Home Defect Detection: Multiâ Step Plan

1) Define Labels And Data Schema

- Taxonomy: split into two lists you control.
  - Object types: wall, window, door, roof, pipe, ceiling, floor, outlet, sink, tile.
  - Defect types: crack, leak, mold, rust, dent, scratch, paint_peel, stain.
- Encoding strategy (MVP): combined classes like window_crack, wall_mold, pipe_leak.
- Keep combinations small initially (20â 60), expand later.

2) Set Up Labeling Workflow

- Keep near/far pairs per case with consistent names (e.g., scene123_far.jpg, scene123_near.jpg).
- Annotate defect regions with bounding boxes and assign combined class.
- Use CVAT/Label Studio; export COCO detection format.

3) Prepare Dataset In COCO Format

- Structure: data/home/train, data/home/val, and COCO JSON in data/home/annotations.
- Dataset YAML: yolo/config/dataset/home.yaml with class_list and class_num.
- Ensure COCO category names exactly match class_list entries.

4) Configure YOLO Model/Classes

- Start with v9-s for speed; v9-c/m for accuracy.
- dataset=home, set image_size (e.g., 640/768), verify class_num matches list length.

5) Run Baseline Training

- pip install -r requirements.txt
- Train: python yolo/lazy.py task=train dataset=home model=v9-s name=home-defect-v1 task.data.batch_size=16 device=cuda
- To train from scratch: add weight=False
- Outputs/checkpoints under runs/train/home-defect-v1

6) Evaluate And Refine Taxonomy

- Validate: python yolo/lazy.py task=validation dataset=home name=home-defect-v1-val device=cuda
- Review per-class AP; prune/merge confusing or rare classes; add data for weak classes.

7) Export Model (ONNX/TensorRT)

- For fast inference: task.fast_inference=onnx or trt with your checkpoint weight path.
- Produce reproducible artifacts for deployment.

8) Design AWS Inference Architecture

- Steady GPU throughput: ECS/EC2 GPU behind ALB; containerized FastAPI with Torch/ONNX/TRT.
- Bursty traffic: API Gateway -> Lambda with ONNX Runtime (CPU) or custom GPU hosts.
- Use S3 for uploads/results; presigned URLs for Android.

- Observability with CloudWatch logs/metrics; trace request IDs.

9) Implement API + Storage
- Endpoint POST /analyze accepts two images (multipart) or S3 URLs: image_far, image_near.
- Returns list of detections with bbox, class, score; also split object/defect fields.
- Secure with IAM for S3 access; use API keys/JWT as needed.

10) Android App Integration
- Upload both images via presigned S3 PUT URLs.
- Call /analyze with S3 URLs and a pair_id.
- Display overlays and labels; optionally show remediation steps.

11) Add Near/Far Fusion Logic
- MVP: detect on each image, union + NMS, keep max-confidence labels; retain uniques.
- Later: cross-image matching by IoU after resize or simple crop feature matching.
- Advanced (optional): train with paired inputs (code changes required).

12) Monitoring And Iteration
- Track latency, GPU utilization, error rates, class distribution, and drift.
- Human-in-the-loop: collect low-confidence cases for relabeling.
- Retrain on cadence (weekly/monthly) as data grows.

Common Commands
- Train: python yolo/lazy.py task=train dataset=home model=v9-s name=home-defect-v1 device=cuda task.data.batch_size=16
- Validate: python yolo/lazy.py task=validation dataset=home name=home-defect-v1-val device=cuda
- Inference: python yolo/lazy.py task=inference dataset=home model=v9-s weight=runs/train/home-defect-v1/checkpoints/last.ckpt task.data.source=path/to/images
- Fast: add task.fast_inference=onnx (or trt on GPU host)

Notes
- Ensure yolo/config/dataset/home.yaml has path/train/validation and exact class_list + class_num.
- Start with a small, high-quality label set and expand.