SEMI Challenge Overview

SEMI Challenge Summary Map

🔄 **Updated Strategic Approach: The Tier-Based System**

> **Following feedback from our mentor, we refined our solution using a tier-based framework — a layered strategy that enables our system to evolve from basic detection to fully autonomous quality control.**

We define 9 tiers, where each tier adds a new level of intelligence, traceability, and business value. This structure helps us plan clearly, scale gradually, and justify our design decisions.

Tier	Function
Tier 0	Detect defects in real-time (e.g. YOLOv8n on edge)
Tier 1	Make basic Accept / Resample / Reject decisions
Tier 2	Adjust sampling using batch history and severity
Tier 3	Trace defects back to machine/stage (forensic mapping)
Tier 4	Control flow (e.g. stop/resample batches with high risk)
Tier 5	Use sampling data to retrain models
Tier 6	Optimize inspection cost vs. defect risk
Tier 7	Auto-schedule inspection priorities based on risk trends

Tier 8	**Learn across multiple lines or sites**
Tier 9	**Achieve full autonomous quality control (AQIS)**

This tier-based model now guides the architecture, logic, and deployment of our AI + Smart Sampling system.

PHASE 1: PROBLEM UNDERSTANDING

No.	Task	Status	Key Insight
1	Understand the challenge	✅	AI + Smart Sampling is needed to replace outdated AQL inspection
2	Map semiconductor process chain	✅	Focus on step 9: Final Visual Inspection (FVI) where missed defects occur
3	Identify QC bottleneck	✅	FVI is manual / AQL-based → error-prone, not scalable
4	Analyze AQL limitations	✅	AQL = random %, no logic, no learning → defects bypassed
5	Highlight business pain points	✅	Labor shortage, low yield, rework cost, missed defects
6	Benchmark current industry status	✅	Tier-1 fabs use AI in mid-line only → FVI remains manual = innovation gap

PHASE 2: SOLUTION DESIGN

No.	Task	Status	Key Insight
7	Design overall AI + Sampling system	✅	Replace AQL with real-time detection + decision logic (Accept/Resample/Reject)

8	Select hardware platform	✅	RPi/Jetson + Camera = low-cost, edge-ready, scalable
9	Develop AI model	✅	YOLOv8n for image detection (lightweight + fast)
10	Sampling engine logic	✅	Rule-based + optional RL; uses defect + batch history to decide sampling %
11	Add learning mechanism (feedback)	✅	LightGBM learns defect trends to adjust future sampling logic
12	Integrate forensic traceability	✅	Map defect to specific production stage/machine using sensor input
13	Introduce cost-aware decision logic	✅	AI adjusts sampling strategy based on inspection cost vs. defect risk

PHASE 3: CORE TECHNOLOGY

No.	Task	Status	Key Insight
14	Choose defect detection model	✅	YOLOv8n: compact, explainable, runs well on edge devices
15	Integrate AI with sampling engine	✅	Model output → triggers rule/RL sampling logic per wafer
16	Define sampling decision rules	✅	Rules based on defect severity, batch history, time-of-day, etc.
17	Implement feedback learning	✅	LightGBM adjusts sampling strategy using historical defect trends
18	Add economic optimization layer	✅	Sampling rate changes based on defect risk vs inspection cost trade-off

19	Enable multi-source data inputs	✅	Combine image + sensor logs (e.g. temperature, equipment ID)
20	Visualize logic flow (dashboard/mock)	✅	Build simplified dashboard showing decisions & learning loop

PHASE 4: MOCK DATA & DEMO

No.	Task	Status	Key Insight
21	Prepare defect image dataset	✅	Use open datasets (scratches, cracks, PCB errors) for simulation
22	Simulate AI output table	✅	ai_outputs.csv: shows defect type, severity, confidence, decision
23	Create sensor log mockup	✅	sensor_log.csv: includes time, machine ID, temp, defect count per batch
24	Define sampling decision logic	✅	Sampling % varies based on AI output + batch risk profile
25	Visualize sample dashboard	✅	Simple table/chart: sampling vs defect trend, auto-adjusted logic
26	Show feedback effect	✅	Display how system learns and modifies sampling over time (light RL loop)
27	Prepare QR/link to view data	🟡	Optional: link or QR to CSV file or dashboard (on poster or slide)

PHASE 1 PROBLEM UNDERSTANDING

1: UNDERSTANDING THE BRIEF

We have carefully analyzed the competition brief and broken down each requirement to clarify what the project must deliver. Below is the detailed analysis:

A. Breakdown of Challenge Requirements

Requirement	**Our Solution**
Smart AI	Use lightweight, explainable models (YOLOv8n / MobileNet) on Jetson/RPi devices
Smart Sampling	Sampling rate is dynamically decided based on defect severity + batch history
Production Monitoring	Data linked to machine ID, shift, and stage for traceability
Failure Prediction	LightGBM learns defect trends from sensor data (optional feedback loop)
Machine Learning Models	YOLOv8n (image), LightGBM (sensor); all optimized for edge deployment
Real-Time Computer Vision	Camera input processed locally without cloud dependency
Forensic Traceability	Defect is mapped back to process stage/machine → enables root cause insight
Cost-Aware Optimization	Sampling adjusted based on inspection cost vs. defect risk trade-off

Industry Pain Points – Real Undetected Defects

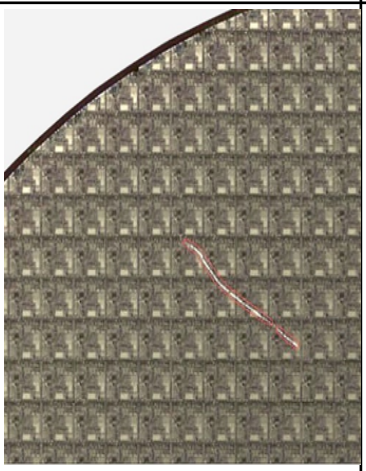
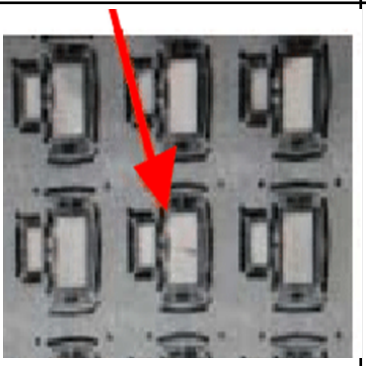
Wafer Surface Scratch	IC Die Crack	PCB Solder Bridge
		
Surface scratch undetected by AQL sampling	Die crack after molding – invisible to human inspection	Solder bridge – causes shorts, missed under random checks

Image Source Attribution (for educational)

Defect Type	Suggested Citation
Wafer Scratch	*Source: Microtronic Inc.*
IC Die Crack	*Source: ResearchGate – TFBGA Study*
PCB Solder Bridge	*Source: TechSparks Blog*

Many critical semiconductor defects are still **missed under traditional AQL-based random checks**, especially during Final Visual Inspection (FVI). These missed defects lead to downstream failures, costly returns, and damage to brand reputation.

B. HARDWARE: SMART IoT INSPECTION SYSTEM

Objective : Design a **low-cost, modular, and scalable smart inspection unit** that:

- Attaches directly to production lines (non-intrusive)
- Captures real-time product images
- Processes data locally using edge AI (no internet/cloud)
- Scales easily with multiple nodes across stages or machines

Key Components & Setup

Component	Example Hardware	Role
Camera Module	RPi Camera / ESP32-CAM / USB Webcam	Captures high-res wafer or PCB images
Edge Processor	Jetson Nano / Raspberry Pi 4	Runs YOLOv8n / MobileNet for defect detection
Connectivity	Wi-Fi / LAN / Local broadcast	Transmits decisions or syncs to optional dashboard
Power & Housing	Battery / adapter + 3D-printed case	Enables portable, factory-safe deployment
Scalability	Modular unit, plug-and-play ready	Add nodes per stage / line without system redesign

Cost & Deployment Advantage

- Target cost: < $100 per node
- No cloud dependency: Works offline, privacy-safe
- Fast deployment: Plug-and-play setup, no IT overhaul
  Factory fit: Designed for small-medium fabs in ASEAN & APAC

System Flow Summary

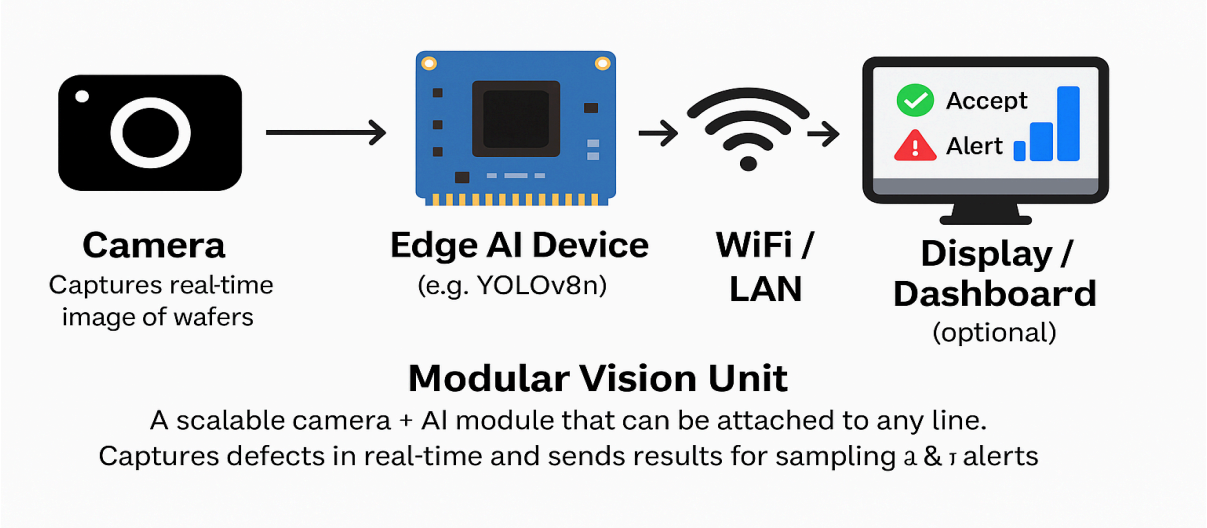
[Camera] → [Edge AI (YOLOv8n)] → [Sampling Decision Logic] → [Dashboard / Output]

Modular Vision Node

Each node acts as a self-contained inspection unit that scans, detects, and decides in real-time — reducing reliance on centralized control and allowing localized decisions per stage.

Figure: Smart IoT Hardware Architecture for Real-Time Defect Inspection

This diagram illustrates the hardware flow from camera modules capturing real-time images, through edge AI devices for on-site processing, to optional cloud gateways and output layers. The modular design enables flexible deployment and scalable expansion across production lines.



C. SOFTWARE: AI / MACHINE LEARNING – THE SYSTEM'S BRAIN

🎯 **Goal:** Design a lightweight, explainable AI system that:

- Detects defects in real-time from product images
- Classifies defect severity: Critical / Major / Minor
- Recommends action: Accept / Resample / Reject
- (Optional) Learns and adjusts sampling logic from sensor/batch data

AI Model Setup (Simplified)

Application	**Model(s) Used**	**Why It's Used**
Image Defect Detection	YOLOv8-nano / MobileNetV3	⚡ Fast, small, runs on edge (RPi/Jetson)
Sensor Data (Optional)	LightGBM / Random Forest	📊 Handles structured, small sensor datasets
Sampling Decision Logic	Rule-based + Adaptive Threshold	✅ Explainable, flexible, and factory-trainable

Image Input → AI Detection → Defect Type + Severity
↓
Smart Sampling Logic → Action: Accept / Resample / Reject
↓
Output: Dashboard + Local Data Storage

Why This Works for Real Factories

- No cloud or GPU needed – runs fully on edge
- Transparent decision-making – not a black box
- Modular logic – easy to adjust, scale, or expand
- Ready for **feedback learning loop** (e.g. LightGBM learns from batch outcome)

2. Mapping the Semiconductor Process Chain:

Strategic Framing for Smart Sampling Insertion

Semiconductor manufacturing includes multiple critical stages — but not all stages are equally suitable for AI-powered smart sampling. By analyzing the full process, we identify where innovation delivers the greatest impact.

Overview of the Production Chain (Simplified)

Stage	Main Operation	QC Method	AI-Sampling Fit
[1] Front-End Fabrication	Build layers on wafer using lithography	Sensor-based, yield tracking	❌ Low visibility
[2] Wafer Inspection & Dicing	Cut into dies, inspect basic defects	Manual inspection	⚠️ Limited sampling
[3] Back-End Packaging	Bonding, molding, encapsulation	Visual/manual + test chips	⚠️ Medium fit
[4] Final Visual Inspection (FVI)	Final surface check before release	AQL % sampling + manual check	✅ Ideal target

Why Final Visual Inspection (FVI) is the Strategic Insertion Point

- 🔍 Defect visibility is highest: Scratches, cracks, bridges → directly seen via image
- 🧍 Human visual checks dominate → inconsistent, error-prone
- 📉 AQL is outdated: random %, no learning, blind to process history
- 🚫 No structured data → can't train AI models on existing setups
- 💥 Highest impact per dollar: One AI camera node at FVI saves most risk per cost

Only FVI gives full access to image + batch + defect outcome in one place.

Conclusion:
Final Visual Inspection is the only point in the chain where:

- Defect risk is high
- AI + sampling + traceability logic can be deployed together
- Structured feedback allows learning and optimization loop

This makes it the ideal entry point for launching Smart Sampling and evolving toward Autonomous Quality Intelligence (AQIS).

3. Identifying the Quality Control Bottleneck

Why Final Visual Inspection Is the Ideal Bottleneck to Disrupt

Traditional Weaknesses of Final Visual Inspection (FVI)

Issue	**Impact**
AQL is random sampling	High-risk defects are missed (blind spots not adaptive)
Manual visual inspection	Slow, labor-heavy, and subjective → leads to inconsistent quality
No data retention	Cannot track which stage/machine caused a defect → zero forensic traceability
No adaptive learning	Every batch is treated the same regardless of risk
No cost optimization logic	Resources wasted on over-inspecting good batches, under-inspecting risky ones

What Makes FVI the "Perfect Bottleneck"

1. Defects are still visible → camera-based AI can detect them live
2. Human/manual limits → ready for automation
3. No learning exists → ideal place to deploy AI that learns
4. One point, multiple data types available → enables full-tier deployment
5. Failure here = customer-facing defect → critical business impact

Summary Insight

Final Visual Inspection is not just a bottleneck — it is a **strategic leverage point**:

- It's where **manual QC fails the most**

- It’s where **AI can outperform humans**
- And it's where the **highest ROI** from smart automation can be achieved

That’s why we place our Smart Sampling AI system **here first**, before expanding upstream.

4. Analyzing the Limits of AQL Sampling

Why AQL Cannot Support Modern Quality Demands

❌ What is AQL?

AQL (Acceptable Quality Level) is a decades-old QC method that inspects only a fixed percentage of products in each batch, regardless of actual risk or defect trends.

It was designed for manual inspection in low-variability industries — not for today’s high-speed, data-rich semiconductor lines.

⚠ **Critical Limitations of AQL**

Limitation	**Explanation**
Random sampling	Misses rare but critical defects (e.g. cracks, bridges)
No link to production parameters	AQL doesn’t consider machine stage, temperature, time-of-day, etc.
Fixed inspection effort	Cannot scale up for high-risk batches, or down for clean runs
No feedback or learning	Same sampling rules used even after multiple escapes
No root cause traceability	Cannot connect defects back to upstream machine/stage
Manual & subjective	Depends on human inspectors → inconsistent quality

Consequences of Relying on AQL

- Missed defects → customer returns, product recalls
- Repeated mistakes → no way to learn from history
- High cost per inspection → even when no defects found
- Not scalable → especially for **high-volume fabs in ASEAN**

Strategic Insight

AQL is not just inefficient — it is structurally incapable of supporting real-time, learning-driven quality control.
Replacing AQL with AI-powered Smart Sampling is not just an improvement — it's an evolution.

5-6. Business Pain Points & Industry Innovation Gap

Why Solving This Problem Delivers Strategic ROI

💸 **Business Pain Points**

Problem	**Business Impact**
Missed defects (due to AQL)	Costly returns, customer rejection, brand damage
Manual inspection dependency	Labor shortages, fatigue, inconsistency
High inspection cost per good unit	Over-inspection of clean batches → wasted resources
No data for improvement	Defects repeat across batches without detection → long-term yield loss
Poor traceability	Unable to pinpoint fault stage → costly re-inspection or over-testing

🏭 **Industry Status (Benchmark)**

Current Practice	**Limitation**

AQL Sampling	Blind to process history, fixed logic, no learning
Mid-line AOI (some tier-1 fabs)	Applied at wafer level; FVI remains manual in many sites
Manual FVI in many ASEAN fabs	No data capture, no logic, fully dependent on inspectors
Lack of real-time quality logic	Factories can’t react dynamically to live defect signals

🚀 **Innovation Gap = Our Opportunity**

The current industry state has created a **gap between detection and decision-making.**

Our solution bridges this gap by integrating:

- Real-time AI detection
- Smart sampling logic
- Feedback learning
- Traceability
- Cost-aware optimization

This creates not just a better QC tool — but a **scalable intelligence layer** that fits ASEAN fabs and global production.

PHASE 2:SOLUTION DESIGN

7. DESIGN OVERALL AI + SMART SAMPLING SYSTEM

System Architecture to Replace AQL with Real-Time AI Decision Engine

🧠 Objective

Build an integrated system that replaces AQL with smart, adaptive sampling, powered by lightweight AI. The system must be:

- Real-time
- Modular & low-cost
- Transparent (not black box)
- Deployable in production fab conditions
  Scalable across lines or stages

🧩 System Layers (Strategic Mapping to 9 Tiers)

Layer	**Function**	**Tier Activation**
Image Capture	Camera module captures real-time product surface images	Base layer (input to Tier 0)
AI Defect Detection	YOLOv8n runs locally to detect defect presence & type	Tier 0
Sampling Engine	Rule + history + severity → decide Accept / Resample / Reject	Tier 1–2
Learning Logic	LightGBM updates sampling strategy based on trends	Tier 3–5
Traceability Map	Match defects to machines/stages via timestamp/sensor log	Tier 3–4
Cost Optimization	Adjusts sample % based on defect risk vs inspection cost	Tier 6
Flow Control	Pause/resample flow if risk spikes	Tier 4

🔁 Core Decision Flow

[Image Capture]

↓

[YOLOv8n Defect Detection]

↓

[Sampling Engine Logic]

↓

Decision: Accept / Resample / Reject

↓

[Dashboard + Feedback Learning (Optional)]

What Makes This System Smart

- **Real-time**: Sampling decision per unit, not per batch
- **Learning-enabled**: Updates based on historical defect patterns
- **Traceable**: All outputs linked to timestamp, machine, batch
- **Cost-aware**: Does more when risk is high, less when safe
- **Scalable**: More nodes = more coverage; logic stays consistent

[Camera] → [AI Edge Box] → [Smart Logic + Learning]

8. SELECT HARDWARE PLATFORM

Optimized for Edge AI, Low Cost, and Scalable Deployment

Objective

Choose a hardware setup that allows the entire smart sampling system to run:

- Locally (no cloud needed)
- On low-power, edge-ready devices
- With cost < **$100 per node**
- Modular and scalable across stages

Recommended Hardware Stack

Component	Suggested Model	Function
Camera Module	RPi Camera / ESP32-CAM / USB Webcam	Captures wafer or PCB surface images
Edge Processor	Raspberry Pi 4 / Jetson Nano	Runs YOLOv8n + Smart Sampling engine locally
Power Unit	Rechargeable battery or adapter	Enables portability, factory-safety compliant
Communication	Wi-Fi / LAN / local data broadcast	Sends summary to dashboard (optional)
Housing	3D-printed casing / aluminum shell	Protects against dust, vibration, and heat

Why It Works for Real Fabs

Feature	Strategic Advantage
Cost-effective (< $100)	Suitable for ASEAN fabs, pilot rollout, SME factories
No cloud/GPU dependency	Runs fully offline, avoids privacy and latency risks
Modular setup	Easily added to different stages or lines
Plug-and-play deployment	Fast setup, no deep IT infrastructure needed

System Deployment Vision

> “A camera + brain in a box”
> → Just mount it on the line, power it, and it starts learning and deciding.

Each hardware unit becomes a local quality agent — able to inspect, decide, and contribute to global learning without needing to ask the cloud.

9. DEVELOP AI MODEL

Real-Time Defect Detection with Lightweight Vision AI

Objective

Build and deploy a computer vision model that can:

- Detect key defect types (scratches, cracks, solder bridges...)
- Run fully on low-cost edge devices (Jetson Nano / RPi 4)
- Offer fast inference + high explainability
- Enable traceable + learnable sampling logic

Model Selection Strategy

Task	**Model Chosen**	**Why This Model**
Image Defect Detection	YOLOv8n / MobileNetV3	⚡ Ultra-lightweight, <10ms inference on Jetson Nano
Severity Estimation	Internal confidence logic	Used to trigger sampling rules (Critical/Major/Minor)
Batch Learning	LightGBM / RF (optional)	Works well with tabular + small sample data

Model Input / Output Logic

Input	**Model Output**
Image (real-time, grayscale/color)	Defect location, class, confidence %
Timestamp, batch ID (metadata)	Stored with result for learning + traceability

How the Model Connects to the System

```
Image Input → YOLOv8n →
  → If (confidence > X%) → Flag: Defect
  → → Sampling Logic (accept / resample / reject)
  → → Batch history updated
```

Why YOLOv8n Is the Best Fit

- 🔋 Runs on low power: Suitable for edge processors like RPi/Jetson
- ⚡ Fast enough: Near real-time inference
- 🔍 Transparent: Explainable bounding boxes + confidence scores
- 🔧 Tunable: Thresholds can be adapted per factory/batch

10. SAMPLING ENGINE LOGIC

From Static AQL to Adaptive, AI-Driven Sampling Decisions

Objective

Design a smart sampling engine that replaces fixed-percentage AQL with:

- Rule-based + data-driven logic
- Defect-aware & severity-aware decisions
- Adaptive behavior per batch, stage, or risk profile

🧠 **Logic Structure Overview**

Input	Decision Logic
Defect type (from YOLOv8n)	Severe defect → Always reject / resample
Confidence level (%)	Higher confidence → stronger decision trigger

Batch defect history (past 10 runs)	Recent spikes → increase sampling rate
Process stage or machine (optional)	Known unstable stages → apply stricter sampling
Inspection cost (configurable)	Cost-aware filter: balance sample % vs budget

Action Outcomes

Condition Met	**Action**
Defect detected (critical/major)	Reject OR Resample more
Minor defect + low batch risk	Accept with log
No defect + clean batch history	Accept with reduced sampling rate
Sudden spike in defect rate	Trigger flow pause / full inspection
Stage flagged (via sensor traceability)	Increase sampling rate for that stage

Built-in Learning & Feedback

After every batch, the system:

- Logs sampling outcomes
- Feeds data into LightGBM (or similar)
- Learns batch-level risk patterns
- Adjusts future sampling % automatically

🧩 Mapping to 9-Tier Strategy

Tier	**Sampling Engine Role**

Tier 1	**Decision logic replaces AQL**
Tier 2	**History-based learning integrated**
Tier 4	**Flow-control logic: pause, intensify, or skip sampling dynamically**
Tier 5–6	**Learns to optimize inspection vs cost trade-off**

```
[Defect Input + History + Stage]
          ↓
[Sampling Engine Logic: Accept / Resample / Reject]
          ↓
[Feedback to Learning Engine + Dashboard]
```

11. ADD LEARNING MECHANISM (FEEDBACK LOOP)

Lightweight Learning That Makes the System Smarter Over Time

ObjectiveCreate a feedback mechanism where the system can:

- Learn from inspection outcomes over time
- Detect hidden defect trends across batches/machines
- Automatically fine-tune future sampling logic

How the Feedback Loop Works

Step	Action Taken
1. After sampling decision	Result (defect type, confidence, action taken) is logged
2. Batch-level summary created	% defects, stage, timestamp, sensor readings aggregated

3. Feed into LightGBM model	Predicts next batch risk score (e.g., "Batch likely high-defect")
4. Sampling % adjusted	Based on predicted risk and cost profile
5. Loop continues per batch	System keeps refining logic every few batches

Why LightGBM?
✅ **Handles small tabular datasets well**
✅ **Fast to retrain on edge**
✅ **Easy to explain & trace logic**

Learning Inputs

- Batch ID, timestamp
- Number & severity of defects detected
- Sampling rate used
- Sensor metadata (optional: temp, machine ID)
- Operator overrides (if any)

Output: Risk Score or Sampling Boost %

Example:
"Batch 1278 → High risk predicted (0.78) → Boost sampling from 20% → 80%"

Strategic Tier Mapping

Tier	**Feedback Function**
Tier 2	Batch-level memory starts building (vs AQL = forgets everything)
Tier 3	Traceability + history = AI feedback context

Tier 5	Sampling logic becomes personalized per fab / line / batch

12. INTEGRATE FORENSIC TRACEABILITY

Linking Defects Back to Stages, Machines, and Root Causes

Objective

Enable the system to **not just detect defects**, but **map them back** to the stage or machine where they most likely originated, using:

- Timestamps
- Sensor log correlation
- Defect class patterns
- Batch or shift metadata

How Traceability Works

Captured Data	**Used To**
Timestamp (from each image)	Align defect with production time
Machine/stage ID (metadata)	Identify process step responsible
Defect type & class	Spot recurring defect patterns linked to specific stages
Sensor readings (temp, speed)	Detect environmental anomalies causing yield loss
Batch/shift info	Link operational variance to defect trends

Example Scenario

- 4 consecutive defects → same shift + same machine
- System flags that stage as “risk elevated”
- Sampling logic increases sample % only for that stage
- Supervisor notified to inspect machine condition

System Implementation

- Each defect log entry includes:
    - Defect class
    - Image timestamp
    - Stage/machine label
- These are **cross-matched** with sensor logs via `sensor_log.csv`

➡ No need for complex MES; lightweight logs are enough for local traceability.

Output Actions

- Adjust sampling only at the problem stage
- Provide insights for preventive maintenance
- Reduce unnecessary sampling in stable stages

Strategic Tier Mapping

Tier	**Traceability Power**
Tier 3	Defect logs mapped to machine/stage = root cause analysis
Tier 4	System can “divert flow” or “resample” based on problem stages

13. COST-AWARE OPTIMIZATION

Sampling Smarter, Not Just Harder – AI with Economic Intelligence

Objective

Build logic that allows the system to **adjust sampling intensity** based on:

- Defect risk level
- Inspection cost per unit
- Business-criticality of the product
- Factory resource constraints

➡ Sampling becomes **risk-based AND cost-balanced**, not just “more is better.”

Key Input Variables

Factor	Impact on Sampling Decision
Defect risk score (from AI)	High risk → increase sampling
Inspection cost per unit	High cost → avoid unnecessary oversampling
Product type/value	High-value items → allow higher inspection budget
Stage failure cost	Expensive-to-rework stages → increase checks there
Operator/machine load	May trigger defer sampling if line is over-capacity

Decision Logic Example(python)

```
if (Defect_Risk * Failure_Cost) > Inspection_Cost:
    Increase_Sampling()
else:
    Decrease_Sampling()
```

=> This ensures **every extra inspection effort is economically justified**.

Sample Scenario

Condition	System Response
Batch = low-risk, low-value	Sampling reduced to minimum
Stage = high-failure cost (e.g. bonding)	Sampling rate boosted selectively

Shift = night (higher defect odds)	Sampling increased for that shift only

Strategic Tier Mapping

Tier	How AI Thinks Like a Factory Manager
Tier 6	Sampling becomes cost-risk optimized, not just quality driven
Tier 7	AI schedules or adjusts resources based on risk-to-cost ratio

Why It Matters in Real Production

- 💰 Reduces wasteful over-sampling in clean runs
- ⚖️ Ensures critical errors are caught early — without overloading inspection
- 📈 Moves beyond "defect rate" to **ROI-based quality strategy**

PHASE 3: CORE TECHNOLOGY

14. CHOOSE DEFECT DETECTION MODEL

Fast, Lightweight, and Explainable Computer Vision for Real-Time QC

Objective

Select and fine-tune a defect detection model that:

- Runs on edge hardware (RPi, Jetson) without cloud/GPU
- Detects visual surface defects in real-time
- Produces explainable outputs (class, box, confidence)
- Enables immediate sampling decisions

Model Selected: YOLOv8n (You Only Look Once – Nano)

Why YOLOv8n?	How it fits our system

⚡ Fast inference (<30ms)	Real-time processing on Jetson Nano
🧠 Lightweight model size	Edge-compatible (runs on RPi 4 with reduced FPS)
📦 Outputs: class + box + confidence	Used directly in sampling logic
🧰 Easy to retrain with custom images	Fits different fabs and defect types
💬 Widely supported, explainable, open source	No license cost, traceable decisions

Model Input & Output Structure

Input	**Output**
Image (480×480 or 640×640)	Detected bounding boxes + Defect class (scratch, crack, etc.) + Confidence %

Training / Fine-Tuning Strategy

- Start with pre-trained YOLOv8n
- Fine-tune using ~200–500 labeled defect images (open source + fab samples)
- Use image augmentation (blur, rotate, contrast) to improve generalization
- Target mAP > 0.75 for 3–4 core defect types

Deployment Strategy

Mode	**Specs**	**Use Case**
Jetson Nano	Full model (real-time 15–20 FPS)	Production-ready nodes

RPi 4	Quantized model (slow FPS, fallback mode)	Low-cost or proof-of-concept run

Tier Mapping (System Integration)

Tier	**YOLOv8n Contribution**
Tier 0	Visual defect detection layer
Tier 1	Triggers Smart Sampling Engine based on outputs
Tier 2	Feeds historical detection data into learning loop

Summary Insight

YOLOv8n is the optimal balance of performance, simplicity, and deployability.
It turns a <$100 device into an AI-powered visual inspector, replacing AQL with intelligence.

15. INTEGRATE AI WITH SAMPLING ENGINE

From Detection to Decision – Making AI Actionable

Objective

Link the output of the AI model (YOLOv8n) with the Smart Sampling Engine, so that every detection leads to an immediate, explainable decision:

- Accept
- Resample
- Reject

…and allow this decision to be learned from and refined over time

Integration Logic Flow

```
YOLOv8n output → [Defect class, location, confidence %]
        ↓
Sampling Engine evaluates:
    - Severity of defect
    - Model confidence
    - Batch history
    - Risk level
        ↓
Decision: Accept / Resample / Reject
        ↓
Log + Feedback → Learning Engine
```

Core Interaction

AI Output	**Sampling Engine Use**
Class = “scratch”, Confidence = 92%	Triggers resample (critical severity + high certainty)
Class = “minor dot”, Confidence = 61%	Accept or log-only (low risk + low confidence)
No detection	May reduce sampling % next batch (based on trend)

Logic Mapping Strategy

YOLO Output Variable	**Sampling Decision Variable**

Class label	Severity → impacts Reject vs Resample
Confidence (%)	Threshold-based trigger
Bounding box location	Optional visual overlay (dashboard/debug)
Detection count/batch	Contributes to batch-level risk scoring

Real-Time Integration Goals

- **Every image processed = 1 decision made**
- **Sampling engine always "sees" the AI output before deciding**
- **Logic is explainable & tunable (no black box)**
- **Results are logged for traceability & learning**

Tier Mapping (System Cohesion)

Tier	**Function at This Integration Point**
Tier 1	AI feeds actionable data into the Sampling Engine
Tier 2	Batch-level learning integrates into decision context
Tier 3	All decisions are traceable by timestamp, class, batch ID

Key Insight

"In traditional QC, detecting the problem is separate from responding to it.
In our system, **detection and decision are one integrated, intelligent step.**"

17. IMPLEMENT FEEDBACK LEARNING

Let the Sampling Engine Learn and Adapt Over Time

Objective

Integrate a lightweight learning module that:

- Learns from previous batch outcomes
- Adjusts sampling decisions based on trend and context
- Works even with limited data (no deep learning required)
- Enhances rule logic with batch-level risk prediction

Learning Flow Overview

1. **After each batch**
   → Log defect count, severity, machine/stage, shift, sampling %

2. **Aggregate recent batches**
   → Create structured dataset (e.g., `sensor_log.csv`)

3. **Train LightGBM or Random Forest model**
   → Predict batch-level defect risk (0–1)

4. **Use prediction in future sampling logic**
   → Higher risk → higher sample %
   → Lower risk → conserve inspection effort

5. **Repeat every N batches** (e.g., every 10)

Model Characteristics

Model	**Why Chosen**
LightGBM	Fast training, low memory, explainable, tabular-ready
Random Forest	Easy to debug, robust to small data

Sample Learning Features

Feature	Source
Avg. defects last 5 runs	ai_outputs.csv / system log
Stage ID (encoded)	batch metadata
Shift time (day/night)	timestamp
Last sample %	previous system decision
Machine ID	metadata / sensor input

➡ Output: `Defect_Risk_Score` (e.g., 0.82)

Integration with Sampling Logic

```
if Defect_Risk_Score > 0.75:
        SampleRate = 80%
elif Defect_Risk_Score < 0.3:
        SampleRate = 10%
```

Tier Mapping

Tier	Learning Role
Tier 2	System now remembers trends across batches
Tier 3	Learns contextual risk based on machine/stage/shift
Tier 5	Sampling becomes a self-tuning training pipeline for AI

Summary Insight:

AQL never learns.
Our engine does — **and gets better with every batch.**

18. ADD ECONOMIC OPTIMIZATION LAYER

Smarter Sampling That Understands Cost and Value

Objective

Make sampling decisions **cost-aware**, so the system can:

- Balance inspection effort vs risk of defect escape
- Allocate sampling budget more effectively
- Prioritize high-value products and high-failure-cost stages
- Avoid wasting resources on over-checking clean, low-risk units

Core Optimization Logic(python)

```
If (Defect_Risk × Failure_Cost) > Inspection_Cost:
        Sample_More()
Else:
        Sample_Less()
```

➡ Sample more only when it actually saves the factory money

➡ Avoid sampling fatigue on clean runs

Key Parameters:

Variable	Definition	Source
Defect_Risk	Predicted by LightGBM or recent defect trend	Feedback engine
Failure_Cost	Cost if defect escapes (depends on stage/product)	Configurable per fab

Inspection_Cost	Cost of 1 inspection (human or machine)	Factory-estimated
Product Value	High-value unit → higher sample threshold	Factory-defined class
Stage Rework Cost	Higher at back-end stages (e.g., post-bonding)	Traceability data

Example Decision Table

Risk	**Product Type**	**Failure Cost**	**Inspect Cost**	**Action**
High	Critical chip	High	Low	Sample 100%
Low	Generic PCB	Low	Medium	Sample 5%
Mid	Moderate item	Medium	Low	Sample 30%

Tier Mapping

Tier	**Role in Cost Optimization**
Tier 6	Sampling engine becomes ROI-aware, not just accuracy-driven
Tier 7	Supports dynamic scheduling or budget allocation based on cost risk

Strategic Insight

- Reduces cost of unnecessary inspections
- Shifts effort toward high-risk, high-impact areas
- Helps fab managers **justify deployment ROI**

- Opens door to economic dashboards or policy tuning

19. ENABLE MULTI-SOURCE DATA INPUT

Integrating Images, Sensors, and Metadata into a Unified Decision Engine

Objective

Design the system to **consume multiple input types**, enabling it to:

- Make smarter sampling decisions with full production context
- Correlate defects with shift, machine, temperature, etc.
- Build a more accurate learning model (LightGBM)
- Enable future integration with MES or IoT platforms

Types of Inputs

Input Type	**Source Example**	**Usage in System**
Image (RGB / IR)	Camera module	Visual defect detection (YOLOv8n)
Timestamp	Auto-generated	Batch traceability, shift analysis
Machine/Stage ID	Manual input or barcode reader	Root cause traceability, localized sampling logic
Sensor Data	Temp, humidity, vibration sensors	Detect machine instability → adjust sampling
Operator Logs	Optional manual input	Override, exception handling

Integration Flow

```
[Image] + [Sensor/Meta Input]
        ↓
[Defect Detection + Batch Context]
        ↓
[Smart Sampling Logic]
        ↓
Decision + Feedback Learning
```

Example Usage

Scenario	System Response
Defects spike only during night shift	Increase sampling % for shift B
Vibration sensor spikes on Machine 3	Add resample logic only for Stage 3
Heat map shows Stage 2 = high error zone	Adjust batch routing or alert supervisor

🧠 Learning Enhancement

Multi-source input → richer dataset for LightGBM → better defect risk prediction.

> "The more context we give the system, the better it protects quality."

Tier Mapping

Tier	How Multi-Input Enhances Intelligence
Tier 3	Links defect patterns to machine/stage/shift
Tier 4	Enables targeted intervention (pause/resample specific line)
Tier 5	Learning is based on full context, not just images