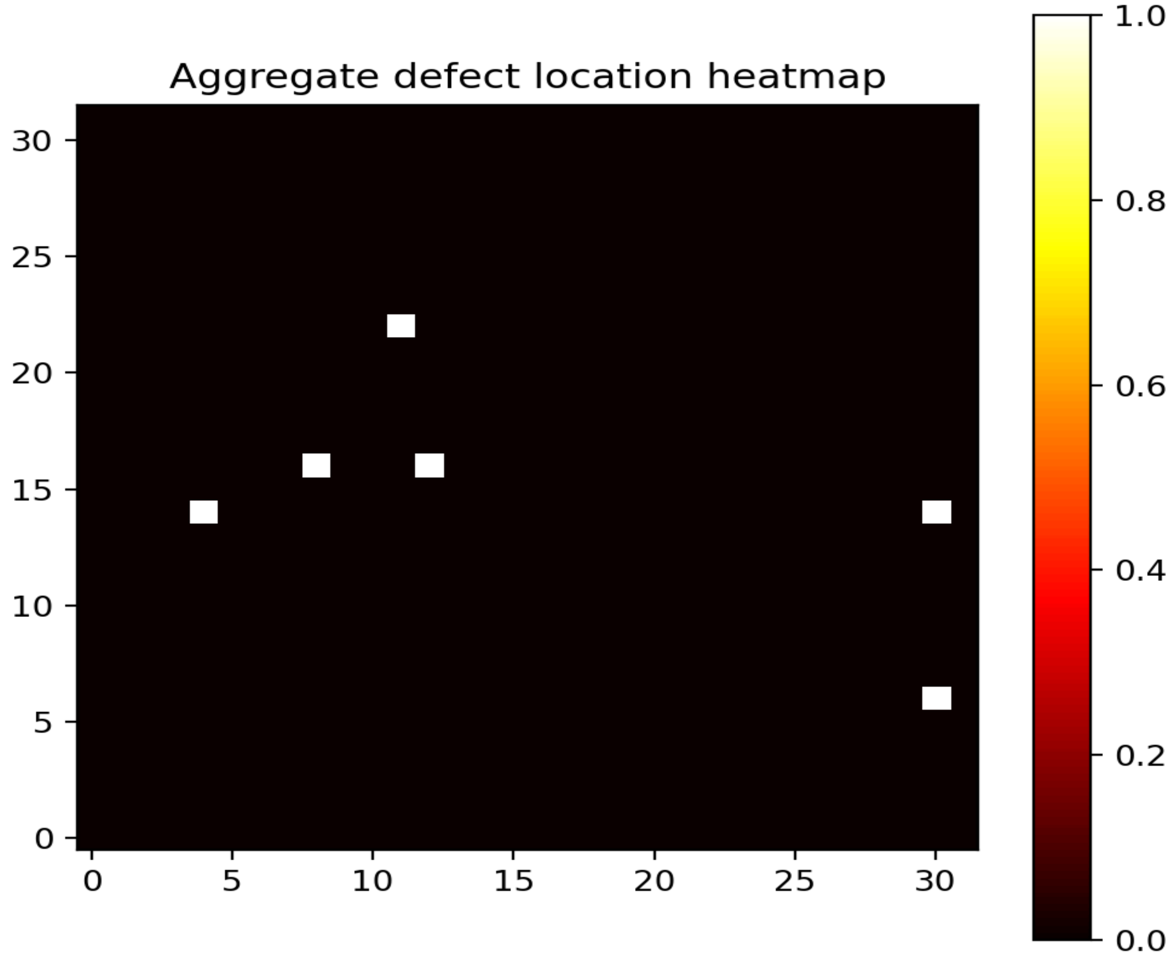


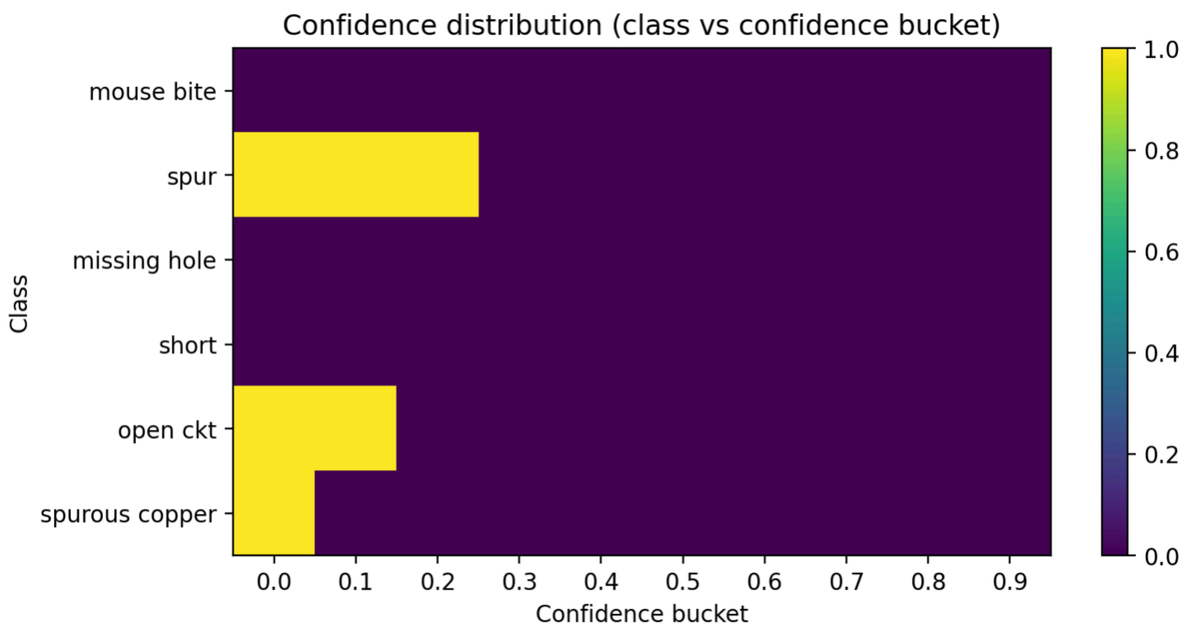
5. **Interactive Inspection View:** Clicking a file name expands a detail view showing:
    - The **Annotated Image** (with bounding boxes and defect labels).
    - A **DataFrame** detailing the coordinates, confidence, and class of every detected bounding box.





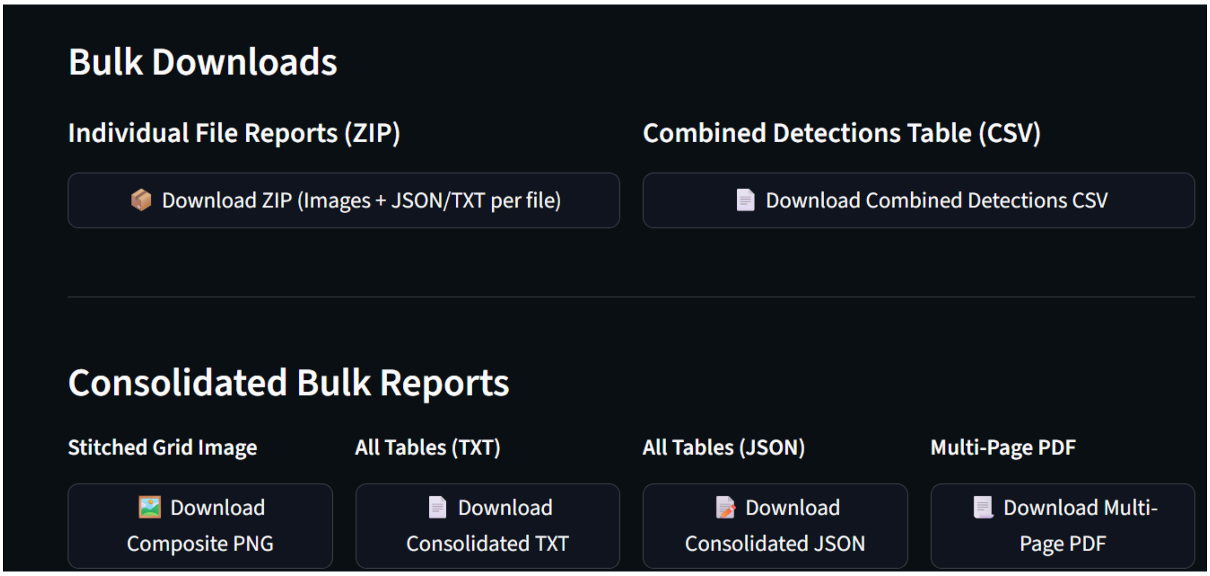
6. **Integrated Analytics (Heatmaps):** The application automatically generates visualizations from the batch results, including:
   - **Class Frequency Chart:** Bar chart showing the total count of each defect type.
   - **Coordinate Density Heatmap:** An aggregated heatmap showing common defect locations across all inspected boards, useful for process engineering feedback.

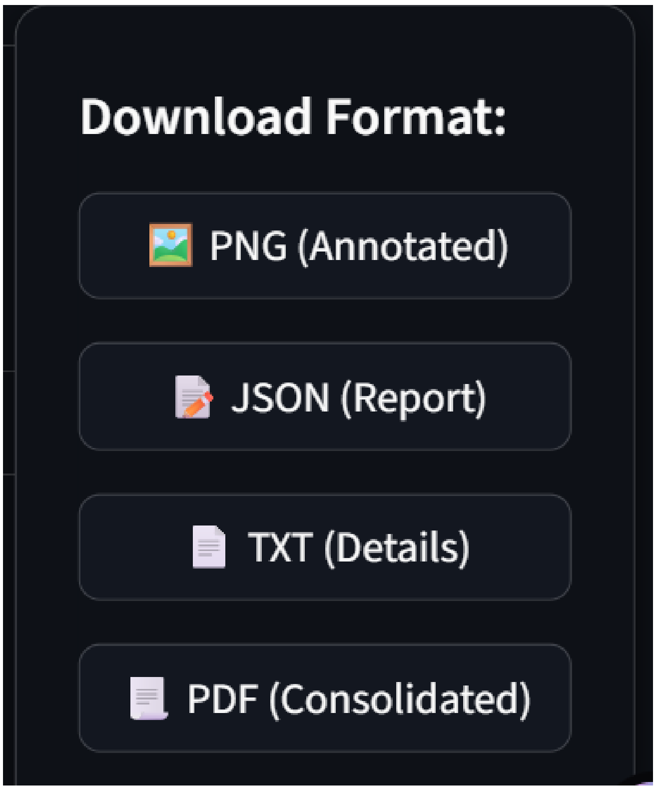

Aggregate defect location heatmap
30
25
20
15
10
5
0
0
5
10
15
20
25
30
1.0
0.8
0.6
0.4
0.2
0.0


Confidence distribution (class vs confidence bucket)
Class
mouse bite
spur
missing hole
short
open ckt
spurous copper
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Confidence bucket
1.0
0.8
0.6
0.4
0.2
0.0

7. **Data Export:** Comprehensive download options are available per-row and in bulk for full traceability:
    - Annotated PNG image.
    - JSON/TXT detailed report.
    - Consolidated CSV of all detections.
    - Consolidated ZIP file containing all reports and images.
    - Consolidated PDF Report for documentation.

This user interface makes the power of the YOLOv11-M model accessible to non-data scientists, ensuring that the technology is practical for immediate industrial adoption.





7. Backend API Implementation (FastAPI)

To complement the Streamlit user interface, a high-performance FastAPI backend was implemented. This allows the PCB Defect Detection system to be integrated into broader industrial automation pipelines where programmatic access—rather than manual UI interaction—is required.

7.1. Backend Architecture and Design

The backend is designed as a RESTful service that serves as a bridge between raw hardware (cameras) and the YOLOv11-M model.

- Asynchronous Framework: FastAPI was selected for its native support for asynchronous requests, ensuring the server can handle multiple inspection triggers concurrently without blocking.
- Static Asset Management: The server utilizes a mounted static directory to store and serve annotated result images, facilitating immediate visual verification via a web URL.
- Performance Monitoring: The backend includes precise millisecond-level timing to track inference speed, which is critical for maintaining high throughput on production lines.

7.2. Core Features of the API

1. Automated Environment Setup: On startup, the system automatically checks for and creates the necessary directory structure (`static/results`) to ensure data persistence for detected images.
2. Rich JSON Data Model: Unlike simple classification, the API returns a comprehensive JSON object containing:
   - Detection confidence (formatted as a percentage).
   - Precise bounding box coordinates (`xyxy` format).
   - Calculated inference speed.
   - A direct URL to the stored annotated image for auditability.
3. Model Integration: The API loads the optimized `best.pt` weights once into memory, reducing the overhead for subsequent detection requests.

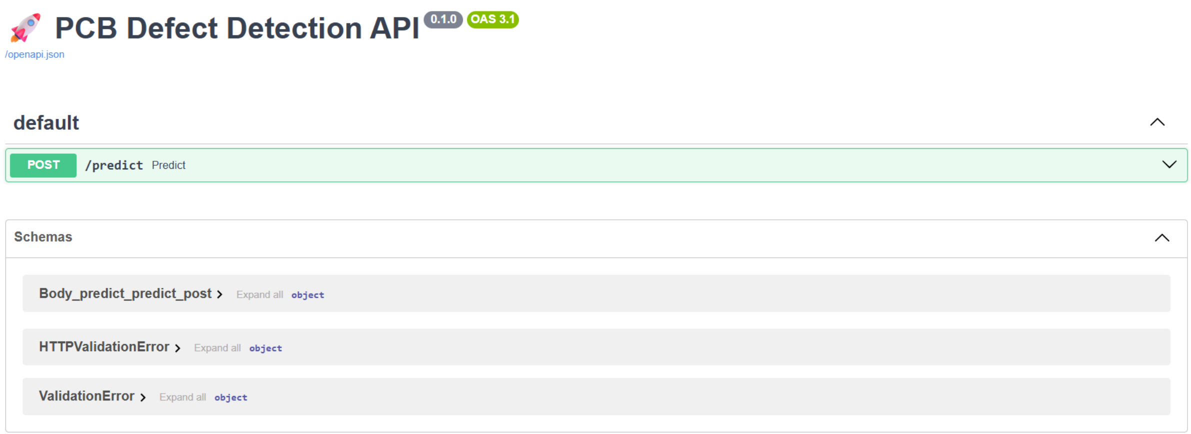
7.3. API Implementation Details

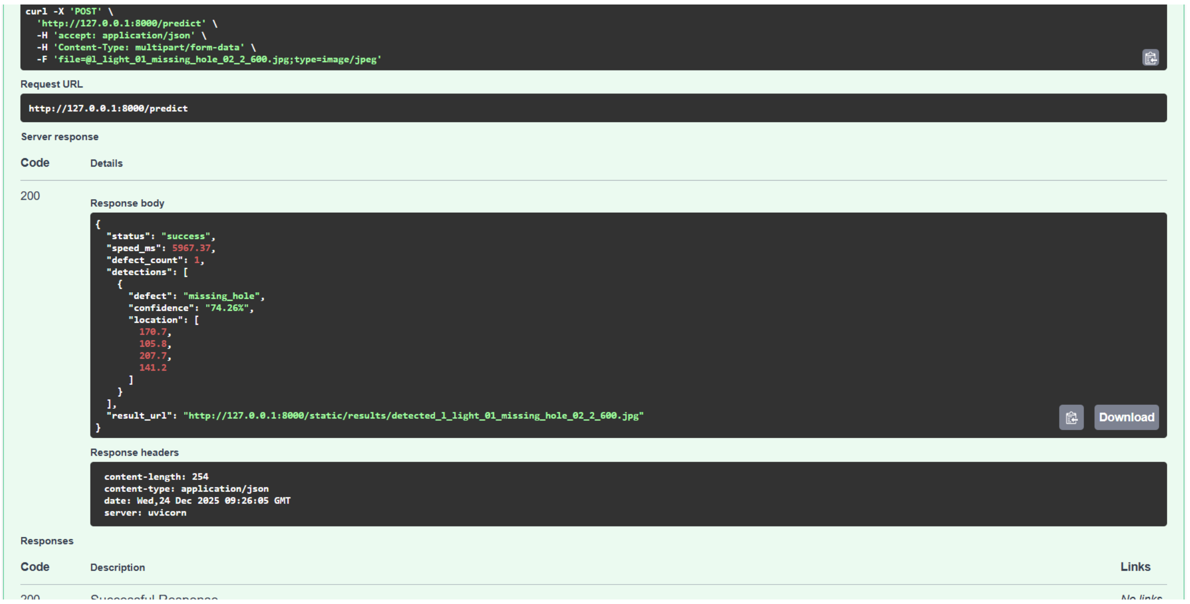
The following code snippet illustrates the implementation of the `/predict` endpoint, which handles image uploads, model inference, and result serialization:

```python
RESULT_DIR = "static/results"
os.makedirs(RESULT_DIR, exist_ok=True)

# 2. MOUNT: This allows you to view images at http://127.0.0.1:8000/static/...
app.mount("/static", StaticFiles(directory="static"), name="static")

# Load the brain
model = YOLO("models/best.pt")

@app.post("/predict")
async def predict(file: UploadFile = File(...)):
    # Start the clock ⏰
    start_time = time.perf_counter()

    # Read image
    contents = await file.read()
    image = Image.open(io.BytesIO(contents)).convert("RGB")

    # Run AI Inference
    results = model.predict(source=image, conf=0.25)

    # 3. COOL FEATURE: Save the image with boxes drawn on it
    # This saves the 'evidence' to your local folder automatically
    res = results[0]
    output_filename = f"detected_{file.filename}"
    output_path = os.path.join(RESULT_DIR, output_filename)
    res.save(filename=output_path)
```

ROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
venv) PS C:\Users\HP\OneDrive\Desktop\pcb_backend> uvicorn main:app --reload
NFO:     Application startup complete.
NFO:     127.0.0.1:55425 - "GET / HTTP/1.1" 404 Not Found
NFO:     127.0.0.1:55425 - "GET /favicon.ico HTTP/1.1" 404 Not Found
NFO:     127.0.0.1:55425 - "GET /docs HTTP/1.1" 200 OK
NFO:     127.0.0.1:55425 - "GET /openapi.json HTTP/1.1" 200 OK
```





.

8. Conclusion

The project successfully implemented and trained a **YOLOv11-M** model for the automated detection and classification of six critical PCB defects. The model achieved excellent performance metrics mAP(50) : 0.9863, F1-score: 0.9815, confirming its robust capability to detect small, complex defects accurately. By utilizing a modern, single-stage detection architecture, the system provides a robust, scalable, and real-time solution that directly addresses the limitations of traditional inspection methods, offering significant value to the electronics manufacturing industry.

9. Documentation and References

The entire process, including data conversion scripts, training setup, and the Streamlit application code, is fully documented. The final model weights (`best.pt`) and the Streamlit application serve as the deliverable system for real-time inspection.