Design Proposal: Custom VLM for Semiconductor PCB Quality Inspection

1. Executive Summary

This document outlines the architectural and strategic design for an offline Vision-Language Model (VLM) based PCB inspection system. The goal is to provide inspectors with a natural language interface for defect detection with structured output (location, confidence) under a 2-second inference constraint. This solution addresses the specific challenges of tiny defect localization, strict latency requirements, and the prevention of visual hallucinations in high-stakes semiconductor manufacturing.

(A) Model Selection

Chosen Base: Qwen2-VL-2B (or similar Small Language Model-based VLM)

Why Qwen2-VL-2B?

- **Model Size & Speed:** With 2 billion parameters, the model fits into the VRAM of industrial edge devices (e.g., NVIDIA Jetson AGX Orin or RTX 4000 series). It allows for 4-bit quantization without losing the linguistic nuance required to understand varied inspector queries.
- **Native Resolution Handling:** Qwen2-VL utilizes **Naive Dynamic Resolution**. Traditional VLMs resize images to a fixed square (e.g., 224x224), which would effectively delete tiny solder defects. This architecture processes the PCB image at its native aspect ratio using a variable number of visual tokens.
- **Architectural Flexibility:** The Qwen2 backbone is highly optimized for tool-calling and structured outputs (JSON), which is necessary for integrating the AI with downstream factory management systems.

Architectural Modifications for Localization

1. **Coordinate Tokenization:** We expand the vocabulary with bins representing normalized coordinates. By treating coordinates as specific tokens (e.g., <bin_452>), the LLM learns spatial geometry as a language, allowing it to "speak" bounding boxes.
2. **Feature Upsampling (Detection Neck):** We insert a lightweight **Feature Pyramid Network (FPN)** between the ViT and the LLM. This allows the model to access low-level, high-resolution features (edges, textures) that are often lost in the deeper, more semantic layers of a standard Vision Transformer.

(B) Design Strategy

1. Vision Encoder (Backbone)

- **Modification:** We will swap the generic CLIP-ViT for a **SigLIP-SO400M** encoder.
- **Reasoning:** SigLIP uses a sigmoid loss that is more effective at identifying rare, fine-grained features (like a 10-micron crack) compared to the contrastive loss of standard CLIP, which is optimized for general scene understanding.

2. Language Decoder (LLM)

- **Modification:** We employ **QLoRA (Quantized Low-Rank Adaptation)**. We freeze the base model and train small adapter matrices. Specifically, we target the cross_attention layers, as these are the "bridge" where the model learns to associate the word "corrosion" with the specific copper-colored pixels in the image.

3. Fusion Mechanism: The C-Abstractor

- **Modification:** Instead of a simple linear layer (which creates a 1:1 mapping of visual patches to tokens), we use a **Cross-modal Abstractor**.
- **Logic:** It uses a set of learnable "query tokens" that attend to the visual features. This compresses a 4K image into ~512 high-density tokens, significantly reducing the computational load on the LLM's self-attention mechanism, which is the primary bottleneck for inference speed.

(C) Optimization for <2s Inference

1. **AWQ (Activation-aware Weight Quantization):** We apply 4-bit quantization. Unlike standard round-to-nearest methods, AWQ protects the 1% of "salient" weights that contribute most to accuracy, ensuring the model doesn't lose its ability to distinguish between "dust" and "solder ball."
2. **KV Cache Paging (vLLM style):** By managing memory in blocks rather than contiguous chunks, we prevent memory fragmentation, allowing the model to handle multi-turn dialogues with the inspector without slowing down.
3. **Speculative Decoding:** We can use a tiny "draft" model (e.g., a 100M parameter CNN) to predict the likely location tokens, which the 2B VLM then verifies in parallel.
4. **Operator Fusion:** Using TensorRT, we fuse the Vision Encoder and the Projection layers into a single CUDA kernel to minimize the overhead of moving data between the GPU's memory and its cores.

(D) Hallucination Mitigation

1. Training on "Hard Negatives"

We include "Golden Boards" (perfect PCBs) in the training set. If the inspector asks "Where is the short circuit?", the model is trained with a high penalty to respond with a specific "NULL" token rather than hallucinating the most likely-looking shadow as a defect.

2. Constrained Beam Search

During inference, we apply a "JSON-schema mask." The model is physically prevented from sampling tokens that don't fit the {"bbox": [x,y,x,y]} format.

3. Visual Grounding Loss (Contrastive)

We add a secondary loss term during fine-tuning:

$$L_{grounding} = 1 - IoU(AttentionMap_{keyword}, GroundTruth_{bbox})$$

This forces the model's internal "attention" to physically align with the bounding boxes provided in the 50,000-image dataset.

(E) Training Plan (Multi-Stage)

Stage 1: Self-Supervised QA Generation

Since the dataset lacks text, we use a "Teacher-Student" pipeline:

1. Feed the 50k images + boxes into an offline **GPT-4o** or **Qwen2-VL-72B**.
2. Generate 5-10 question-answer pairs per image (e.g., "Is the polarity marker correct on C5?", "Describe the defect in the top-left quadrant.").
3. This creates a 500,000-sample "PCB-Speech" dataset.

Stage 2: Alignment (Visual-Spatial)

- **Focus:** Training the <box> tokens.
- **Task:** The model is given an image and a box, and must predict the defect name; then given a name, it must predict the box. This builds a bidirectional mapping between pixels and labels.

Stage 3: Domain-Specific IFT (Instruction Fine-Tuning)

- **Data Mix:** 80% PCB QA pairs, 10% general reasoning (to prevent "catastrophic forgetting"), and 10% "Refusal" samples (where the model must say "I don't know" or "No defect found").

(F) Validation & KPIs

1. **Localization Precision:** We measure **mAP (mean Average Precision)** at IoU 0.5. For industrial use, we require mAP > 0.92.
2. **Hallucination Rate (FPR):** In 1,000 tests on clean boards, the model must achieve a **False Positive Rate of <0.1%**.
3. **Linguistic Robustness:** We test using "Adversarial Queries" (e.g., "Find the bug" vs "Locate the defect" vs "Is there an error here?"). The structured output must remain consistent regardless of phrasing.
4. **Hardware Benchmarking:** * *Target:* <1.8s per query on NVIDIA Orin (30W mode).
    - *Memory:* Total footprint < 4GB VRAM.