Case Study: Automated Wafer Defect Detection Using Variational Autoencoders

Section 1: Industry Context and Business Problem

Industry: Semiconductor Manufacturing

The global semiconductor industry is valued at over USD 600 billion, with fabrication ("fab") facilities producing billions of chips annually. A single modern fab costs USD 10-20 billion to build and operates 24/7. At the heart of every fab is a quality control pipeline: every wafer must be inspected for surface defects before it proceeds to the next processing step. A single undetected defect can render an entire wafer — worth USD 5,000-USD 50,000 depending on the process node — worthless.

Company Profile: NovaSilicon

- **Founded:** 2019
- **Headquarters:** Austin, Texas
- **Employees:** 820 (including 45 ML/data engineers)
- **Stage:** Series C (USD 340M raised, USD 1.8B valuation)
- **Product:** Advanced 7nm and 5nm logic chips for automotive and edge AI applications
- **Fab Capacity:** 12,000 wafers per month across two production lines
- **Key Customers:** Three Tier-1 automotive OEMs and two major edge AI chip integrators

NovaSilicon operates a state-of-the-art fab that produces advanced logic chips for safety-critical automotive applications. Their chips power autonomous driving perception stacks and in-vehicle AI inference engines. Given the safety-critical nature of their end applications, NovaSilicon maintains a zero-tolerance defect policy: every shipped chip must pass rigorous quality standards.

Business Challenge

NovaSilicon's current wafer inspection pipeline relies on a combination of rule-based optical inspection systems and a team of 28 human inspectors working in three shifts. The system works as follows:

1. High-resolution optical microscopes capture surface images of each wafer at multiple inspection points (approximately 200 images per wafer, each 256x256 pixels).
2. Rule-based algorithms flag images with brightness or texture anomalies.

3. Human inspectors review flagged images and make pass/fail decisions.
4. Inspectors also classify defect types for root-cause analysis.

This pipeline has three critical problems:

Problem 1: Throughput Bottleneck. As NovaSilicon scales to 12,000 wafers/month, the inspection team must review approximately 2.4 million images per month. The current team can handle 1.8 million, creating a growing backlog. Hiring more inspectors is expensive (USD 85K/year fully loaded per inspector) and the labor market for trained semiconductor inspectors is extremely tight.

Problem 2: Inconsistency. Human inspector agreement rate is only 78%. The same image shown to two different inspectors will receive different classifications 22% of the time. This inconsistency leads to both false passes (defective wafers reaching customers) and false rejects (good wafers being scrapped unnecessarily).

Problem 3: Novel Defect Blindness. The rule-based system is tuned to detect known defect patterns (scratches, particles, pattern shifts). When a new defect type emerges — as happened twice in the past year due to process changes — the rules miss it entirely until an engineer manually programs new detection logic. The average time to detect a novel defect class is 11 days, during which hundreds of defective wafers may be produced.

Why It Matters

- **Scrap Cost:** Each false-negative (missed defect) that reaches packaging costs USD 12,000 in wasted downstream processing. NovaSilicon estimates 340 false-negatives per month, costing USD 4.1M annually.
- **Over-Rejection Cost:** Each false-positive (good wafer scrapped) costs the full wafer value, averaging USD 18,000. An estimated 120 false-positives per month cost USD 25.9M annually.
- **Customer Risk:** If a defective chip reaches an automotive customer and fails in the field, the liability exposure is measured in hundreds of millions of dollars, plus irreparable reputation damage.
- **Novel Defect Exposure:** The 11-day detection lag for new defect types represents a potential exposure of 4,400 wafers (USD 79M in wafer value) before the defect is caught.

Total estimated annual cost of the current inspection inadequacy: **USD 109M**.

Constraints

Constraint	Requirement
Inference Latency	< 50ms per image (to maintain line speed)
Training Data	Abundant normal wafer images (~500K); very few labeled defect images (~200 total across 5 known defect types)

Constraint	Requirement
Compute Budget	4x NVIDIA A100 GPUs for training; 2x NVIDIA T4 GPUs for inference
Deployment	On-premise (fab data cannot leave the facility due to IP protection)
Compliance	ITAR restrictions on data export; internal ISO 9001 quality management
False Negative Rate	Must be < 0.1% (safety-critical downstream application)
Availability	99.9% uptime — the inspection system cannot be a production bottleneck
Team Expertise	ML team has PyTorch experience but limited generative modeling background

---

Section 2: Technical Problem Formulation

Problem Type: Unsupervised Anomaly Detection via Generative Modeling

The core challenge is detecting defects when we have abundant normal data but very few (and highly diverse) defect examples. This rules out standard supervised classification for the following reasons:

1. **Class imbalance:** Normal images outnumber defective images by 2500:1. Even with aggressive oversampling or loss reweighting, supervised classifiers struggle with such extreme imbalance.
2. **Open-set detection:** New defect types emerge unpredictably. A supervised classifier trained on 5 known defect types will assign high confidence to normal when shown a 6th, previously unseen defect type. The model cannot detect what it has never seen.
3. **Label scarcity:** Labeling defect images requires expert inspectors and is expensive. Acquiring a balanced labeled dataset would require years of production data.

The alternative framing is **anomaly detection**: learn what "normal" looks like, and flag anything that deviates from normal. This is fundamentally an unsupervised (or self-supervised) problem — we train exclusively on normal data.

Among anomaly detection approaches, we choose **generative modeling** over discriminative alternatives (one-class SVM, isolation forest) because:

- Generative models learn the full data distribution, enabling both detection (via reconstruction error) and diagnosis (via latent space analysis).
- The reconstruction itself is interpretable — inspectors can see exactly what the model "expected" versus what it observed, pinpointing the defect location.
- The latent space provides a natural mechanism for clustering defect types without requiring defect labels.

Why a Variational Autoencoder

Among generative models, the VAE is the right choice for this problem for several fundamental reasons:

1. **Smooth latent space.** The KL divergence regularization ensures the latent space is continuous and well-organized. Normal wafers will cluster together, and defective wafers will be pushed to low-density regions. This property — which a regular autoencoder lacks — is essential for reliable anomaly scoring.

2. **Principled reconstruction error.** The VAE's reconstruction term in the ELBO provides a principled, probabilistic reconstruction error metric. Unlike an autoencoder's MSE loss (which is arbitrary in scale), the VAE's negative log-likelihood has a proper probabilistic interpretation.

3. **Controllable tradeoff.** The ELBO's two terms — reconstruction and KL divergence — provide a natural lever for controlling the sensitivity-specificity tradeoff. Weighting reconstruction more heavily makes the model more sensitive to subtle defects; weighting KL more heavily makes the latent space more regular, improving generalization.

4. **Inference speed.** A VAE requires only a single forward pass through the encoder and decoder at inference time (~5ms on a T4 GPU for a 256x256 image). This easily meets the 50ms latency requirement. Diffusion models, while producing sharper reconstructions, require hundreds of denoising steps and would not meet the latency budget.

5. **Training stability.** VAEs train with a single, well-defined objective (ELBO maximization) using standard gradient descent. GANs, the other major generative family, suffer from mode collapse and training instability that would be unacceptable in a safety-critical production system.

Input Specification

Property	Value
Format	Single-channel grayscale images
Resolution	256 x 256 pixels
Pixel Range	[0, 1] after normalization
Tensor Shape	(batch_size, 1, 256, 256)

Each image captures a localized region of the wafer surface under controlled illumination. The single-channel format is chosen because the optical inspection microscopes produce grayscale images. Images are center-cropped and intensity-normalized to remove illumination variation across inspection stations.

Output Specification

The model produces three outputs for each input image:

1. **Reconstructed Image** $\hat{x} \in [0,1]^{256\times256}$ — what the model "expects" a normal wafer to look like at this location.

2. **Anomaly Score** $a(x) \in \mathbb{R}_{\geq 0}$ — a scalar measuring how anomalous the input is. Defined as the pixel-wise reconstruction error aggregated over the image:

$$a(x) = \frac{1}{D}\sum_{i=1}^{D}(x_i - \hat{x}_i)^2$$

where $D = 256 \times 256 = 65{,}536$ is the number of pixels.

1. **Anomaly Heatmap** $h(x) \in \mathbb{R}_{\geq 0}^{256\times256}$ — a spatial map of per-pixel reconstruction error, enabling defect localization:

$$h(x)_{ij} = (x_{ij} - \hat{x}_{ij})^2$$

The reconstructed image is chosen as the output representation (rather than, say, a binary label) because it provides maximum interpretability: a human inspector can overlay the anomaly heatmap on the original image to immediately see where and how the defect manifests.

Mathematical Foundation

The VAE's training objective is the Evidence Lower Bound (ELBO):

$$\mathcal{L}(\theta,\phi;x) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{Reconstruction}} - \underbrace{D_{KL}(q_\phi(z|x)\|p(z))}_{\text{Regularization}}$$

Reconstruction Term. For grayscale images with pixel values in [0, 1], we model the decoder output as a Bernoulli distribution for each pixel:

$$\log p_\theta(x|z) = \sum_{i=1}^{D}[x_i \log \hat{x}_i + (1-x_i)\log(1-\hat{x}_i)]$$

This is the negative binary cross-entropy. It penalizes the model for assigning low probability to the true pixel values. In the anomaly detection context, this term forces the model to faithfully reconstruct normal wafer patterns. When the model encounters a defective image, it cannot reconstruct the defect (because it was never trained on defects), producing a high reconstruction error at the defect location.

KL Divergence Term. With a Gaussian encoder $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma^2_\phi(x)\cdot I)$ and standard normal prior $p(z) = \mathcal{N}(0, I)$, the KL divergence has a closed-form solution:

$$D_{KL} = -\frac{1}{2}\sum_{j=1}^{J}\left(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right)$$

where J is the latent dimension. This term ensures the latent space remains organized. Without it, the encoder could memorize each normal image as an isolated point, leaving gaps where defective images would be mapped. The KL term prevents this by requiring the encoder distributions to overlap with the standard normal prior.

Why each term matters for anomaly detection:

- If you remove the reconstruction term, the model learns a perfect standard normal latent space but cannot reconstruct anything — anomaly detection fails because all images have equally bad reconstructions.
- If you remove the KL term, the model becomes a regular autoencoder with a fragmented latent space. It may achieve good reconstruction on training data but will also reconstruct defects well (by memorizing patterns), reducing anomaly detection sensitivity.
- The balance between these terms is controlled by a coefficient β in β-VAE: $\mathcal{L} = \text{Recon} - \beta \cdot D_{KL}$. For anomaly detection, $\beta < 1$ is often preferred (emphasize reconstruction quality to maximize sensitivity to pixel-level defects).

Reparameterization Trick. The sampling step $z \sim q_\phi(z|x)$ is made differentiable via:

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

This enables end-to-end gradient-based training of both the encoder and decoder.

Loss Function

The full training loss for a batch of N normal images:

$$\mathcal{L}_{\text{total}} = \frac{1}{N} \sum_{n=1}^{N} \left[-\sum_{i=1}^{D} [x_i^{(n)} \log \hat{x}_i^{(n)} + (1 - x_i^{(n)}) \log(1 - \hat{x}_i^{(n)})] + \beta \cdot$$

Term	Purpose	Effect of Removal
Reconstruction (BCE)	Faithful pixel-level reconstruction of normal patterns	Model cannot reconstruct anything; all anomaly scores are uniformly high
KL Divergence	Smooth, regular latent space; prevents memorization	Latent space becomes fragmented; model may memorize and reconstruct defects
β coefficient	Controls reconstruction vs. regularization balance	$\beta = 1$: standard VAE. $\beta < 1$: sharper reconstructions, better pixel-level anomaly detection. $\beta > 1$: more disentangled latent space, better for clustering

Evaluation Metrics

Metric	Definition	Target	Rationale
AUROC (primary)	Area under ROC curve for anomaly detection	> 0.95	Standard metric for binary detection performance across all thresholds

Metric	Definition	Target	Rationale
AUPRC	Area under precision-recall curve	> 0.85	More informative than AUROC under extreme class imbalance
FNR at FPR=5%	False negative rate when false positive rate is 5%	< 0.5%	Directly measures missed defect rate at an operationally acceptable false alarm rate
Pixel-level AUROC	Per-pixel anomaly localization accuracy	> 0.90	Measures defect localization quality for root-cause analysis
Inference Latency	Time per image on T4 GPU	< 50ms	Production line speed requirement

Baseline Approach

The current baseline is a rule-based system using hand-crafted texture features (Gabor filters, local binary patterns) fed into a one-class SVM. This system achieves:

- AUROC: 0.82
- FNR at FPR=5%: 8.3%
- Cannot localize defects (binary image-level decision only)
- Requires manual tuning for each new product line (~2 weeks of engineer time)
- Cannot detect novel defect types until rules are manually updated

The VAE approach aims to exceed all of these metrics while providing defect localization and generalization to novel defect types.

Technical Constraints

Constraint	Budget
Model Parameters	< 15M (to fit on T4 GPU with room for batching)
Training Time	< 8 hours on 4x A100
Inference Latency	< 50ms per image on T4
Latent Dimension	128 (balances reconstruction quality and latent space regularity)
Training Data	~500K normal wafer images
Validation Data	1,000 normal + 200 defective (5 known types, ~40 each)

Section 3: Implementation Notebook Structure

3.1 Data Acquisition Strategy

Dataset: MVTec Anomaly Detection Dataset (MVTec AD)

The MVTec AD dataset is a widely-used benchmark for unsupervised anomaly detection in industrial inspection. We will use the "transistor" category as a proxy for wafer inspection, as it contains high-resolution grayscale images of electronic components with various defect types.

- **Training set:** 213 defect-free images (normal only)
- **Test set:** 60 normal + 40 defective images across 4 defect types (bent lead, cut lead, damaged case, misplaced)
- **Image resolution:** 1024x1024, which we resize to 256x256
- **Ground truth:** Pixel-level segmentation masks for defective images

Why this dataset: It mirrors the real-world setting where normal data is abundant and labeled defect data is scarce. The defect types are diverse and realistic — exactly the scenario NovaSilicon faces.

Data Pipeline: 1. Download MVTec AD transistor subset 2. Resize to 256x256 3. Convert to single-channel grayscale (if needed) 4. Normalize pixel values to [0, 1] 5. Split training set: 80% train, 20% validation (all normal images)

TODO: Implement data augmentation (random rotation, horizontal/vertical flip, Gaussian noise) to increase the effective training set size. Justify which augmentations are appropriate for wafer images and which are not.

3.2 Exploratory Data Analysis

Before building any model, we must understand the data:

- Plot a grid of 16 random normal images to understand the visual patterns
- Plot histograms of pixel intensity distributions across the training set
- Compute and visualize the mean image and standard deviation image
- Plot 16 defective images (from the test set) alongside their ground-truth masks
- Compare pixel intensity distributions of normal vs. defective images

TODO: Write EDA code to generate these visualizations. Answer the following questions: 1. What is the dominant texture pattern in normal images? 2. How do defective images differ visually from normal ones? 3. Are the defects localized (affecting a small region) or global (affecting the entire image)? 4. Based on the pixel distributions, would simple thresholding work as a baseline? Why or why not?

3.3 Baseline Approach

Before building the VAE, implement a simple baseline to establish a performance floor.

Baseline: PCA Reconstruction Error

Principal Component Analysis can be viewed as a linear autoencoder. By projecting images into a low-dimensional PCA space and reconstructing them, the reconstruction error serves as an anomaly score.

1. Flatten all training images to vectors (256x256 = 65,536 dimensions)
2. Fit PCA with k components on the training set
3. For each test image: project to PCA space, reconstruct, compute MSE
4. Threshold the MSE to classify normal vs. anomalous

TODO: Implement the PCA baseline. Try $k \in \{10, 50, 100, 200\}$. Compute AUROC for each. Plot the ROC curves. Report the best AUROC achieved.

Why PCA as a baseline: PCA captures the dominant linear modes of variation in normal images. It fails for anomaly detection because real-world defects often involve subtle, nonlinear deviations that PCA cannot distinguish from normal variation. This motivates the need for a nonlinear generative model (the VAE).

3.4 Model Design

Architecture: Convolutional VAE

We use a convolutional architecture rather than the fully-connected architecture from the article because: - Images have spatial structure that convolutions exploit (translation equivariance) - Convolutional models are far more parameter-efficient for image data - The decoder can produce spatially coherent reconstructions needed for defect localization

Encoder: Four convolutional blocks, each consisting of Conv2d -> BatchNorm -> LeakyReLU -> stride-2 downsampling. Output is flattened and passed through two linear heads to produce μ and $\log(\sigma^2)$.

Layer	Output Shape	Parameters
Input	(1, 256, 256)	-
Conv 3x3, 32 filters, stride 2	(32, 128, 128)	320
Conv 3x3, 64 filters, stride 2	(64, 64, 64)	18,496
Conv 3x3, 128 filters, stride 2	(128, 32, 32)	73,856
Conv 3x3, 256 filters, stride 2	(256, 16, 16)	295,168
Flatten	(65,536)	-
Linear -> mu	(128)	8,388,736
Linear -> logvar	(128)	8,388,736

Decoder: Mirror of encoder using ConvTranspose2d for upsampling. Final layer uses sigmoid activation to produce pixel probabilities in [0, 1].

Design Decisions: - **LeakyReLU over ReLU:** Prevents dead neurons in the encoder, which is important because we need the encoder to produce meaningful gradients for all input patterns. - **BatchNorm:** Stabilizes training by normalizing activations, allowing higher learning rates. - **Latent dimension 128:** Chosen to balance reconstruction quality (higher dim = better reconstruction) against latent space regularity (lower dim = smoother latent space). For 256x256 grayscale images with moderate complexity, 128 is a reasonable starting point. - **Stride-2 convolution over pooling:** Learnable downsampling preserves more information than fixed max/average pooling.

TODO: Implement the convolutional encoder and decoder. We provide the class signatures and forward method structure. You must fill in the layer definitions and the forward pass logic.

3.5 Training Strategy

Optimizer: AdamW with weight decay 1e-5

Why AdamW: Adam's adaptive learning rates handle the varying gradient magnitudes across encoder and decoder layers. Weight decay provides additional regularization beyond the KL term, preventing the convolutional filters from overfitting to specific normal patterns.

Learning Rate Schedule: Cosine annealing from 1e-3 to 1e-5 over the full training run

Why cosine: Cosine annealing provides a smooth decay that avoids the sharp transitions of step schedules. The initial high learning rate allows the model to quickly learn dominant patterns; the slow final decay allows fine-tuning of reconstruction quality.

KL Annealing: Linearly increase β from 0 to 1 over the first 10 epochs

Why anneal: Starting with $\beta = 0$ (pure autoencoder) allows the model to first learn to reconstruct well before the KL term begins regularizing the latent space. Without annealing, the KL term can dominate early training, collapsing the latent space to the prior before the decoder learns to reconstruct anything — a phenomenon called "posterior collapse."

Training Configuration: - Batch size: 32 - Epochs: 50 - Gradient clipping: max norm 1.0 - Early stopping: patience 10 on validation reconstruction loss

TODO: Implement the training loop with KL annealing, gradient clipping, and separate logging of reconstruction loss, KL loss, and total loss. Plot all three loss curves on the same figure at the end of training.

3.6 Evaluation

After training, evaluate the model's anomaly detection performance:

1. **Compute anomaly scores** for all test images (both normal and defective)
2. **Plot the score distributions** for normal vs. defective images — are they separable?
3. **Compute AUROC** and **AUPRC** using sklearn
4. **Plot the ROC curve** with the operating point at FPR=5% highlighted

5. **Compute pixel-level AUROC** using the anomaly heatmaps and ground-truth masks
6. **Compare against the PCA baseline** from Section 3.3

TODO: Implement the evaluation pipeline. Generate all plots and metrics listed above. Compute the FNR at FPR=5% and compare to the target of < 0.5%.

3.7 Error Analysis

Systematic error analysis is critical for deploying an anomaly detection system in production.

1. **False Negatives (Missed Defects):** Identify all defective images that the model scores below the threshold. For each:
2. Visualize the original image, reconstruction, and anomaly heatmap
3. Categorize why the model missed it (subtle defect? defect in a region the model reconstructs poorly anyway?)
4. **False Positives (False Alarms):** Identify all normal images that the model scores above the threshold. For each:
5. What about this normal image made the model think it was defective?
6. Is there an unusual but normal pattern (e.g., a rare but valid component orientation)?
7. **Defect Type Breakdown:** Compute per-defect-type detection rates. Which defect types are easiest/hardest to detect? Why?

TODO: Implement the error analysis. Categorize the top 3 failure modes. For each, propose a concrete mitigation strategy (e.g., data augmentation, architecture change, post-processing).

3.8 Scalability and Deployment Considerations

With the model trained and evaluated, consider the production deployment:

1. **Inference Benchmarking:** Measure the inference time per image on both GPU and CPU. Run 100 images and report mean and P99 latency.
2. **Batch vs. Single Inference:** Compare latency for batch sizes 1, 8, 32, 64. Determine the optimal batch size for the production throughput requirement.
3. **Model Export:** Export the trained model to TorchScript for production serving. Verify that the exported model produces identical outputs.
4. **Memory Footprint:** Report the model size on disk and GPU memory usage during inference.

TODO: Implement the benchmarking script. Export the model to TorchScript. Report all latency and memory numbers in a summary table.

3.9 Ethical and Regulatory Analysis

Even in manufacturing settings, ML systems have ethical and regulatory implications.

1. **Failure Mode Impact:** If the model misses a defect in a safety-critical automotive chip, what are the downstream consequences? How does this compare to the human inspector failure rate?
2. **Automation Bias:** Will human inspectors over-rely on the model's output and reduce their own vigilance? How can the system design mitigate this?
3. **Worker Displacement:** If the model automates 80% of inspections, what happens to the 28 human inspectors? What is the responsible transition plan?
4. **Regulatory Compliance:** What documentation and validation is required under ISO 9001 and automotive quality standards (IATF 16949) to deploy an ML-based inspection system?

TODO: Write a brief (300-500 word) ethical impact assessment addressing the four points above. Propose at least one concrete mitigation for each risk.

Section 4: Production and System Design Extension

Architecture Overview

The production system consists of five major components:

```
[Optical Inspection Station]
         |
         v
[Image Preprocessing Service]
         |
         v
[VAE Inference Service (GPU)]
    |              |
    v              v
[Anomaly Score]  [Heatmap]
    |              |
    v              v
[Decision Engine (threshold + rules)]
    |          |           |
    v          v           v
[PASS]   [REVIEW]    [REJECT]
              |
              v
    [Human Inspector UI]
              |
              v
    [Feedback Loop -> Retraining Pipeline]
```

API Design

Inference Endpoint

```
POST /api/v1/inspect
Content-Type: application/octet-stream
```

```
Request Body: Raw image bytes (PNG or TIFF, 256x256 grayscale)

Response (JSON):
{
  "wafer_id": "WF-2024-001234",
  "inspection_point": 42,
  "anomaly_score": 0.0342,
  "decision": "PASS",            // PASS | REVIEW | REJECT
  "confidence": 0.97,
  "heatmap_url": "/api/v1/heatmaps/WF-2024-001234-42.png",
  "latent_vector": [0.12, -0.43, ...],  // 128-dim
  "model_version": "v2.3.1",
  "inference_time_ms": 8.2
}
```

Thresholds: - anomaly_score < 0.02: PASS (automated) - 0.02 <= anomaly_score < 0.05: REVIEW (human inspector) - anomaly_score >= 0.05: REJECT (automated)

The REVIEW band ensures that borderline cases receive human oversight, providing a safety net during the initial deployment phase. As confidence in the model grows, the REVIEW band can be narrowed.

Serving Infrastructure

- **Model Server:** NVIDIA Triton Inference Server with TorchScript backend
- **GPU:** 2x NVIDIA T4 (one primary, one failover)
- **Load Balancer:** Round-robin across GPU instances
- **Scaling:** Fixed capacity (not autoscaling) — the inspection rate is deterministic and bounded by the optical system's throughput
- **Caching:** No caching — every image must be evaluated fresh

Latency Budget

Component	Budget	Notes
Image transfer (camera to service)	5ms	Local network, images are small
Preprocessing (resize, normalize)	2ms	CPU-bound, negligible
VAE forward pass (encoder + decoder)	8ms	GPU inference on T4
Anomaly score computation	1ms	Element-wise operations
Heatmap generation	2ms	Optional, async for REVIEW cases
Decision logic	< 1ms	Simple threshold comparison
Response serialization	< 1ms	JSON encoding
Total	**~20ms**	Well within 50ms budget

Monitoring

Real-Time Metrics (Grafana Dashboard): - Anomaly score distribution (rolling 1-hour histogram) - PASS/REVIEW/REJECT rates per hour - Inference latency P50, P95, P99 - GPU utilization and memory usage - Queue depth (images waiting for inference)

Daily Metrics: - False positive rate (estimated from human inspector overrides of REJECT decisions) - Human inspector agreement rate on REVIEW cases - Anomaly score drift (comparing today's distribution to the baseline)

Alerts: - REJECT rate exceeds 5% of inspections (possible process issue or model degradation) - P99 latency exceeds 40ms (approaching the 50ms budget) - GPU memory utilization exceeds 85% - Anomaly score distribution shift detected (KS test p-value < 0.01)

Model Drift Detection

Semiconductor manufacturing processes evolve: new products, new process recipes, equipment wear. The model must detect when the data distribution has shifted beyond its training domain.

Strategy: 1. **Reconstruction loss monitoring.** Track the average reconstruction loss on images labeled PASS. If this metric trends upward, the model is seeing patterns it was not trained on — even if those patterns are normal for the current process. 2. **Latent space monitoring.** Track the mean and variance of the encoder's latent vectors over time. Sudden shifts indicate distribution change. 3. **KS test on anomaly scores.** Weekly Kolmogorov-Smirnov test comparing the current week's PASS anomaly scores to the training set's anomaly scores.

Response: When drift is detected, trigger a model retraining pipeline using the most recent 3 months of PASS images (validated by human inspectors).

Model Versioning

- **Registry:** MLflow model registry with Git-tagged model versions
- **Naming:** `vae-wafer-v{major}.{minor}.{patch}` (e.g., `vae-wafer-v2.3.1`)
- **Rollback:** One-command rollback to previous version in Triton; maximum rollback time: 30 seconds
- **Retention:** Last 5 model versions kept warm; older versions archived to S3

A/B Testing

New models are deployed via shadow mode before promotion:

1. **Shadow Phase (1 week):** New model runs in parallel with production model. Both models score every image, but only the production model's decision is used. Compare AUROC, latency, and score distributions.
2. **Canary Phase (1 week):** 10% of traffic routed to the new model. Monitor REJECT rate, REVIEW rate, and human inspector override rate.

3. **Full Rollout:** If canary metrics meet targets, promote to 100%.

Statistical significance: Require $p < 0.01$ on a two-sided proportion test for the REJECT rate difference before declaring the new model better or worse.

Guardrail metrics: If FNR at any point exceeds 1% (10x the target), immediately roll back.

CI/CD for ML

Training Pipeline (Airflow DAG): 1. Data extraction: Pull latest 3 months of PASS images from inspection database 2. Data validation: Check for corrupted images, distribution sanity checks 3. Training: Launch training job on 4x A100 cluster 4. Evaluation: Run full evaluation suite (AUROC, AUPRC, FNR, latency) 5. Model validation gate: All metrics must meet thresholds 6. Registry: Push to MLflow model registry with evaluation report 7. Shadow deployment: Automatically deploy to shadow mode

Trigger: Manual trigger or automated monthly retraining on the 1st of each month.

Cost Analysis

Item	Monthly Cost
Training (4x A100, 8 hrs/month)	USD 250
Inference (2x T4, 24/7)	USD 1,400
Storage (model artifacts, logs)	USD 50
MLflow server	USD 100
Monitoring (Grafana/Prometheus)	USD 150
Total	**USD 1,950/month**

ROI: The current inspection system costs USD 109M/year in waste, over-rejection, and risk. Even a 10% improvement from the VAE system saves USD 10.9M/year, against a compute cost of USD 23,400/year — a 466x return on infrastructure investment.

End of Case Study