Designing VLM for Diagnosing Defective PCBs

(A) Model Selection

Selected Architecture: Qwen-VL-Chat (Int4 Quantized)

Main Reasons: While LLaVA is popular, **Qwen-VL** is superior for this specific industrial scenario due to three critical factors:

1. Unlike generic VLMs, Qwen-VL was pre-trained with bounding box coordinates input/output capabilities. It treats coordinates as special tokens, making it inherently better at localization/detection tasks than the other models.
2. PCBs contain minute defects (hairline fractures, solder bridges). Standard LLaVA resizes images to smaller resolution losing detail. Qwen-VL supports and higher natively, preserving small defect features.
3. The model performs exceptionally well even when quantized to 4-bit, fitting comfortably within the memory constraints of industrial edge computing devices such while meeting the latency target.

Other important factors to consider:

- **Resolution Sensitivity:** Critical for PCB features.
- **Inference Speed:** Must meet strict timings in a manufacturing pipeline.
- **Licensing:** Apache 2.0 (allows commercial use).

(B) Design Strategy & Architectural Modifications

To adapt the general-purpose Qwen-VL for specialized PCB inspection, we will implement a **Hybrid Vision-Language Architecture**.

1. **Vision Encoder Modification :**
    - Instead of standard resizing, we can implement a **Dynamic Resolution approach**. Large PCB images can be cropped into overlapping patches to form a sliding window like mechanism.

- We can add a lightweight **Coordinate Embedding Layer** to inform the LLM which patch it is looking at, preventing spatial confusion during patch aggregation.

2. **Language Decoder:**
    - The LLM backbone (Qwen-7B) remains, but we replace the standard tokenizer's vocabulary for numbers with dedicated bin tokens for coordinates (0-1000) to improve regression accuracy for bounding boxes.
3. **Structured Output Head:**
    - Generic VLMs output free text. So to tackle that we can append a **Grammar-Constrained Decoding** layer to force the model to strictly output valid JSON schema and prevent it from generating unnecessary text.
    -

(C) Optimization (<2s Latency)

To achieve the sub-2-second inference target on offline hardware, we employ a three-level optimization strategy.

1. **Quantization (4-bit GPTQ/AWQ):**
    - We will use **AWQ (Activation-aware Weight Quantization)** to compress the 7B model weights to 4-bit. This reduces VRAM usage from ~14GB to ~5GB and increases memory bandwidth throughput, directly speeding up token generation.
2. **FlashAttention-2:**
    - Implement FlashAttention-2 in the attention layers to reduce the quadratic complexity of the attention mechanism to near-linear, significantly speeding up the processing of high-resolution image tokens.
3. **TensorRT-LLM Deployment:**
    - Instead of raw PyTorch, the model will be compiled using **NVIDIA TensorRT-LLM**. This provides kernel fusion and optimized matrix multiplications. Which should further decrease inference timings.

(D) Hallucination Mitigation

We can employ the following approaches to mitigate hallucination:

1. **Constraint-Based Decoding:**
    - We restrict the output token space. If the model detects no defect, it is forced to output a specific "No Defect" token rather than generating extra unnecessary output.
2. **DPO (Direct Preference Optimization):**
    - After Supervised Fine-Tuning, we can perform a DPO stage.
    - **Dataset:** We construct "negative pairs" where the "rejected" response contains a hallucinated defect and the "chosen" response is the correct ground truth.
    - **Loss Function:** We employ the DPO(Direct Preference Optimization) loss. This will force the model to look at instances of positive behaviour and instances of hallucinations and then adjust the weights to push the probability of hallucinations lower and the probabilities of positive instances are pushed higher.
3. **Confidence Thresholding:**
    - The system will extract log-probabilities for the "defect class" tokens. If the confidence score is below a strict threshold, the system flags the board for **Human Review** rather than making an autonomous decision.

(E) Training Plan

Since no QA pairs exist, we must synthetically generate them from the 50,000 images + Bounding Boxes (BBox).

Phase 1: Synthetic Data Generation

- Convert annotations into QA pairs using templates.

    - *Input:* Image + BBox [x1, y1, x2, y2] + Label "Short Circuit".
    - *Generated Question:* "Locate the short circuit on this PCB."
    - *Generated Answer:* "Found short circuit at [x1, y1, x2, y2] with z% confidence."
- **Negative Sampling:** Create 20,000 QA pairs where the question asks about a defect that does not exist, with the target answer "No [defect_name] detected." This is crucial for reducing false positives.

Phase 2: Vision-Language Alignment (LoRA Fine-tuning)

- **Technique:** QLoRA (Quantized Low-Rank Adaptation).
- **Trainable Parameters:** We freeze the ViT and the core LLM. We only train the **Projector** and **LoRA adapters** attached to the LLM's attention layers.
- **Augmentation:** Mosaic augmentation (mixing multiple PCB images) to teach the model to handle clutter and scale variations.

Phase 3: Hard Negative Mining

- Evaluate the model on a validation set. Collect instances where the model hallucinated. Add these back into the training set with corrected labels for another small epoch of fine-tuning.

(F) Validation Strategy

We will validate the system using metrics specifically designed for object detection and text generation overlap.

1. **Localization Precision (mAP@50):**
    - We extract the coordinates from the VLM's text output and calculate the **Mean Average Precision (mAP)** at IoU threshold 0.5 against the ground truth bounding boxes.
2. **Counting Accuracy (RMSE):**
    - For quantitative questions like "How many resistors are missing?", we calculate the Root Mean Square Error between the predicted count and actual count.

3. **Object Hallucination Rate (OHR):**
    - We prompt the model with questions about defects that are *not* present.
    - OHR = (Number of hallucinatory responses/Total queries about absent objects)
4. **Output Structural Integrity**
    - Percentage of outputs that are valid, readble JSONs or other desirable files. Should be 100% so that all outputs are valid.