Case Study: Multimodal Product Defect Detection at Stratos Manufacturing

Section 1: Industry Context and Business Problem

Industry: Advanced Manufacturing -- Semiconductor Fabrication

Company Profile: Stratos Manufacturing

Stratos Manufacturing is a mid-tier semiconductor fabrication company operating three fabs across the United States. With \$2.8B in annual revenue and 4,200 employees, Stratos specializes in producing application-specific integrated circuits (ASICs) for automotive and industrial customers. Their product line includes power management ICs, sensor interface chips, and motor control ASICs -- components where reliability is non-negotiable because they go into safety-critical automotive systems.

Business Challenge: The Defect Detection Bottleneck

Semiconductor manufacturing involves hundreds of process steps, and each wafer undergoes multiple inspection stages. Currently, Stratos employs a hybrid quality assurance pipeline:

1. **Automated Optical Inspection (AOI):** High-resolution microscope images of each die are captured at multiple process stages. An existing CNN-based classifier flags potential defects with 89% accuracy.
2. **Process Logs:** Each wafer carries a digital passport of process parameters -- temperature profiles, etch durations, chemical concentrations, chamber pressure readings -- stored as structured tabular data.
3. **Operator Notes:** Human operators record free-text observations during production: "slight discoloration on edge of wafer," "vibration spike during CMP step," "recipe adjusted for humidity."

The problem is that these three information sources are analyzed independently. The AOI system only sees images. The process monitoring system only sees numerical logs. The operator notes sit in a text database that nobody mines systematically.

This fragmentation costs Stratos \$47M annually in three ways:

- **False positives (32% of AOI flags):** The vision-only classifier flags cosmetic variations that are not actual defects, sending good wafers for expensive manual re-inspection.
- **Missed defects (escape rate of 3.2%):** Some defects are only detectable when image anomalies are correlated with process deviations recorded in the logs or operator notes. The vision-only system misses these.

- **Slow root cause analysis:** When a defect batch is detected, engineers spend an average of 18 hours manually correlating images, process data, and operator notes to identify the root cause.

Stakes: Stratos's largest customer, a Tier 1 automotive supplier, has threatened to shift \$180M in annual orders to a competitor unless the defect escape rate drops below 1% within 12 months. Additionally, each hour of root cause analysis delays corrective action, potentially contaminating downstream wafers.

Constraints: - Inference latency must be under 200ms per die (inline inspection requirement) - The solution must handle missing modalities gracefully (operator notes are absent for ~40% of wafers) - Training data is limited: ~50,000 labeled image-log-text triples (class-imbalanced: 92% good, 8% defective) - Must comply with automotive quality standards (IATF 16949) requiring explainability of defect classification decisions

---

Section 2: Technical Problem Formulation

Problem Type: Multimodal Classification with Missing Modalities

Justification: This is a classification problem (defect vs. non-defect, with sub-categories) that requires fusing information from three distinct modalities: microscope images, structured process logs, and free-text operator notes. The key technical challenge is that no single modality provides sufficient signal for reliable classification -- the system must learn cross-modal patterns.

Input/Output Specification

Inputs: - $x^{(v)} \in \mathbb{R}^{256 \times 256 \times 3}$: High-resolution die microscope image (RGB) - $x^{(p)} \in \mathbb{R}^{48}$: Process log vector (48 numerical parameters: temperatures, pressures, durations, etc.) - $x^{(t)} \in \mathbb{Z}^{L}$: Tokenized operator notes (variable length L , may be empty)

Outputs: - $\hat{y} \in \{0, 1, 2, 3, 4\}$: Defect class - 0: No defect (good die) - 1: Particle contamination - 2: Scratch/mechanical damage - 3: Pattern defect (lithography issue) - 4: Process deviation (chemical/thermal anomaly) - $\hat{a} \in [0, 1]^{K}$: Attention weights over image regions (for explainability)

Mathematical Foundation

The core model architecture uses **mid-level fusion with gated cross-attention**, chosen because: 1. The task requires fine-grained correlation between image regions and process parameters (e.g., a discoloration pattern correlating with a temperature spike) 2. Operator notes are frequently missing, requiring a fusion mechanism that can gracefully degrade 3. Explainability requirements demand attention weights that can be visualized

Modality Encoders:

$$h_v = \text{ViT}(x^{(v)}) \in \mathbb{R}^{N_P \times d}$$

$h_p = W_p \cdot x^{(p)} + b_p \in \mathbb{R}^{1 \times d}$

$$h_t = \text{TransformerEnc}(x^{(t)}) \in \mathbb{R}^{L \times d}$$

where N_p is the number of image patches, d is the shared embedding dimension, and L is the text sequence length.

Fusion via Gated Cross-Attention:

For each pair of modalities, we apply gated cross-attention:

$$h'_v = h_v + \tanh(\alpha_{vp}) \cdot \text{CrossAttn}(h_v, h_p)$$

$h''_v = h'_v + \tanh(\alpha_{vt}) \cdot \text{CrossAttn}(h'_v, h_t)$

where α_{vp} and α_{vt} are learnable gates initialized to zero. When text is missing ($x^{(t)}$ is empty), we simply skip the second cross-attention step -- the gated architecture handles this naturally since $\tanh(0) = 0$.

Loss Function

$$\mathcal{L} = \mathcal{L}_{\text{focal}} + \lambda_1 \mathcal{L}_{\text{contrastive}} + \lambda_2 \mathcal{L}_{\text{attention}}$$

Term 1: Focal Loss ($\mathcal{L}_{\text{focal}}$)

$$\mathcal{L}_{\text{focal}} = -\sum_{c=0}^{4} \alpha_c (1 - p_c)^{\gamma} \cdot y_c \log(p_c)$$

Justification: Standard cross-entropy fails on our class-imbalanced dataset (92% good dies). Focal loss down-weights easy examples (correctly classified good dies) and focuses the gradient on hard examples (missed defects). We set $\gamma = 2$ and use inverse-frequency class weights for α_c .

Term 2: Contrastive Alignment Loss ($\mathcal{L}_{\text{contrastive}}$)

$$\mathcal{L}_{\text{contrastive}} = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{\exp(\text{sim}(v_i, p_i)/\tau)}{\sum_{j=1}^{N} \exp(\text{sim}(v_i, p_j)/\tau)}$$

Justification: Ensures that the image encoder and process log encoder produce aligned representations for the same wafer. Without this term, the cross-attention mechanism would receive unaligned inputs, degrading fusion quality.

Term 3: Attention Regularization ($\mathcal{L}_{\text{attention}}$)

$$\mathcal{L}_{\text{attention}} = -\sum_k a_k \log a_k$$

Justification: Encourages the attention weights to be concentrated rather than uniform. A uniform attention distribution (attending equally to all image regions) provides no useful explainability. This entropy penalty pushes the model to focus on specific defective regions.

Evaluation Metrics

- **Primary:** Defect escape rate (% of defective dies classified as good) -- target < 1%
- **Secondary:** False positive rate (% of good dies flagged as defective) -- target < 15%
- **Tertiary:** Macro-averaged F1 score across all 5 classes

Baseline: The existing vision-only CNN achieves: - Escape rate: 3.2% - False positive rate: 32% - Macro F1: 0.71

Why multimodal fusion is the right approach: The vision-only system cannot distinguish cosmetic variations from real defects without process context. A temperature spike + discoloration pattern = real defect. Normal temperature + discoloration = cosmetic variation. Only cross-modal reasoning can make this distinction.

Section 3: Implementation Notebook Structure

3.1 Data Loading and Preprocessing

```
def load_multimodal_dataset(data_dir, split='train'):
    """
    Load the multimodal defect detection dataset.

    Args:
        data_dir: Path to dataset directory
        split: 'train', 'val', or 'test'

    Returns:
        images: (N, 3, 256, 256) tensor -- die microscope images
        process_logs: (N, 48) tensor -- process parameters
        operator_notes: list of N strings (may contain empty strings)
        labels: (N,) tensor -- defect class labels (0-4)
    """
    # ============ TODO ============
    # Step 1: Load images from data_dir/{split}/images/
    # Step 2: Load process logs from data_dir/{split}/process_logs.csv
    # Step 3: Load operator notes from data_dir/{split}/operator_notes.json
    # Step 4: Load labels from data_dir/{split}/labels.csv
    # Step 5: Apply image augmentations (for train split only):
    #         - Random horizontal/vertical flip
    #         - Random rotation (+/- 15 degrees)
    #         - Color jitter (brightness, contrast)
    # Step 6: Normalize images to [0, 1] range
    # Step 7: Normalize process logs (z-score normalization)
    # ==============================
    pass
```

3.2 Exploratory Data Analysis

```
def explore_dataset(images, process_logs, operator_notes, labels):
    """
    Generate exploratory analysis of the multimodal dataset.
```

```
    Must produce:
    1. Class distribution histogram
    2. Sample images from each defect class
    3. Process parameter distributions per class (box plots)
    4. Word cloud of operator notes per defect class
    5. Missing modality analysis (% of samples with empty notes)
    6. Correlation heatmap between process parameters
    """
    # ============ TODO ============
    # Implement all 6 visualizations above
    # ==============================
    pass
```

3.3 Baseline: Vision-Only Model

```
def build_vision_baseline(num_classes=5):
    """
    Build a vision-only baseline using a pretrained ResNet-18.

    Architecture:
        Image -> ResNet-18 (pretrained) -> Global Avg Pool -> FC -> Output

    Returns:
        model: nn.Module
    """
    # ============ TODO ============
    # Step 1: Load pretrained ResNet-18
    # Step 2: Replace final FC layer for 5-class output
    # Step 3: Freeze early layers (first 3 residual blocks)
    # ==============================
    pass
```

3.4 Multimodal Fusion Model

```
class MultimodalDefectDetector(nn.Module):
    """
    Multimodal defect detection model with gated cross-attention fusion.

    Architecture:
        Image -> ViT encoder -> visual tokens (N_p x d)
        Process logs -> Linear projection -> process token (1 x d)
        Operator notes -> Text encoder -> text tokens (L x d)

        Visual tokens <- Gated CrossAttn with process token
        Visual tokens <- Gated CrossAttn with text tokens
        Fused tokens -> Classification head -> defect class
    """
    def __init__(self, img_size=256, patch_size=16, embed_dim=256,
                 process_dim=48, vocab_size=10000, num_classes=5):
        super().__init__()
        # ============ TODO ============
        # Step 1: Build ViT-based image encoder
        # Step 2: Build process log projection layer
        # Step 3: Build text encoder (simple Transformer or embedding + pooling)
        # Step 4: Build gated cross-attention layers:
        #         - vision_process_xattn: visual tokens attend to process token
        #         - vision_text_xattn: visual tokens attend to text tokens
        # Step 5: Build classification head
        # ==============================
        pass

    def forward(self, images, process_logs, operator_note_tokens=None):
        """
        Args:
            images: (B, 3, 256, 256)
            process_logs: (B, 48)
            operator_note_tokens: (B, L) or None if no notes available

        Returns:
            logits: (B, 5)
            attention_weights: dict of attention maps for explainability
```

```
        """
        # ============ TODO ============
        # Step 1: Encode each modality
        # Step 2: Apply gated cross-attention (vision x process)
        # Step 3: If text available, apply gated cross-attention (vision x text)
        # Step 4: Pool visual tokens and classify
        # Step 5: Return logits and attention weights
        # ==============================
        pass
```

3.5 Training Pipeline

```
def train_multimodal(model, train_loader, val_loader, epochs=50, lr=1e-4):
    """
    Two-stage training pipeline.

    Stage 1 (epochs 1-10): Contrastive alignment only
        - Freeze classification head
        - Train encoders + projection layers with contrastive loss
        - Goal: align image and process embeddings

    Stage 2 (epochs 11-50): Full training
        - Unfreeze classification head
        - Train with focal loss + contrastive loss + attention regularization
        - Use cosine annealing learning rate schedule

    Args:
        model: MultimodalDefectDetector
        train_loader: DataLoader with (images, process_logs, notes, labels)
        val_loader: DataLoader for validation
        epochs: total training epochs
        lr: initial learning rate
    """
    # ============ TODO ============
    # Implement the two-stage training pipeline above
    # Track: train loss, val loss, escape rate, false positive rate, macro F1
    # Use early stopping based on val escape rate
    # ==============================
    pass
```

3.6 Evaluation

```
def evaluate_model(model, test_loader):
    """
    Comprehensive evaluation of the multimodal model.

    Must compute and report:
    1. Overall accuracy
    2. Per-class precision, recall, F1
    3. Confusion matrix (visualized as heatmap)
    4. Defect escape rate
    5. False positive rate
    6. Comparison with vision-only baseline
    7. Performance with and without operator notes
    """
    # ============ TODO ============
    # Implement all 7 evaluation metrics above
    # ==============================
    pass
```

3.7 Error Analysis

```
def error_analysis(model, test_loader, num_examples=20):
    """
    Detailed error analysis on misclassified samples.

    For each misclassified sample, show:
    1. The die microscope image with attention heatmap overlay
    2. The process parameters that deviate most from normal
```

```
    3. The operator notes (if available)
    4. Model's predicted class vs true class
    5. Model's confidence (softmax probabilities)

    Also aggregate:
    6. Which defect types are most confused with each other
    7. Whether errors correlate with missing operator notes
    8. Whether errors cluster in specific process parameter ranges
    """
    # ============ TODO ============
    # Implement error analysis as described above
    # ==============================
    pass
```

3.8 Deployment Considerations

```
def optimize_for_deployment(model):
    """
    Prepare model for inline inspection deployment.

    Steps:
    1. Quantize model to INT8 (torch.quantization)
    2. Trace model with torch.jit.trace for optimized inference
    3. Benchmark inference latency (target: <200ms on T4 GPU)
    4. Test that quantized model maintains escape rate < 1%
    5. Export as ONNX for deployment flexibility
    """
    # ============ TODO ============
    # Implement deployment optimization pipeline
    # ==============================
    pass
```

3.9 Ethics and Fairness

```
def fairness_audit(model, test_loader, metadata):
    """
    Audit model for fairness across manufacturing conditions.

    Check for performance disparities across:
    1. Different fabrication chambers (fab A vs fab B vs fab C)
    2. Different shifts (day vs night -- operator fatigue)
    3. Different wafer positions (center vs edge)
    4. Presence vs absence of operator notes
    5. Seasonal variation (humidity/temperature effects)

    Report:
    - Per-subgroup escape rate and false positive rate
    - Maximum performance gap across subgroups
    - Recommendations for bias mitigation
    """
    # ============ TODO ============
    # Implement fairness audit across the dimensions above
    # ==============================
    pass
```

Section 4: Production and System Design Extension

System Architecture

The production deployment consists of four main components:

1. **Inline Inspection Module:** Runs the multimodal model on each die as it passes through the inspection station. Receives images from the AOI camera, process logs from the MES (Manufacturing Execution System), and operator notes from the shift management tool.

2. **Feature Store:** Caches encoded modality features for fast retrieval during root cause analysis. When a defect batch is detected, engineers can query the feature store to find historically similar defects across all three modalities.

3. **Explainability Dashboard:** Visualizes attention heatmaps overlaid on die images, highlights anomalous process parameters, and surfaces relevant operator notes for each flagged die.

4. **Feedback Loop:** Operators can confirm or override model decisions. These corrections flow back into a retraining queue for continuous model improvement.

API Design

```
POST /api/v1/inspect
Content-Type: multipart/form-data

Parameters:
  - image: JPEG (256x256 die microscope image)
  - process_log: JSON (48 process parameters)
  - operator_notes: string (optional, may be empty)
  - wafer_id: string (for traceability)

Response:
{
  "defect_class": 0-4,
  "confidence": 0.0-1.0,
  "attention_heatmap": base64 PNG,
  "anomalous_parameters": ["temperature_zone_3", "etch_duration"],
  "explanation": "Discoloration in die region B3 correlates with +2.1 sigma temperature deviation in zone 3.",
  "latency_ms": 142
}
```

Model Serving

- **Hardware:** NVIDIA T4 GPU per inspection station (inference)
- **Framework:** TorchServe with custom handler for multimodal input batching
- **Batch size:** Dynamic batching (1-8 dies) to maximize GPU utilization
- **Latency budget:** 200ms total (120ms model inference + 80ms pre/post-processing)
- **Throughput:** ~300 dies/minute per station (5 dies/second)

Monitoring

- **Model performance:** Track escape rate and false positive rate on a rolling 24-hour window. Alert if escape rate exceeds 1.5% or false positive rate exceeds 20%.
- **Feature drift:** Monitor distribution shift in process log features using KL divergence. Alert if any parameter drifts >3 sigma from training distribution.
- **Latency:** Track P50, P95, P99 inference latency. Alert if P99 exceeds 300ms.
- **Missing modality rate:** Track percentage of inspections without operator notes. If it drops below 50%, investigate whether operators are being encouraged to enter notes.

Data Drift Detection

- **Image drift:** Monitor pixel intensity distributions and FID (Frechet Inception Distance) between recent inspection images and training set.
- **Process drift:** Track individual process parameter means and variances. New recipes or equipment changes will shift distributions.
- **Text drift:** Monitor vocabulary distribution in operator notes. New terminology may indicate process changes.
- **Trigger for retraining:** Any drift metric exceeding threshold for >6 consecutive hours.

A/B Testing

- **Deployment strategy:** Shadow mode first (run multimodal model alongside existing CNN, compare decisions without affecting production)
- **Duration:** 4 weeks of shadow mode, then gradual rollout (10% -> 25% -> 50% -> 100%)
- **Success criteria:** Escape rate < 1%, false positive rate < 15%, zero increase in production line downtime
- **Rollback:** Automatic fallback to vision-only CNN if escape rate exceeds 2% for any 4-hour window

CI/CD Pipeline

1. **Data versioning:** DVC for tracking training data updates (new defect images, process logs)
2. **Model registry:** MLflow for experiment tracking and model versioning
3. **Automated testing:**
4. Unit tests: model forward pass, loss computation, data loading
5. Integration tests: end-to-end inference with sample wafers
6. Performance tests: latency benchmarks, accuracy on held-out defect catalog
7. **Deployment:** GitOps with ArgoCD, deploying to edge Kubernetes cluster at each fab
8. **Canary releases:** New model version serves 5% of traffic, monitored for 48 hours before full rollout

Cost Estimation

Component	Monthly Cost
T4 GPUs (3 fabs x 4 stations)	\$8,400
Data storage (images + logs)	\$2,100
Retraining compute (weekly)	\$1,200
MLOps platform (MLflow, DVC)	\$800
Monitoring (Datadog/Grafana)	\$500
Total	**\$13,000/month**

ROI: The multimodal system is projected to reduce annual defect costs from \$47M to \$18M, yielding net savings of \$29M/year against a \$156K/year operating cost -- a 186x return on investment.