**Project Plan: PCB Component Detection and Fault Identification Using YOLO and AI Models**

### ****Phase 1: Dataset Collection & Preparation****

#### Step 1: Explore and Download Public Datasets

Here are key sources to collect datasets for PCB components and defect detection:

| **Platform** | **Dataset / URL** | **Content Description** |
| --- | --- | --- |
| Kaggle | <https://www.kaggle.com/datasets/shubhendra7/pcb-defect-dataset> | Common defects like mouse bite, open circuit, etc. |
| Roboflow | <https://universe.roboflow.com/pcb-defects> | Annotated YOLO-ready defect and component datasets |
| GitHub | <https://github.com/luoyetx/deep-pcb> | Benchmark dataset for PCB defect detection |
| GitHub | <https://github.com/FICS/PCB-dataset> | Real-world PCB dataset with component annotations |
| Hugging Face | <https://huggingface.co/datasets?search=pcb> | Search for PCB datasets compatible with YOLO |

#### Step 2: Dataset Preparation

1. Choose a dataset from the list above or collect your own PCB images using high-resolution cameras.
2. Annotate the images using tools such as:
   * Roboflow (online tool with export options)
   * LabelImg (offline annotation tool)
   * CVAT (for collaborative labeling)
3. Define classes (e.g., missing\_hole, mouse\_bite, etc.) consistently.
4. Export the dataset in YOLO format (YOLOv5/v8 format):

project\_root/

datasets/

dataset\_name\_1/

images/

train/

val/

test/

labels/

train/

val/

test/

config.yaml

dataset\_name\_2/

images/

...

labels/

...

config.yaml

models/

yolov8n.pt

yolov8s.pt

outputs/

dataset\_name\_1/

dataset\_name\_2/

train.py

inference.py

gui\_app/

streamlit\_app.py

utils.py

1. Each dataset folder should contain a config.yaml like this:

task: detect

train: datasets/dataset\_name/images/train

val: datasets/dataset\_name/images/val

nc: 6

names: ["missing\_hole", "mouse\_bite", "open\_circuit", "short", "spur", "spurious\_copper"]

1. Modify train.py to accept this YAML path as an argument and train accordingly.
2. Prepare streamlit\_app.py to dynamically read available datasets and models, display selection menus, and run inference using uploaded PCB images or a live webcam feed.

### ****Streamlit Interface Design (streamlit\_app.py)****

#### Streamlit App Workflow

1. **Sidebar Configuration**
   * Dropdown to select available dataset (auto-discovered from datasets/ directory)
   * Dropdown to select YOLO model weights from outputs/ or models/
2. **Upload Inference Image**
   * Drag-and-drop or browse to upload a PCB image
   * Automatically run inference on the uploaded image
3. **View Results**
   * Show bounding boxes with class labels and confidence scores
   * Optional toggle to display annotations as labels or bounding box only
4. **Live Detection Mode (Optional)**
   * Button to enable webcam or USB camera stream
   * Real-time frame capture, inference, and result overlay
5. **Advanced Options (Expandable Panel)**
   * Confidence threshold slider
   * Class filtering (show only certain defect types)
   * Batch process multiple uploaded images

#### Code Structure Example

gui\_app/

streamlit\_app.py # Main Streamlit app

utils.py # Helper functions: load\_models, get\_datasets, draw\_boxes

assets/ # Logos, icons, and theme assets

templates/ # HTML or Jinja for formatting if needed

This app will make model testing and deployment interactive and user-friendly for quality engineers and developers alike.

### ****Phase 2: Model Training****

#### Step 3: Choose a YOLO Version

* Install the Ultralytics YOLO library:

pip install ultralytics

* Download pretrained model weights from <https://github.com/ultralytics/yolov5> or <https://github.com/ultralytics/ultralytics>

#### Step 4: Training Configuration (Fine Steps)

1. Open terminal or script editor.
2. Ensure you have required folders and YAML config ready.
3. Run training:

yolo task=detect mode=train model=models/yolov8n.pt data=datasets/dataset\_name/config.yaml epochs=50 imgsz=640

1. Monitor logs and loss values.
2. Pause/Resume training if needed using checkpoint files.
3. Training output will be saved in runs/detect/train/ by default or can be redirected to outputs/.

### ****Phase 3: Evaluation and Analysis****

#### Step 5: Model Evaluation (Fine Steps)

1. Run evaluation with trained weights:

yolo task=detect mode=val model=outputs/dataset\_name/weights/best.pt data=datasets/dataset\_name/config.yaml

1. Check mAP, precision, recall, and F1 score.
2. Generate confusion matrix and visual plots.
3. Record results per class and overall.

#### Step 6: Perform Error Analysis

1. Manually review predictions with false positives/negatives.
2. Use Roboflow or CVAT to visualize predictions.
3. Identify problematic cases (e.g., occluded or rotated components).
4. Retrain or fine-tune with hard examples.

### ****Phase 4: Inference and Integration****

#### Step 7: Batch and Real-Time Inference

1. Load model in Python:

from ultralytics import YOLO

model = YOLO('outputs/dataset\_name/weights/best.pt')

1. Run on images:

results = model('sample.jpg')

results.show()

1. Integrate into GUI or web app:
   * PyQt5 for desktop
   * Streamlit for web dashboard
   * Use OpenCV for frame-by-frame camera inference

### ****Phase 5: Deployment and Documentation****

#### Step 8: Model Optimization and Deployment

1. Convert model to ONNX:

yolo export model=best.pt format=onnx

1. For Jetson/Raspberry Pi:
   * Optimize with TensorRT
   * Reduce model size or use yolov8n for speed
2. Connect to USB microscope/camera for real-time analysis

#### Step 9: Reporting and Publication

1. Document key results, charts, performance metrics
2. Create before/after inspection visuals
3. Highlight use-case in satellite sensor board inspection
4. Compare against manual inspection time and error rates

### ****Optional Enhancements****

#### 1. ****Segmentation-Based Analysis****

* Use **YOLOv8-seg** or **SAM (Segment Anything)** for mask-based output.

#### 2. ****Component Classification****

* Extract detected component images
* Feed into a classifier (e.g., EfficientNet) for label classification

#### 3. ****Multi-Task Learning****

* Custom training combining detection and classification in a single network

#### 4. ****Anomaly Detection Using Autoencoders****

* Train autoencoders on good PCBs
* High reconstruction error implies potential fault

#### 5. ****3D PCB and Multilayer Analysis****

* Explore datasets with X-ray/3D imaging for deeper layer inspection

#### 6. ****Explainability and AI Insight****

* Use Grad-CAM to visualize model’s focus area
* Improve user trust and validation in critical use-cases

This roadmap provides a robust, detailed, and scalable plan to implement AI-powered PCB inspection from scratch to deployment and beyond.

the above exact details in the same format including font size style to saved into document and make it donwloadble dont miss any content