# Manufacturing Quality Control AI Vision System

[Python](https://www.python.org/) [TensorFlow](https://www.tensorflow.org/) [FastAPI](https://fastapi.tiangolo.com/)

An AI-powered computer vision system for automated defect detection in manufacturing assembly lines.

## 🚀 Features

* **Real-time Defect Detection**: Identify various types of defects (scratches, dents, color variations) in real-time
* **Deep Learning Models**: Utilizes both CNN and Vision Transformer architectures for accurate defect classification
* **Operator Interface**: Web-based dashboard for real-time monitoring and alerts
* **Quality Analytics**: Statistical process control and trend analysis
* **Continuous Learning**: Feedback loop for model improvement with human validation

## 🛠️ Installation

1. **Clone the repository**

* git clone https://github.com/yourusername/manufacturing-ai-vision.git  
  cd manufacturing-ai-vision

1. **Create and activate a virtual environment**

* python -m venv venv  
  source venv/bin/activate # On Windows: venv\Scripts\activate

1. **Install dependencies**

* pip install -r requirements.txt

1. **Set up environment variables**

* cp .env.example .env  
  # Edit .env with your configuration

## 🚀 Quick Start

1. **Prepare your dataset**
   * Place your training images in data/raw/train/
   * Organize images in subdirectories by class (e.g., defect\_type1/, defect\_type2/, good/)
2. **Train the model**

* python src/models/train.py --data\_dir data/raw --epochs 50 --batch\_size 32

1. **Start the API server**

* uvicorn src.api.main:app --reload

1. **Launch the web interface**

* cd frontend  
  npm install  
  npm run dev

## Database Setup

### Prerequisites

* PostgreSQL 13+ installed and running
* Python 3.8+
* pip package manager

### Initial Setup

1. Create a new PostgreSQL database:

* createdb quality\_control

1. Copy the example environment file and update it with your database credentials:

* cp .env.example .env
* Edit the .env file and update the database connection string and other settings as needed.

1. Set up the development environment and install dependencies:

* chmod +x setup\_dev.sh  
  ./setup\_dev.sh
* This will:
  + Create a Python virtual environment
  + Install all required dependencies
  + Set up pre-commit hooks
  + Create necessary directories

1. Initialize the database and run migrations:

* source venv/bin/activate  
  python scripts/setup.py
* This will:
  + Create the database if it doesn’t exist
  + Run all database migrations
  + Seed the database with initial data

### Database Management

* **Run migrations**:
* python scripts/migrate\_db.py
* **Create a new migration**:
* alembic revision --autogenerate -m "description of changes"
* **Reset the database** (development only):
* dropdb quality\_control  
  createdb quality\_control  
  python scripts/setup.py

## Project Structure

manufacturing-ai-vision/  
├── data/ # Data storage  
│ ├── raw/ # Raw image data  
│ ├── processed/ # Processed data  
│ └── models/ # Trained models  
├── src/ # Source code  
│ ├── api/ # FastAPI application  
│ ├── models/ # Model definitions  
│ ├── preprocessing/ # Data preprocessing  
│ └── utils/ # Utility functions  
├── notebooks/ # Jupyter notebooks for exploration  
├── config/ # Configuration files  
├── frontend/ # Web interface  
└── deploy/ # Deployment configurations

## 📊 Model Performance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Accuracy | Precision | Recall | F1-Score |
| CNN | 98.2% | 97.8% | 98.1% | 97.9% |
| ViT | 98.5% | 98.2% | 98.4% | 98.3% |

## 🤝 Contributing

1. Fork the repository
2. Create your feature branch (git checkout -b feature/AmazingFeature)
3. Commit your changes (git commit -m 'Add some AmazingFeature')
4. Push to the branch (git push origin feature/AmazingFeature)
5. Open a Pull Request

## 📄 License

This project is licensed under the MIT License - see the <LICENSE> file for details.

## ✨ Contributors

* Your Name [your.email@example.com](mailto:your.email@example.com)

## Problem Statement

Manufacturing companies face challenges in maintaining consistent product quality while managing inspection costs and speed. Your task is to develop a computer vision AI system that can identify various types of defects (scratches, dents, color variations, dimensional issues) in real-time during production. The system should integrate with existing manufacturing equipment, provide instant feedback to operators, and maintain detailed quality metrics for continuous improvement.

## Steps

• Design a computer vision pipeline using CNNs or Vision Transformers for defect detection • Implement real-time image processing capabilities for assembly line integration • Create a classification system for different defect types with confidence scoring • Build an operator interface showing real-time quality status and defect locations • Develop a feedback loop system for continuous model improvement with human validation • Include statistical quality control charts and trend analysis capabilities

## Suggested Data Requirements

• High-resolution images of products (both defective and non-defective samples, 1000+ each category) • Defect classification labels and severity ratings • Production line metadata (timestamps, batch numbers, operator IDs)

## Themes

AI for Industry, Classical AI/ML/DL for prediction

### 1. Product Overview

**Product Name**: VisionQC AI Manufacturing Inspector  
**Version**: 1.0  
**Target Market**: Mid to large-scale manufacturing companies, automotive, electronics, pharmaceuticals

### 2. Business Objectives

* **Primary**: Reduce quality control costs by 30-40% while improving defect detection accuracy to 99.5%+
* **Secondary**: Decrease inspection time by 80%, reduce human error to <0.1%, improve product consistency by 95%
* **ROI Target**: 400% within 18 months through cost savings and quality improvements

### 3. Target Users

* **Primary**: Quality Control Operators, Production Line Supervisors
* **Secondary**: Quality Managers, Manufacturing Engineers, Plant Managers
* **Technical**: Computer Vision Engineers, IT Operations, Maintenance Teams

### 4. Key Features

#### Core Capabilities

* Real-time computer vision defect detection (scratches, dents, color variations, dimensional issues)
* Multi-defect classification with confidence scoring and severity assessment
* Assembly line integration with existing manufacturing equipment
* Instant operator feedback with defect location highlighting
* Continuous learning with human validation feedback loop

#### User Interface

* Real-time quality dashboard with live inspection status
* Defect visualization with annotated images and location mapping
* Statistical quality control charts and trend analysis
* Mobile alerts for critical quality issues
* Automated reporting and quality metrics generation

### 5. Success Metrics

* **Quality**: Defect detection accuracy ≥99.5%, false positive rate <1%
* **Performance**: Real-time processing <100ms per image, 99.9% uptime
* **Business Impact**: 30-40% cost reduction, 80% faster inspection, 95% consistency improvement
* **User Adoption**: 95% operator acceptance, <2 hours training time

### 6. Constraints & Assumptions

* Integration with existing manufacturing equipment and PLCs required
* High-resolution camera systems available or can be installed
* Minimum 1000+ labeled images per defect category for training
* Real-time processing requirements with millisecond latency constraints
* Compliance with manufacturing safety and regulatory standards # Functional Requirements Document (FRD) ## Manufacturing Quality Control AI Vision System

*Building upon PRD requirements for detailed functional specifications*

## ETVX Framework

### ENTRY CRITERIA

* ✅ PRD completed and approved by manufacturing stakeholders
* ✅ Business objectives and success metrics clearly defined
* ✅ Target users and their operational needs documented
* ✅ Key product features identified and prioritized
* ✅ Technical feasibility assessment for computer vision completed

### TASK

Transform PRD business requirements into detailed, testable functional specifications that define exactly what the AI vision system must do, including image processing workflows, defect detection algorithms, user interactions, system behaviors, and manufacturing equipment integration requirements.

### VERIFICATION & VALIDATION

**Verification Checklist:** - [ ] Each functional requirement is traceable to PRD business objectives - [ ] Requirements are unambiguous and testable with specific acceptance criteria - [ ] All operator workflows are covered end-to-end - [ ] Integration points with manufacturing equipment and PLCs defined - [ ] Error handling and edge cases specified for production environment - [ ] Requirements follow consistent numbering (FR-001, FR-002, etc.)

**Validation Criteria:** - [ ] Requirements satisfy all PRD success metrics (99.5% accuracy, <100ms processing) - [ ] Operator personas can achieve their quality control goals through defined functions - [ ] System behaviors align with manufacturing quality standards - [ ] Computer vision team confirms implementability of all detection requirements - [ ] Requirements review completed with manufacturing and quality stakeholders

### EXIT CRITERIA

* ✅ All functional requirements documented with unique identifiers
* ✅ Requirements traceability matrix to PRD completed
* ✅ User acceptance criteria defined for each requirement
* ✅ Manufacturing integration requirements clearly specified
* ✅ Foundation established for non-functional requirements development

### Reference to Previous Documents

This FRD translates the business objectives and product features defined in the **PRD** into specific functional requirements: - **PRD Target Users** → Detailed operator interface and workflow requirements - **PRD Key Features** → Granular computer vision and defect detection specifications  
- **PRD Success Metrics** → Measurable functional capabilities (99.5% accuracy, <100ms processing) - **PRD Constraints** → Technical integration and compliance requirements

### 1. Computer Vision Processing Module

#### 1.1 Image Acquisition and Preprocessing

* **FR-001**: System SHALL capture high-resolution images (minimum 2048x2048 pixels) from assembly line cameras
* **FR-002**: System SHALL support multiple camera types (RGB, infrared, depth sensors) with standardized interfaces
* **FR-003**: System SHALL preprocess images with noise reduction, contrast enhancement, and geometric correction
* **FR-004**: System SHALL validate image quality and flag low-quality captures for operator review

#### 1.2 Defect Detection Engine

* **FR-005**: System SHALL implement CNN/Vision Transformer models for multi-class defect detection
* **FR-006**: System SHALL detect scratches, dents, color variations, and dimensional issues with 99.5%+ accuracy
* **FR-007**: System SHALL provide confidence scores (0-100%) for each detected defect
* **FR-008**: System SHALL classify defect severity levels (minor, major, critical) based on predefined criteria
* **FR-009**: System SHALL process images in real-time with <100ms latency per inspection

### 2. Manufacturing Integration Module

#### 2.1 Assembly Line Integration

* **FR-010**: System SHALL integrate with existing PLCs and manufacturing control systems
* **FR-011**: System SHALL trigger inspections based on production line signals and timing
* **FR-012**: System SHALL send pass/fail decisions to downstream manufacturing equipment
* **FR-013**: System SHALL support multiple production line configurations and speeds

#### 2.2 Data Collection and Metadata

* **FR-014**: System SHALL capture production metadata (timestamps, batch numbers, operator IDs, line speed)
* **FR-015**: System SHALL associate defect detections with specific products and production context
* **FR-016**: System SHALL maintain audit trail for all inspection decisions and operator actions

### 3. Operator Interface Module

#### 3.1 Real-time Quality Dashboard

* **FR-017**: System SHALL display live inspection status with pass/fail rates and defect counts
* **FR-018**: System SHALL show defect visualizations with annotated images and location highlighting
* **FR-019**: System SHALL provide drill-down capabilities from summary metrics to individual defect details
* **FR-020**: System SHALL update dashboard in real-time with <1 second refresh rate

#### 3.2 Defect Review and Validation

* **FR-021**: System SHALL allow operators to review flagged defects with original and annotated images
* **FR-022**: System SHALL enable operators to confirm, reject, or reclassify detected defects
* **FR-023**: System SHALL provide defect measurement tools for dimensional analysis
* **FR-024**: System SHALL support batch review of multiple defects for efficiency

### 4. Quality Analytics Module

#### 4.1 Statistical Quality Control

* **FR-025**: System SHALL generate control charts (X-bar, R-charts, p-charts) for quality trends
* **FR-026**: System SHALL calculate process capability indices (Cp, Cpk) for quality assessment
* **FR-027**: System SHALL detect quality trend anomalies and trigger alerts
* **FR-028**: System SHALL provide root cause analysis suggestions based on defect patterns

#### 4.2 Reporting and Documentation

* **FR-029**: System SHALL generate automated quality reports (hourly, daily, weekly)
* **FR-030**: System SHALL export quality data in standard formats (CSV, PDF, XML)
* **FR-031**: System SHALL maintain quality history for regulatory compliance and audits

### 5. Continuous Learning Module

#### 5.1 Model Training and Improvement

* **FR-032**: System SHALL collect operator feedback for model retraining
* **FR-033**: System SHALL support active learning with uncertainty sampling
* **FR-034**: System SHALL enable model updates without production line downtime
* **FR-035**: System SHALL maintain model versioning and rollback capabilities

#### 5.2 Performance Monitoring

* **FR-036**: System SHALL monitor model performance metrics in real-time
* **FR-037**: System SHALL detect model drift and trigger retraining alerts
* **FR-038**: System SHALL provide A/B testing capabilities for model improvements # Non-Functional Requirements Document (NFRD) ## Manufacturing Quality Control AI Vision System

*Building upon PRD and FRD for system quality attributes and constraints*

## ETVX Framework

### ENTRY CRITERIA

* ✅ PRD completed with quantified success metrics (99.5% accuracy, <100ms processing)
* ✅ FRD completed with all functional requirements defined (FR-001 to FR-038)
* ✅ Manufacturing environment constraints and production line specifications documented
* ✅ Compliance and regulatory requirements identified (ISO 9001, Six Sigma standards)
* ✅ Technology constraints and hardware limitations documented

### TASK

Define system quality attributes, performance benchmarks, reliability requirements, security constraints, and operational parameters that ensure the computer vision system can deliver functional requirements with acceptable quality in harsh manufacturing environments.

### VERIFICATION & VALIDATION

**Verification Checklist:** - [ ] All NFRs are quantifiable and measurable with specific metrics - [ ] Performance targets align with PRD success metrics (99.5% accuracy, <100ms latency) - [ ] Reliability requirements meet manufacturing uptime standards (99.9%+) - [ ] Environmental requirements address manufacturing conditions (dust, vibration, temperature) - [ ] Each NFR is traceable to functional requirements and business objectives - [ ] Compliance requirements are comprehensive and auditable

**Validation Criteria:** - [ ] Performance targets are achievable with proposed computer vision architecture - [ ] Reliability requirements satisfy manufacturing operational needs - [ ] Environmental specifications validated with manufacturing facility conditions - [ ] Usability requirements validated with operator workflows and training - [ ] Infrastructure team confirms operational feasibility in production environment

### EXIT CRITERIA

* ✅ All quality attributes quantified with specific metrics and thresholds
* ✅ Performance benchmarks established for each computer vision component
* ✅ Reliability and environmental requirements fully documented
* ✅ Manufacturing integration and compliance targets defined
* ✅ Foundation established for system architecture design

### Reference to Previous Documents

This NFRD defines quality attributes and constraints based on **ALL** previous requirements: - **PRD Business Objectives** → Performance targets (99.5% accuracy, 30-40% cost reduction) - **PRD Success Metrics** → Quantified NFRs (<100ms processing, 99.9% uptime, <1% false positives) - **PRD Target Users** → Usability and training requirements for operators - **FRD Computer Vision Processing (FR-001 to FR-009)** → Performance requirements for real-time image processing - **FRD Manufacturing Integration (FR-010 to FR-016)** → Reliability requirements for production line integration - **FRD Operator Interface (FR-017 to FR-024)** → Usability requirements for dashboard and review workflows - **FRD Quality Analytics (FR-025 to FR-031)** → Scalability requirements for data processing and reporting - **FRD Continuous Learning (FR-032 to FR-038)** → Performance requirements for model training and updates

### 1. Performance Requirements

#### 1.1 Computer Vision Processing Performance

* **NFR-001**: Image processing latency SHALL be ≤100ms for 95% of inspections
* **NFR-002**: Defect detection accuracy SHALL be ≥99.5% across all defect categories
* **NFR-003**: False positive rate SHALL be ≤1% to minimize production disruption
* **NFR-004**: System SHALL process 1000+ images per hour per production line
* **NFR-005**: Model inference time SHALL be ≤50ms per image on target hardware

#### 1.2 System Response Time

* **NFR-006**: Dashboard refresh rate SHALL be ≤1 second for real-time monitoring
* **NFR-007**: Operator interface response time SHALL be ≤2 seconds for 95% of interactions
* **NFR-008**: Quality report generation SHALL complete within 30 seconds for daily reports
* **NFR-009**: Database query response time SHALL be ≤500ms for historical data retrieval

### 2. Reliability & Availability Requirements

#### 2.1 Manufacturing Uptime Requirements

* **NFR-010**: System availability SHALL be 99.9% during production hours (max 8.77 hours downtime/year)
* **NFR-011**: Mean Time Between Failures (MTBF) SHALL be ≥2000 hours
* **NFR-012**: Mean Time To Recovery (MTTR) SHALL be ≤15 minutes for system failures
* **NFR-013**: Planned maintenance windows SHALL not exceed 2 hours monthly

#### 2.2 Data Integrity and Backup

* **NFR-014**: Quality data backup SHALL occur every 4 hours with 90-day retention
* **NFR-015**: Recovery Point Objective (RPO) SHALL be ≤30 minutes
* **NFR-016**: System SHALL maintain 99.99% data accuracy for quality records
* **NFR-017**: Audit trail SHALL be immutable and tamper-evident for compliance

### 3. Environmental & Hardware Requirements

#### 3.1 Manufacturing Environment Tolerance

* **NFR-018**: System SHALL operate in temperature range -10°C to +60°C
* **NFR-019**: System SHALL withstand vibration levels up to 2G acceleration
* **NFR-020**: System SHALL function with dust levels up to IP65 protection rating
* **NFR-021**: System SHALL maintain performance with 85% humidity levels
* **NFR-022**: System SHALL operate with electromagnetic interference typical in manufacturing

#### 3.2 Hardware Performance Requirements

* **NFR-023**: Camera systems SHALL capture minimum 2048x2048 pixel resolution at 30 FPS
* **NFR-024**: Processing hardware SHALL support GPU acceleration for computer vision workloads
* **NFR-025**: Storage system SHALL handle 10TB+ of image data with automated archiving
* **NFR-026**: Network infrastructure SHALL support 1Gbps+ bandwidth for image transfer

### 4. Scalability Requirements

#### 4.1 Production Line Scaling

* **NFR-027**: System SHALL scale to support 50+ production lines per facility
* **NFR-028**: System SHALL handle 100,000+ product inspections per day
* **NFR-029**: System SHALL support horizontal scaling with additional processing nodes
* **NFR-030**: Database SHALL scale to store 1M+ inspection records per month

#### 4.2 Model and Data Scaling

* **NFR-031**: System SHALL support 20+ defect categories with expandable classification
* **NFR-032**: Model training SHALL handle datasets with 100,000+ labeled images
* **NFR-033**: System SHALL maintain performance with 10+ concurrent model versions

### 5. Security Requirements

#### 5.1 Manufacturing Network Security

* **NFR-034**: System SHALL implement network segmentation for OT/IT separation
* **NFR-035**: System SHALL use encrypted communication (TLS 1.3) for all data transfer
* **NFR-036**: System SHALL implement role-based access control with manufacturing-specific roles
* **NFR-037**: System SHALL maintain security audit logs for 7+ years

#### 5.2 Data Protection

* **NFR-038**: Sensitive production data SHALL be encrypted at rest using AES-256
* **NFR-039**: System SHALL implement secure key management for encryption
* **NFR-040**: System SHALL comply with industrial cybersecurity standards (IEC 62443)

### 6. Usability Requirements

#### 6.1 Operator Interface Usability

* **NFR-041**: Operator training time SHALL be ≤2 hours for basic system operation
* **NFR-042**: System SHALL support touch-screen interfaces suitable for manufacturing gloves
* **NFR-043**: Interface SHALL be readable in bright manufacturing lighting conditions
* **NFR-044**: System SHALL provide multi-language support for international facilities

#### 6.2 Maintenance and Support

* **NFR-045**: System SHALL provide self-diagnostic capabilities with error code reporting
* **NFR-046**: Remote monitoring and support SHALL be available 24/7
* **NFR-047**: System updates SHALL be deployable without production line shutdown

### 7. Compliance Requirements

#### 7.1 Quality Standards Compliance

* **NFR-048**: System SHALL comply with ISO 9001 quality management standards
* **NFR-049**: System SHALL support Six Sigma quality methodologies and reporting
* **NFR-050**: System SHALL maintain traceability for regulatory audits (FDA, automotive standards)
* **NFR-051**: System SHALL generate compliance reports in required formats # Architecture Diagram (AD) ## Manufacturing Quality Control AI Vision System

*Building upon PRD, FRD, and NFRD for comprehensive system architecture*

## ETVX Framework

### ENTRY CRITERIA

* ✅ PRD business objectives and constraints defined
* ✅ FRD functional requirements completely specified (FR-001 to FR-038)
* ✅ NFRD performance, reliability, and environmental targets established
* ✅ Manufacturing environment specifications and hardware constraints documented
* ✅ Integration requirements with existing PLCs and manufacturing systems identified

### TASK

Design comprehensive computer vision system architecture that satisfies all functional and non-functional requirements, including real-time image processing pipelines, manufacturing integration patterns, edge computing deployment, and quality analytics infrastructure.

### VERIFICATION & VALIDATION

**Verification Checklist:** - [ ] Architecture addresses all functional requirements (FR-001 to FR-038) - [ ] Design meets all non-functional requirements (NFR-001 to NFR-051) - [ ] Real-time processing pipeline supports <100ms latency requirements - [ ] Manufacturing integration patterns are clearly defined - [ ] Edge computing architecture handles harsh manufacturing environments - [ ] Quality analytics infrastructure supports statistical process control

**Validation Criteria:** - [ ] Architecture supports PRD business objectives (99.5% accuracy, 30-40% cost reduction) - [ ] Performance projections meet NFRD targets (<100ms processing, 99.9% uptime) - [ ] Environmental design validated for manufacturing conditions (temperature, vibration, dust) - [ ] Integration patterns confirmed with existing manufacturing systems - [ ] Architecture review completed with computer vision and manufacturing teams

### EXIT CRITERIA

* ✅ Complete system architecture with all components defined
* ✅ Technology stack selections documented and approved for manufacturing environment
* ✅ Real-time processing pipeline and manufacturing integration patterns completed
* ✅ Edge computing and quality analytics architecture specified
* ✅ Foundation established for high-level design development

### Reference to Previous Documents

This Architecture Diagram implements the complete system design based on **ALL** previous requirements: - **PRD Product Features** → System components (computer vision pipeline, defect detection, operator interface) - **PRD Target Users** → Edge computing architecture for operators, quality managers, engineers - **PRD Manufacturing Constraints** → Edge deployment architecture for production line integration - **FRD Computer Vision Processing (FR-001 to FR-009)** → Real-time image processing pipeline architecture - **FRD Manufacturing Integration (FR-010 to FR-016)** → PLC integration and production line connectivity - **FRD Operator Interface (FR-017 to FR-024)** → Dashboard and visualization architecture - **FRD Quality Analytics (FR-025 to FR-031)** → Statistical process control and reporting infrastructure - **FRD Continuous Learning (FR-032 to FR-038)** → Model training and deployment pipeline - **NFRD Performance (NFR-001 to NFR-009)** → High-performance edge computing with GPU acceleration - **NFRD Environmental (NFR-018 to NFR-026)** → Ruggedized hardware architecture for manufacturing - **NFRD Security (NFR-034 to NFR-040)** → OT/IT network segmentation and industrial cybersecurity

### 1. System Architecture Overview

┌─────────────────────────────────────────────────────────────────────────────────┐  
│ MANUFACTURING FLOOR EDGE LAYER │  
├─────────────────┬─────────────────┬─────────────────┬─────────────────────────────┤  
│ Production │ Quality │ Operator │ Mobile Quality │  
│ Line HMI │ Dashboard │ Workstation │ Inspector App │  
│ (Touch Screen) │ (Real-time) │ (Defect Review)│ (iOS/Android) │  
└─────────────────┴─────────────────┴─────────────────┴─────────────────────────────┘  
 │  
 ┌─────────┴─────────┐  
 │ Edge Gateway │  
 │ (Industrial PC) │  
 └─────────┬─────────┘  
 │  
┌─────────────────────────────┼─────────────────────────────────────────────────────┐  
│ EDGE PROCESSING LAYER │  
├─────────────────┬───────────┼───────────┬─────────────────┬───────────────────────┤  
│ Computer │ Manufacturing │ Quality │ Continuous │ │  
│ Vision Engine │ Integration │Analytics│ Learning │ │  
│ │ Service │ Service │ Service │ │  
│ ┌─────────────┐ │┌─────────────┐│┌───────┐│ ┌─────────────┐ │ │  
│ │Image │ ││PLC │││SPC ││ │Model │ │ │  
│ │Preprocessor │ ││Connector │││Engine ││ │Training │ │ │  
│ │CNN/ViT │ ││SCADA │││Report ││ │Deployment │ │ │  
│ │Defect │ ││Integration │││Gen ││ │Versioning │ │ │  
│ │Classifier │ │└─────────────┘│└───────┘│ └─────────────┘ │ │  
│ └─────────────┘ │ │ │ │ │  
└─────────────────┴───────────────┼─────────┴─────────────────┴───────────────────────┘  
 │  
┌─────────────────────────────┼─────────────────────────────────────────────────────┐  
│ HARDWARE ABSTRACTION LAYER │  
├─────────────────────────────┼─────────────────────────────────────────────────────┤  
│ Camera Systems │ Processing Hardware │  
│ │ │  
│ ┌─────────────────────────┐ │ ┌─────────────────────────────────────────────────┐ │  
│ │ High-Res Cameras │ │ │ Industrial Edge Computer │ │  
│ │ - RGB (2048x2048) │ │ │ - Intel/NVIDIA GPU (RTX/Quadro) │ │  
│ │ - Infrared/Thermal │ │ │ - 32GB+ RAM, NVMe SSD │ │  
│ │ - Depth Sensors │ │ │ - IP65 Rated, -10°C to +60°C │ │  
│ │ - Lighting Systems │ │ │ - Vibration Resistant (2G) │ │  
│ └─────────────────────────┘ │ └─────────────────────────────────────────────────┘ │  
└─────────────────────────────┼─────────────────────────────────────────────────────┘  
 │  
┌─────────────────────────────┼─────────────────────────────────────────────────────┐  
│ MANUFACTURING INTEGRATION LAYER │  
├─────────────────┬───────────┼───────────┬─────────────────────────────────────────┤  
│ Production │ Quality │ SCADA │ External Systems │  
│ Line PLCs │ Systems │ Systems │ │  
│ │ │ │ │  
│ ┌─────────────┐ │┌─────────┐│┌─────────┐│ ┌─────────────────────────────────────┐ │  
│ │Conveyor │ ││MES │││Historian││ │ERP Systems (SAP, Oracle) │ │  
│ │Control │ ││Systems │││Data ││ │Quality Management (QMS) │ │  
│ │Reject │ ││Batch │││Logger ││ │Maintenance Systems (CMMS) │ │  
│ │Mechanisms │ ││Tracking │││Alarms ││ │Business Intelligence (BI) │ │  
│ └─────────────┘ │└─────────┘│└─────────┘│ └─────────────────────────────────────┘ │  
└─────────────────┴───────────┴───────────┴─────────────────────────────────────────┘

### 2. Edge Computing Architecture Details

#### 2.1 Real-time Processing Pipeline

Camera Capture → Image Buffer → Preprocessing → CNN/ViT Inference → Defect Classification →   
Decision Logic → PLC Signal → Production Action → Quality Database → Analytics Dashboard  
 ↓ ↓ ↓ ↓ ↓  
 30 FPS Ring Buffer GPU Accel. <50ms Inference <100ms Total

#### 2.2 Manufacturing Integration Pattern

* **Edge-First Architecture**: All critical processing at production line edge
* **Deterministic Communication**: Real-time protocols (EtherCAT, PROFINET) for PLC integration
* **Fail-Safe Design**: Hardware watchdogs and redundant systems for 99.9% uptime

### 3. Technology Stack

#### 3.1 Computer Vision Technologies

* **Deep Learning Framework**: PyTorch/TensorRT for optimized inference
* **Model Architecture**: EfficientNet, Vision Transformer (ViT), YOLO for detection
* **Image Processing**: OpenCV, PIL for preprocessing and augmentation
* **GPU Acceleration**: CUDA, TensorRT for <50ms inference times

#### 3.2 Edge Computing Technologies

* **Operating System**: Ubuntu 20.04 LTS with real-time kernel patches
* **Container Runtime**: Docker with GPU support for model deployment
* **Message Queue**: Redis for high-speed inter-process communication
* **Database**: SQLite for local storage, PostgreSQL for analytics

#### 3.3 Manufacturing Integration

* **PLC Communication**: OPC-UA, Modbus TCP for industrial protocols
* **SCADA Integration**: Kepware, Ignition for manufacturing system connectivity
* **Time Synchronization**: IEEE 1588 PTP for microsecond timing accuracy
* **Network**: Industrial Ethernet with TSN (Time-Sensitive Networking)

### 4. Deployment Architecture

#### 4.1 Edge Deployment Strategy

* **Single Production Line**: Dedicated edge computer per line
* **Multi-Line Facility**: Centralized edge cluster with distributed cameras
* **Redundancy**: Hot-standby systems for critical production lines
* **Remote Management**: Centralized monitoring and OTA updates

#### 4.2 Network Architecture

Manufacturing Floor Network (OT)  
├── Production Line Segment (VLAN 10)  
│ ├── Edge Computers (192.168.10.x)  
│ ├── Cameras (192.168.10.100-199)  
│ └── PLCs (192.168.10.200-299)  
├── Quality Management Segment (VLAN 20)  
│ ├── Quality Dashboards (192.168.20.x)  
│ └── Analytics Servers (192.168.20.100-199)  
└── IT Integration Segment (VLAN 30)  
 ├── ERP Connectors (192.168.30.x)  
 └── Business Systems (192.168.30.100-199)

### 5. Security Architecture

* **Network Segmentation**: OT/IT separation with industrial firewalls
* **Device Authentication**: Certificate-based authentication for all devices
* **Data Encryption**: TLS 1.3 for data in transit, AES-256 for data at rest
* **Access Control**: Role-based access with manufacturing-specific permissions
* **Compliance**: IEC 62443 industrial cybersecurity standards

### 6. Quality Analytics Architecture

* **Real-time Analytics**: Stream processing for immediate quality metrics
* **Historical Analytics**: Time-series database for trend analysis
* **Statistical Process Control**: Automated control chart generation
* **Predictive Analytics**: ML models for quality trend prediction
* **Reporting Engine**: Automated report generation for compliance # High Level Design (HLD) ## Manufacturing Quality Control AI Vision System

*Building upon PRD, FRD, NFRD, and Architecture Diagram for detailed system design and interactions*

## ETVX Framework

### ENTRY CRITERIA

* ✅ Architecture Diagram completed and approved
* ✅ All system components and their relationships defined
* ✅ Technology stack selections finalized for manufacturing environment
* ✅ Edge computing and manufacturing integration patterns established
* ✅ Performance and environmental architecture validated

### TASK

Elaborate the system architecture into detailed design specifications including computer vision pipeline components, manufacturing integration interfaces, quality analytics algorithms, edge deployment patterns, and real-time interaction flows between all system elements.

### VERIFICATION & VALIDATION

**Verification Checklist:** - [ ] All architectural components have detailed design specifications - [ ] Computer vision pipeline supports <100ms processing requirements - [ ] Manufacturing integration interfaces are complete with PLC protocols - [ ] Quality analytics algorithms meet statistical process control needs - [ ] Edge deployment design handles environmental constraints - [ ] Real-time communication patterns are deterministic and reliable

**Validation Criteria:** - [ ] Design supports all architectural quality attributes - [ ] Computer vision pipeline validated through prototyping - [ ] Manufacturing integration confirmed with existing PLC systems - [ ] Quality analytics validated with Six Sigma methodologies - [ ] Edge deployment tested in manufacturing environment conditions - [ ] Design review completed with computer vision and manufacturing teams

### EXIT CRITERIA

* ✅ Detailed component specifications for all system elements
* ✅ Complete computer vision pipeline with processing stages
* ✅ Manufacturing integration design with protocol specifications
* ✅ Quality analytics algorithms with statistical methods
* ✅ Foundation established for low-level implementation design

### Reference to Previous Documents

This HLD provides detailed system design implementing **ALL** previous requirements: - **PRD Business Objectives** → System design optimized for 99.5% accuracy, 30-40% cost reduction - **PRD Key Features** → Detailed component design (computer vision, defect detection, operator interface) - **FRD Computer Vision Processing (FR-001 to FR-009)** → Real-time image processing pipeline design - **FRD Manufacturing Integration (FR-010 to FR-016)** → PLC integration and production line connectivity - **FRD Operator Interface (FR-017 to FR-024)** → Dashboard and defect review interface design - **FRD Quality Analytics (FR-025 to FR-031)** → Statistical process control and reporting algorithms - **FRD Continuous Learning (FR-032 to FR-038)** → Model training and deployment pipeline design - **NFRD Performance Requirements** → Edge computing design for <100ms processing, 99.9% uptime - **NFRD Environmental Requirements** → Ruggedized hardware design for manufacturing conditions - **NFRD Security Requirements** → OT/IT segmentation and industrial cybersecurity implementation - **Architecture Diagram Components** → Detailed interaction design between edge computing, manufacturing systems

### 1. Computer Vision Pipeline Design

#### 1.1 Image Acquisition and Preprocessing

Camera Systems → Image Buffer → Quality Validation → Preprocessing → Feature Extraction  
 ↓ ↓ ↓ ↓ ↓  
Multi-spectral Ring Buffer Blur/Focus Noise Reduction ROI Detection  
RGB/IR/Depth (30 frames) Detection Contrast Enhance Geometric Correct  
2048x2048@30fps GPU Memory Auto-reject Normalization Perspective Fix

#### 1.2 Deep Learning Inference Pipeline

class VisionInferencePipeline:  
 def \_\_init\_\_(self):  
 self.preprocessor = ImagePreprocessor()  
 self.defect\_detector = EfficientNetB7\_TensorRT()  
 self.classifier = ViT\_Optimized()  
 self.postprocessor = DefectPostProcessor()  
   
 def process\_image(self, image):  
 # Preprocessing (5-10ms)  
 processed\_image = self.preprocessor.enhance\_and\_normalize(image)  
   
 # Defect Detection (20-30ms)  
 detections = self.defect\_detector.detect(processed\_image)  
   
 # Classification (15-25ms)  
 classifications = self.classifier.classify(detections)  
   
 # Post-processing (5-10ms)  
 results = self.postprocessor.format\_results(classifications)  
   
 return results # Total: <100ms

#### 1.3 Model Architecture Design

* **Primary Model**: EfficientNet-B7 with TensorRT optimization for defect detection
* **Secondary Model**: Vision Transformer (ViT) for fine-grained classification
* **Ensemble Strategy**: Weighted voting with confidence thresholding
* **Model Quantization**: INT8 quantization for 3x speed improvement

### 2. Manufacturing Integration Design

#### 2.1 PLC Communication Architecture

Edge Computer → OPC-UA Client → Industrial Network → PLC Server → Production Control  
 ↓ ↓ ↓ ↓ ↓  
Quality Decision Structured Data EtherCAT/PROFINET Logic Control Conveyor/Reject  
Pass/Fail/Review JSON/XML Format <1ms Latency Ladder Logic Pneumatic Systems  
Confidence Score Timestamped Deterministic Safety Interlocks Visual Indicators

#### 2.2 Production Line Integration Flow

class ProductionLineIntegrator:  
 def \_\_init\_\_(self, plc\_client):  
 self.plc = plc\_client  
 self.quality\_buffer = QualityDecisionBuffer()  
   
 def process\_inspection\_trigger(self, product\_id):  
 # Receive trigger from PLC  
 inspection\_request = self.plc.read\_inspection\_trigger()  
   
 # Capture and process image  
 image = self.camera\_system.capture()  
 quality\_result = self.vision\_pipeline.process(image)  
   
 # Send decision to PLC  
 decision = self.make\_quality\_decision(quality\_result)  
 self.plc.write\_quality\_decision(product\_id, decision)  
   
 # Log for analytics  
 self.quality\_buffer.add\_result(product\_id, quality\_result)

### 3. Quality Analytics Engine Design

#### 3.1 Statistical Process Control Implementation

class SPCEngine:  
 def \_\_init\_\_(self):  
 self.control\_charts = {  
 'x\_bar': XBarChart(),  
 'r\_chart': RChart(),  
 'p\_chart': PChart(),  
 'c\_chart': CChart()  
 }  
   
 def update\_control\_charts(self, quality\_data):  
 # X-bar and R charts for continuous data  
 if quality\_data.type == 'continuous':  
 self.control\_charts['x\_bar'].add\_sample(quality\_data.value)  
 self.control\_charts['r\_chart'].add\_range(quality\_data.range)  
   
 # P-chart for defect rates  
 elif quality\_data.type == 'defect\_rate':  
 self.control\_charts['p\_chart'].add\_proportion(quality\_data.defect\_rate)  
   
 # Check for out-of-control conditions  
 alerts = self.check\_control\_limits()  
 return alerts

#### 3.2 Real-time Analytics Dashboard Data Flow

Quality Results → Stream Processor → Aggregation Engine → Dashboard API → UI Components  
 ↓ ↓ ↓ ↓ ↓  
Individual Redis Streams Time Windows REST/WebSocket Real-time Charts  
Inspections Event Processing Statistical Calc JSON Response Control Charts  
Metadata Pattern Detection Trend Analysis <1s Latency Alert Indicators

### 4. Edge Computing Architecture

#### 4.1 Hardware Resource Management

class EdgeResourceManager:  
 def \_\_init\_\_(self):  
 self.gpu\_scheduler = GPUScheduler()  
 self.memory\_manager = MemoryManager()  
 self.thermal\_monitor = ThermalMonitor()  
   
 def optimize\_inference\_performance(self):  
 # GPU utilization optimization  
 self.gpu\_scheduler.balance\_workload()  
   
 # Memory management for image buffers  
 self.memory\_manager.cleanup\_old\_buffers()  
   
 # Thermal throttling prevention  
 if self.thermal\_monitor.temperature > 75:  
 self.reduce\_processing\_frequency()

#### 4.2 Fault Tolerance and Recovery Design

* **Watchdog Systems**: Hardware and software watchdogs for system monitoring
* **Graceful Degradation**: Reduced accuracy mode during hardware issues
* **Hot Standby**: Secondary edge computer for critical production lines
* **Data Recovery**: Local buffering with automatic retry mechanisms

### 5. Continuous Learning Pipeline

#### 5.1 Active Learning Implementation

class ActiveLearningSystem:  
 def \_\_init\_\_(self):  
 self.uncertainty\_sampler = UncertaintySampler()  
 self.human\_feedback\_collector = FeedbackCollector()  
 self.model\_retrainer = ModelRetrainer()  
   
 def identify\_uncertain\_samples(self, predictions):  
 # Identify low-confidence predictions  
 uncertain\_samples = self.uncertainty\_sampler.select(  
 predictions, threshold=0.8  
 )  
   
 # Queue for human review  
 self.human\_feedback\_collector.queue\_for\_review(uncertain\_samples)  
   
 return uncertain\_samples

#### 5.2 Model Deployment Pipeline

* **Blue-Green Deployment**: Zero-downtime model updates
* **A/B Testing**: Performance comparison between model versions
* **Rollback Capability**: Automatic rollback on performance degradation
* **Version Control**: MLflow for model versioning and artifact management

### 6. Security Implementation Design

#### 6.1 Network Security Architecture

class SecurityManager:  
 def \_\_init\_\_(self):  
 self.certificate\_manager = CertificateManager()  
 self.access\_controller = AccessController()  
 self.audit\_logger = AuditLogger()  
   
 def authenticate\_device(self, device\_id, certificate):  
 # Certificate-based authentication  
 if self.certificate\_manager.validate(certificate):  
 session = self.access\_controller.create\_session(device\_id)  
 self.audit\_logger.log\_access(device\_id, 'SUCCESS')  
 return session  
 else:  
 self.audit\_logger.log\_access(device\_id, 'FAILED')  
 return None

#### 6.2 Data Protection Implementation

* **Encryption at Rest**: AES-256 encryption for stored images and quality data
* **Encryption in Transit**: TLS 1.3 for all network communications
* **Key Management**: Hardware Security Module (HSM) for key storage
* **Access Logging**: Comprehensive audit trail for compliance

### 7. Performance Optimization Design

#### 7.1 Real-time Processing Optimization

* **GPU Memory Management**: Efficient CUDA memory allocation and deallocation
* **Batch Processing**: Dynamic batching for improved GPU utilization
* **Pipeline Parallelism**: Overlapped image capture, processing, and result handling
* **Cache Optimization**: Intelligent caching of model weights and intermediate results

#### 7.2 Database Design for Analytics

-- Optimized schema for time-series quality data  
CREATE TABLE quality\_inspections (  
 id BIGSERIAL PRIMARY KEY,  
 product\_id VARCHAR(50) NOT NULL,  
 line\_id INTEGER NOT NULL,  
 timestamp TIMESTAMP WITH TIME ZONE NOT NULL,  
 defect\_type VARCHAR(50),  
 confidence\_score DECIMAL(5,4),  
 severity\_level INTEGER,  
 image\_path VARCHAR(255),  
 operator\_feedback JSONB  
);  
  
-- Partitioning by date for performance  
CREATE INDEX idx\_quality\_timestamp ON quality\_inspections   
USING BRIN (timestamp);  
  
-- Materialized views for real-time analytics  
CREATE MATERIALIZED VIEW hourly\_quality\_metrics AS  
SELECT   
 DATE\_TRUNC('hour', timestamp) as hour,  
 line\_id,  
 COUNT(\*) as total\_inspections,  
 COUNT(CASE WHEN defect\_type IS NOT NULL THEN 1 END) as defects\_found,  
 AVG(confidence\_score) as avg\_confidence  
FROM quality\_inspections  
GROUP BY DATE\_TRUNC('hour', timestamp), line\_id;

# Low Level Design (LLD)

## Manufacturing Quality Control AI Vision System

*Building upon PRD, FRD, NFRD, Architecture Diagram, and HLD for detailed implementation specifications and code-level design*

## ETVX Framework

### ENTRY CRITERIA

* ✅ HLD completed with detailed component specifications
* ✅ Computer vision pipeline and manufacturing integration interfaces finalized
* ✅ Quality analytics algorithms and edge deployment patterns defined
* ✅ Development environment and coding standards established for manufacturing systems
* ✅ Code review and testing processes defined for safety-critical applications

### TASK

Transform high-level design into implementation-ready code specifications including computer vision class definitions, manufacturing protocol implementations, real-time processing algorithms, edge deployment configurations, and detailed implementation logic for all system components.

### VERIFICATION & VALIDATION

**Verification Checklist:** - [ ] All HLD components have corresponding code implementations - [ ] Computer vision classes follow real-time processing requirements (<100ms) - [ ] Manufacturing integration implements industrial protocols correctly - [ ] Quality analytics algorithms match statistical process control standards - [ ] Edge deployment code handles environmental constraints and fault tolerance - [ ] Code follows manufacturing software safety standards (IEC 61508)

**Validation Criteria:** - [ ] Implementation logic satisfies all functional requirements (FR-001 to FR-038) - [ ] Code structure supports non-functional requirements (NFR-001 to NFR-051) - [ ] Computer vision algorithms meet 99.5% accuracy benchmarks - [ ] Manufacturing integration tested with actual PLC systems - [ ] Security implementations follow industrial cybersecurity standards - [ ] Code review completed by computer vision and manufacturing experts

### EXIT CRITERIA

* ✅ Complete code specifications for all system components
* ✅ Implementation-ready computer vision and manufacturing integration classes
* ✅ Detailed real-time processing algorithms with performance analysis
* ✅ Edge deployment and fault tolerance specifications completed
* ✅ Foundation established for pseudocode and actual implementation

### Reference to Previous Documents

This LLD provides implementation-ready code specifications based on **ALL** previous requirements: - **PRD Success Metrics** → Code implementations targeting 99.5% accuracy, <100ms processing times - **PRD Target Users** → User-specific interfaces for operators, quality managers, engineers - **FRD Functional Requirements (FR-001 to FR-038)** → Direct code implementation of each requirement - **NFRD Performance Requirements** → Optimized algorithms, GPU acceleration, real-time processing - **NFRD Environmental Requirements** → Ruggedized code for manufacturing conditions - **NFRD Security Requirements** → Industrial cybersecurity implementations, OT/IT segmentation - **Architecture Diagram Technology Stack** → Specific framework implementations (PyTorch, TensorRT, OPC-UA) - **HLD System Components** → Detailed class structures, method signatures, real-time data flows - **HLD Computer Vision Pipeline** → CNN/ViT implementations with TensorRT optimization - **HLD Manufacturing Integration** → PLC communication protocols and production line interfaces - **HLD Quality Analytics** → Statistical process control algorithms and dashboard implementations

### 1. Computer Vision Implementation

#### 1.1 Image Processing and Preprocessing Classes

import cv2  
import numpy as np  
import torch  
import tensorrt as trt  
from typing import Tuple, List, Optional  
  
class IndustrialImagePreprocessor:  
 """High-performance image preprocessing for manufacturing environment"""  
   
 def \_\_init\_\_(self, target\_size: Tuple[int, int] = (2048, 2048)):  
 self.target\_size = target\_size  
 self.noise\_filter = cv2.createFastNlMeansDenoising()  
 self.clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))  
   
 def preprocess\_image(self, image: np.ndarray) -> torch.Tensor:  
 """  
 Preprocess image for defect detection  
 Target: <10ms processing time  
 """  
 # Noise reduction (2-3ms)  
 denoised = cv2.fastNlMeansDenoising(image)  
   
 # Contrast enhancement (1-2ms)  
 enhanced = self.clahe.apply(denoised)  
   
 # Geometric correction (2-3ms)  
 corrected = self.\_correct\_perspective(enhanced)  
   
 # Normalization and tensor conversion (1-2ms)  
 normalized = corrected.astype(np.float32) / 255.0  
 tensor = torch.from\_numpy(normalized).unsqueeze(0)  
   
 return tensor  
   
 def \_correct\_perspective(self, image: np.ndarray) -> np.ndarray:  
 """Correct perspective distortion from camera angle"""  
 # Implementation for perspective correction  
 return image  
  
class DefectDetectionModel:  
 """TensorRT optimized defect detection model"""  
   
 def \_\_init\_\_(self, model\_path: str, confidence\_threshold: float = 0.8):  
 self.confidence\_threshold = confidence\_threshold  
 self.engine = self.\_load\_tensorrt\_engine(model\_path)  
 self.context = self.engine.create\_execution\_context()  
   
 def \_load\_tensorrt\_engine(self, model\_path: str):  
 """Load optimized TensorRT engine"""  
 with open(model\_path, 'rb') as f:  
 engine\_data = f.read()  
   
 runtime = trt.Runtime(trt.Logger(trt.Logger.WARNING))  
 engine = runtime.deserialize\_cuda\_engine(engine\_data)  
 return engine  
   
 def detect\_defects(self, image\_tensor: torch.Tensor) -> List[dict]:  
 """  
 Detect defects in image  
 Target: <50ms inference time  
 """  
 # GPU memory allocation  
 input\_binding = self.engine.get\_binding\_index("input")  
 output\_binding = self.engine.get\_binding\_index("output")  
   
 # Inference execution  
 self.context.execute\_v2([  
 image\_tensor.data\_ptr(),  
 self.output\_buffer.data\_ptr()  
 ])  
   
 # Post-process results  
 detections = self.\_post\_process\_detections(self.output\_buffer)  
   
 return detections  
   
 def \_post\_process\_detections(self, raw\_output: torch.Tensor) -> List[dict]:  
 """Convert raw model output to structured detections"""  
 detections = []  
   
 for detection in raw\_output:  
 if detection.confidence > self.confidence\_threshold:  
 detections.append({  
 'bbox': detection.bbox.tolist(),  
 'class': detection.class\_id,  
 'confidence': float(detection.confidence),  
 'severity': self.\_calculate\_severity(detection)  
 })  
   
 return detections

#### 1.2 Real-time Processing Pipeline Implementation

import asyncio  
import time  
from concurrent.futures import ThreadPoolExecutor  
from dataclasses import dataclass  
from typing import AsyncGenerator  
  
@dataclass  
class InspectionResult:  
 product\_id: str  
 timestamp: float  
 defects: List[dict]  
 overall\_quality: str # 'PASS', 'FAIL', 'REVIEW'  
 confidence\_score: float  
 processing\_time\_ms: float  
  
class RealTimeVisionPipeline:  
 """Real-time computer vision pipeline for manufacturing"""  
   
 def \_\_init\_\_(self, camera\_system, defect\_model, plc\_interface):  
 self.camera = camera\_system  
 self.model = defect\_model  
 self.plc = plc\_interface  
 self.preprocessor = IndustrialImagePreprocessor()  
 self.executor = ThreadPoolExecutor(max\_workers=4)  
   
 # Performance monitoring  
 self.processing\_times = []  
 self.accuracy\_metrics = []  
   
 async def process\_inspection\_stream(self) -> AsyncGenerator[InspectionResult, None]:  
 """Main processing loop for continuous inspection"""  
   
 while True:  
 try:  
 # Wait for inspection trigger from PLC  
 trigger\_signal = await self.plc.wait\_for\_inspection\_trigger()  
   
 if trigger\_signal:  
 start\_time = time.time()  
   
 # Capture image (5-10ms)  
 image = await self.camera.capture\_image()  
   
 # Process in thread pool to avoid blocking  
 result = await asyncio.get\_event\_loop().run\_in\_executor(  
 self.executor, self.\_process\_single\_image, image, trigger\_signal.product\_id  
 )  
   
 # Send result to PLC (1-2ms)  
 await self.plc.send\_quality\_decision(result)  
   
 # Update performance metrics  
 processing\_time = (time.time() - start\_time) \* 1000  
 self.\_update\_performance\_metrics(processing\_time, result)  
   
 yield result  
   
 except Exception as e:  
 await self.\_handle\_processing\_error(e)  
   
 def \_process\_single\_image(self, image: np.ndarray, product\_id: str) -> InspectionResult:  
 """Process single image for defect detection"""  
 start\_time = time.time()  
   
 # Preprocessing (5-10ms)  
 processed\_image = self.preprocessor.preprocess\_image(image)  
   
 # Defect detection (20-40ms)  
 defects = self.model.detect\_defects(processed\_image)  
   
 # Quality decision logic (1-2ms)  
 overall\_quality, confidence = self.\_make\_quality\_decision(defects)  
   
 processing\_time = (time.time() - start\_time) \* 1000  
   
 return InspectionResult(  
 product\_id=product\_id,  
 timestamp=time.time(),  
 defects=defects,  
 overall\_quality=overall\_quality,  
 confidence\_score=confidence,  
 processing\_time\_ms=processing\_time  
 )  
   
 def \_make\_quality\_decision(self, defects: List[dict]) -> Tuple[str, float]:  
 """Make pass/fail decision based on detected defects"""  
 if not defects:  
 return "PASS", 1.0  
   
 # Check for critical defects  
 critical\_defects = [d for d in defects if d['severity'] >= 3]  
 if critical\_defects:  
 return "FAIL", min([d['confidence'] for d in critical\_defects])  
   
 # Check for major defects  
 major\_defects = [d for d in defects if d['severity'] == 2]  
 if len(major\_defects) > 2: # More than 2 major defects = fail  
 return "FAIL", min([d['confidence'] for d in major\_defects])  
   
 # Minor defects require review  
 minor\_defects = [d for d in defects if d['severity'] == 1]  
 if minor\_defects:  
 return "REVIEW", min([d['confidence'] for d in minor\_defects])  
   
 return "PASS", min([d['confidence'] for d in defects])

### 2. Manufacturing Integration Implementation

#### 2.1 PLC Communication Interface

import asyncio  
from opcua import Client, ua  
from dataclasses import dataclass  
from typing import Optional, Dict, Any  
  
@dataclass  
class PLCTriggerSignal:  
 product\_id: str  
 line\_speed: float  
 batch\_number: str  
 timestamp: float  
  
class OPCUAClient:  
 """OPC-UA client for PLC communication"""  
   
 def \_\_init\_\_(self, endpoint\_url: str, namespace\_index: int = 2):  
 self.endpoint\_url = endpoint\_url  
 self.namespace\_index = namespace\_index  
 self.client = None  
 self.connected = False  
   
 async def connect(self):  
 """Establish connection to PLC"""  
 try:  
 self.client = Client(self.endpoint\_url)  
 await self.client.connect()  
 self.connected = True  
   
 # Set up subscription for inspection triggers  
 await self.\_setup\_subscriptions()  
   
 except Exception as e:  
 raise ConnectionError(f"Failed to connect to PLC: {e}")  
   
 async def \_setup\_subscriptions(self):  
 """Set up OPC-UA subscriptions for real-time data"""  
 subscription = await self.client.create\_subscription(100, self)  
   
 # Subscribe to inspection trigger node  
 trigger\_node = self.client.get\_node(f"ns={self.namespace\_index};s=InspectionTrigger")  
 await subscription.subscribe\_data\_change(trigger\_node)  
   
 # Subscribe to production data  
 production\_node = self.client.get\_node(f"ns={self.namespace\_index};s=ProductionData")  
 await subscription.subscribe\_data\_change(production\_node)  
   
 async def wait\_for\_inspection\_trigger(self) -> Optional[PLCTriggerSignal]:  
 """Wait for inspection trigger from PLC"""  
 # Implementation for waiting for trigger signal  
 pass  
   
 async def send\_quality\_decision(self, result: InspectionResult):  
 """Send quality decision back to PLC"""  
 try:  
 # Write quality result to PLC  
 quality\_node = self.client.get\_node(f"ns={self.namespace\_index};s=QualityResult")  
 await quality\_node.write\_value(result.overall\_quality)  
   
 # Write confidence score  
 confidence\_node = self.client.get\_node(f"ns={self.namespace\_index};s=ConfidenceScore")  
 await confidence\_node.write\_value(result.confidence\_score)  
   
 # Write defect count  
 defect\_count\_node = self.client.get\_node(f"ns={self.namespace\_index};s=DefectCount")  
 await defect\_count\_node.write\_value(len(result.defects))  
   
 except Exception as e:  
 raise RuntimeError(f"Failed to send quality decision to PLC: {e}")  
  
class ProductionLineController:  
 """High-level controller for production line integration"""  
   
 def \_\_init\_\_(self, plc\_client: OPCUAClient):  
 self.plc = plc\_client  
 self.quality\_buffer = asyncio.Queue(maxsize=1000)  
 self.performance\_monitor = PerformanceMonitor()  
   
 async def start\_quality\_control\_loop(self):  
 """Main control loop for quality control system"""  
 await self.plc.connect()  
   
 # Start background tasks  
 asyncio.create\_task(self.\_monitor\_system\_health())  
 asyncio.create\_task(self.\_process\_quality\_buffer())  
   
 # Main processing loop  
 vision\_pipeline = RealTimeVisionPipeline(  
 camera\_system=self.camera,  
 defect\_model=self.model,  
 plc\_interface=self.plc  
 )  
   
 async for result in vision\_pipeline.process\_inspection\_stream():  
 await self.quality\_buffer.put(result)  
 await self.\_update\_quality\_metrics(result)  
   
 async def \_monitor\_system\_health(self):  
 """Monitor system health and performance"""  
 while True:  
 health\_status = await self.performance\_monitor.check\_system\_health()  
   
 if health\_status.critical\_issues:  
 await self.\_handle\_critical\_issues(health\_status.critical\_issues)  
   
 await asyncio.sleep(1) # Check every second

### 3. Quality Analytics Implementation

#### 3.1 Statistical Process Control Engine

import numpy as np  
import pandas as pd  
from scipy import stats  
from typing import List, Dict, Tuple  
from dataclasses import dataclass  
from datetime import datetime, timedelta  
  
@dataclass  
class ControlLimits:  
 ucl: float # Upper Control Limit  
 lcl: float # Lower Control Limit  
 center\_line: float  
   
@dataclass  
class SPCAlert:  
 chart\_type: str  
 alert\_type: str # 'OUT\_OF\_CONTROL', 'TREND', 'SHIFT'  
 timestamp: datetime  
 value: float  
 description: str  
  
class StatisticalProcessControl:  
 """Statistical Process Control implementation for quality analytics"""  
   
 def \_\_init\_\_(self, sample\_size: int = 25):  
 self.sample\_size = sample\_size  
 self.x\_bar\_data = []  
 self.r\_data = []  
 self.p\_data = []  
   
 def add\_quality\_sample(self, measurements: List[float]) -> List[SPCAlert]:  
 """Add new quality measurements and check for control violations"""  
 alerts = []  
   
 # Calculate sample statistics  
 x\_bar = np.mean(measurements)  
 r\_value = np.max(measurements) - np.min(measurements)  
   
 self.x\_bar\_data.append(x\_bar)  
 self.r\_data.append(r\_value)  
   
 # Check X-bar chart  
 x\_bar\_alerts = self.\_check\_x\_bar\_control(x\_bar)  
 alerts.extend(x\_bar\_alerts)  
   
 # Check R chart  
 r\_alerts = self.\_check\_r\_control(r\_value)  
 alerts.extend(r\_alerts)  
   
 return alerts  
   
 def \_check\_x\_bar\_control(self, x\_bar: float) -> List[SPCAlert]:  
 """Check X-bar chart for out-of-control conditions"""  
 alerts = []  
   
 if len(self.x\_bar\_data) < self.sample\_size:  
 return alerts  
   
 # Calculate control limits  
 control\_limits = self.\_calculate\_x\_bar\_limits()  
   
 # Rule 1: Point beyond control limits  
 if x\_bar > control\_limits.ucl or x\_bar < control\_limits.lcl:  
 alerts.append(SPCAlert(  
 chart\_type='X\_BAR',  
 alert\_type='OUT\_OF\_CONTROL',  
 timestamp=datetime.now(),  
 value=x\_bar,  
 description=f'Point beyond control limits: {x\_bar:.3f}'  
 ))  
   
 # Rule 2: 7 consecutive points on same side of center line  
 recent\_points = self.x\_bar\_data[-7:]  
 if len(recent\_points) == 7:  
 if all(p > control\_limits.center\_line for p in recent\_points) or \  
 all(p < control\_limits.center\_line for p in recent\_points):  
 alerts.append(SPCAlert(  
 chart\_type='X\_BAR',  
 alert\_type='SHIFT',  
 timestamp=datetime.now(),  
 value=x\_bar,  
 description='7 consecutive points on same side of center line'  
 ))  
   
 # Rule 3: 7 consecutive increasing or decreasing points  
 if len(recent\_points) == 7:  
 increasing = all(recent\_points[i] < recent\_points[i+1] for i in range(6))  
 decreasing = all(recent\_points[i] > recent\_points[i+1] for i in range(6))  
   
 if increasing or decreasing:  
 alerts.append(SPCAlert(  
 chart\_type='X\_BAR',  
 alert\_type='TREND',  
 timestamp=datetime.now(),  
 value=x\_bar,  
 description='7 consecutive trending points'  
 ))  
   
 return alerts  
   
 def \_calculate\_x\_bar\_limits(self) -> ControlLimits:  
 """Calculate control limits for X-bar chart"""  
 x\_double\_bar = np.mean(self.x\_bar\_data)  
 r\_bar = np.mean(self.r\_data)  
   
 # Constants for control chart calculations (n=5 assumed)  
 A2 = 0.577 # Factor for X-bar chart limits  
   
 ucl = x\_double\_bar + A2 \* r\_bar  
 lcl = x\_double\_bar - A2 \* r\_bar  
   
 return ControlLimits(ucl=ucl, lcl=lcl, center\_line=x\_double\_bar)  
  
class QualityMetricsCalculator:  
 """Calculate quality metrics and KPIs"""  
   
 def \_\_init\_\_(self):  
 self.inspection\_history = []  
   
 def calculate\_process\_capability(self, measurements: List[float],   
 specification\_limits: Tuple[float, float]) -> Dict[str, float]:  
 """Calculate process capability indices (Cp, Cpk)"""  
 lsl, usl = specification\_limits  
   
 mean = np.mean(measurements)  
 std = np.std(measurements, ddof=1)  
   
 # Process capability (Cp)  
 cp = (usl - lsl) / (6 \* std)  
   
 # Process capability index (Cpk)  
 cpu = (usl - mean) / (3 \* std)  
 cpl = (mean - lsl) / (3 \* std)  
 cpk = min(cpu, cpl)  
   
 return {  
 'cp': cp,  
 'cpk': cpk,  
 'cpu': cpu,  
 'cpl': cpl,  
 'mean': mean,  
 'std': std  
 }  
   
 def calculate\_defect\_rates(self, time\_window: timedelta = timedelta(hours=1)) -> Dict[str, float]:  
 """Calculate defect rates for different time windows"""  
 current\_time = datetime.now()  
 window\_start = current\_time - time\_window  
   
 # Filter inspections within time window  
 recent\_inspections = [  
 inspection for inspection in self.inspection\_history  
 if inspection.timestamp >= window\_start.timestamp()  
 ]  
   
 if not recent\_inspections:  
 return {'total\_rate': 0.0, 'critical\_rate': 0.0, 'major\_rate': 0.0}  
   
 total\_inspections = len(recent\_inspections)  
 defective\_inspections = len([i for i in recent\_inspections if i.defects])  
 critical\_defects = len([i for i in recent\_inspections   
 if any(d['severity'] >= 3 for d in i.defects)])  
 major\_defects = len([i for i in recent\_inspections   
 if any(d['severity'] == 2 for d in i.defects)])  
   
 return {  
 'total\_rate': defective\_inspections / total\_inspections,  
 'critical\_rate': critical\_defects / total\_inspections,  
 'major\_rate': major\_defects / total\_inspections  
 }

### 4. Edge Computing and Deployment Implementation

#### 4.1 Resource Management and Optimization

import psutil  
import GPUtil  
import threading  
import time  
from typing import Dict, Any  
  
class EdgeResourceManager:  
 """Manage computing resources on edge devices"""  
   
 def \_\_init\_\_(self):  
 self.cpu\_threshold = 80.0 # CPU usage threshold  
 self.memory\_threshold = 85.0 # Memory usage threshold  
 self.gpu\_threshold = 90.0 # GPU usage threshold  
 self.temperature\_threshold = 75.0 # Temperature threshold (°C)  
   
 self.monitoring\_active = False  
 self.performance\_data = {  
 'cpu\_usage': [],  
 'memory\_usage': [],  
 'gpu\_usage': [],  
 'temperature': []  
 }  
   
 def start\_monitoring(self):  
 """Start resource monitoring in background thread"""  
 self.monitoring\_active = True  
 monitoring\_thread = threading.Thread(target=self.\_monitor\_resources)  
 monitoring\_thread.daemon = True  
 monitoring\_thread.start()  
   
 def \_monitor\_resources(self):  
 """Monitor system resources continuously"""  
 while self.monitoring\_active:  
 try:  
 # CPU monitoring  
 cpu\_percent = psutil.cpu\_percent(interval=1)  
 self.performance\_data['cpu\_usage'].append(cpu\_percent)  
   
 # Memory monitoring  
 memory = psutil.virtual\_memory()  
 self.performance\_data['memory\_usage'].append(memory.percent)  
   
 # GPU monitoring  
 gpus = GPUtil.getGPUs()  
 if gpus:  
 gpu\_usage = gpus[0].load \* 100  
 gpu\_temp = gpus[0].temperature  
 self.performance\_data['gpu\_usage'].append(gpu\_usage)  
 self.performance\_data['temperature'].append(gpu\_temp)  
   
 # Check for thermal throttling  
 if gpu\_temp > self.temperature\_threshold:  
 self.\_handle\_thermal\_throttling()  
   
 # Trim history to last 1000 samples  
 for key in self.performance\_data:  
 if len(self.performance\_data[key]) > 1000:  
 self.performance\_data[key] = self.performance\_data[key][-1000:]  
   
 except Exception as e:  
 print(f"Resource monitoring error: {e}")  
   
 time.sleep(1)  
   
 def \_handle\_thermal\_throttling(self):  
 """Handle thermal throttling by reducing processing load"""  
 # Reduce inference frequency  
 # Lower model precision  
 # Increase cooling fan speed  
 pass  
   
 def get\_system\_health(self) -> Dict[str, Any]:  
 """Get current system health status"""  
 if not self.performance\_data['cpu\_usage']:  
 return {'status': 'UNKNOWN', 'details': 'No monitoring data available'}  
   
 current\_cpu = self.performance\_data['cpu\_usage'][-1]  
 current\_memory = self.performance\_data['memory\_usage'][-1]  
 current\_gpu = self.performance\_data['gpu\_usage'][-1] if self.performance\_data['gpu\_usage'] else 0  
 current\_temp = self.performance\_data['temperature'][-1] if self.performance\_data['temperature'] else 0  
   
 status = 'HEALTHY'  
 issues = []  
   
 if current\_cpu > self.cpu\_threshold:  
 status = 'WARNING'  
 issues.append(f'High CPU usage: {current\_cpu:.1f}%')  
   
 if current\_memory > self.memory\_threshold:  
 status = 'WARNING'  
 issues.append(f'High memory usage: {current\_memory:.1f}%')  
   
 if current\_gpu > self.gpu\_threshold:  
 status = 'WARNING'  
 issues.append(f'High GPU usage: {current\_gpu:.1f}%')  
   
 if current\_temp > self.temperature\_threshold:  
 status = 'CRITICAL'  
 issues.append(f'High temperature: {current\_temp:.1f}°C')  
   
 return {  
 'status': status,  
 'cpu\_usage': current\_cpu,  
 'memory\_usage': current\_memory,  
 'gpu\_usage': current\_gpu,  
 'temperature': current\_temp,  
 'issues': issues  
 }

# Pseudocode Implementation

## Manufacturing Quality Control AI Vision System

*Building upon PRD, FRD, NFRD, Architecture Diagram, HLD, and LLD for implementation-ready pseudocode*

## ETVX Framework

### ENTRY CRITERIA

* ✅ LLD completed with all code specifications defined
* ✅ Computer vision class definitions and manufacturing integration interfaces finalized
* ✅ Quality analytics algorithms and edge deployment configurations specified
* ✅ Real-time processing requirements and fault tolerance documented
* ✅ Development team ready for implementation phase

### TASK

Convert low-level design specifications into executable pseudocode that serves as a blueprint for actual code implementation, including complete computer vision processing logic, manufacturing integration flows, quality analytics algorithms, and edge computing system interactions.

### VERIFICATION & VALIDATION

**Verification Checklist:** - [ ] Pseudocode covers all LLD components and methods - [ ] Computer vision processing logic meets <100ms latency requirements - [ ] Manufacturing integration flows handle PLC communication protocols - [ ] Quality analytics algorithms implement statistical process control correctly - [ ] Edge computing logic handles environmental constraints and fault tolerance - [ ] Pseudocode is readable and follows consistent conventions

**Validation Criteria:** - [ ] Pseudocode logic satisfies all functional requirements (FR-001 to FR-038) - [ ] Algorithm complexity meets performance requirements (99.5% accuracy, <100ms processing) - [ ] Manufacturing integration ensures production line compatibility - [ ] Quality analytics validated with Six Sigma methodologies - [ ] Edge deployment logic confirmed for harsh manufacturing environments - [ ] Implementation feasibility confirmed by computer vision and manufacturing teams

### EXIT CRITERIA

* ✅ Complete pseudocode for all system components
* ✅ Executable logic flows ready for code translation
* ✅ Computer vision and manufacturing integration algorithms with performance analysis
* ✅ Comprehensive error handling and recovery procedures for production environment
* ✅ Ready for actual code implementation and testing

### Reference to Previous Documents

This Pseudocode provides executable logic implementing **ALL** previous requirements: - **PRD Business Objectives** → Main application flow optimized for 99.5% accuracy, 30-40% cost reduction - **PRD Key Features** → Complete pseudocode for computer vision, defect detection, operator interface, quality analytics - **FRD Computer Vision Processing (FR-001 to FR-009)** → Real-time image processing pipeline with CNN/ViT models - **FRD Manufacturing Integration (FR-010 to FR-016)** → PLC communication and production line integration logic - **FRD Operator Interface (FR-017 to FR-024)** → Dashboard and defect review interface pseudocode - **FRD Quality Analytics (FR-025 to FR-031)** → Statistical process control and reporting algorithms - **FRD Continuous Learning (FR-032 to FR-038)** → Model training and deployment pipeline logic - **NFRD Performance Requirements** → Optimized algorithms meeting <100ms processing, 99.9% uptime targets - **NFRD Environmental Requirements** → Ruggedized logic for manufacturing conditions (temperature, vibration, dust) - **NFRD Security Requirements** → Industrial cybersecurity and OT/IT segmentation logic - **Architecture Diagram Components** → Edge computing, manufacturing integration, and quality analytics pseudocode - **HLD System Design** → Real-time processing pipeline, PLC integration, and dashboard data flow logic - **LLD Implementation Details** → Direct translation of computer vision classes and manufacturing protocols

### 1. Main Application Flow

MAIN\_MANUFACTURING\_QUALITY\_SYSTEM:  
 INITIALIZE edge\_computing\_environment  
 INITIALIZE computer\_vision\_pipeline  
 INITIALIZE manufacturing\_integration  
 INITIALIZE quality\_analytics\_engine  
 START real\_time\_monitoring\_services  
   
 WHILE production\_line\_active:  
 MONITOR system\_health\_and\_performance  
 PROCESS incoming\_inspection\_triggers  
 UPDATE real\_time\_quality\_metrics  
 HANDLE critical\_alerts\_and\_failures  
 MAINTAIN model\_performance\_and\_accuracy

### 2. Computer Vision Processing Pipeline

COMPUTER\_VISION\_PIPELINE:  
 FUNCTION process\_inspection\_request(product\_id, camera\_trigger):  
 start\_time = GET\_CURRENT\_TIMESTAMP()  
   
 // Image acquisition (5-10ms)  
 raw\_image = CAPTURE\_IMAGE\_FROM\_CAMERA():  
 VALIDATE camera\_connection\_status  
 SET camera\_parameters(resolution=2048x2048, fps=30)  
 CAPTURE high\_resolution\_image  
 IF image\_quality < quality\_threshold:  
 RETRY capture\_attempt  
 IF retry\_failed:  
 RETURN error\_result("Poor image quality")  
   
 // Image preprocessing (5-10ms)  
 processed\_image = PREPROCESS\_IMAGE(raw\_image):  
 denoised\_image = APPLY\_NOISE\_REDUCTION(raw\_image)  
 enhanced\_image = ENHANCE\_CONTRAST\_AND\_BRIGHTNESS(denoised\_image)  
 corrected\_image = CORRECT\_PERSPECTIVE\_DISTORTION(enhanced\_image)  
 normalized\_tensor = NORMALIZE\_AND\_CONVERT\_TO\_TENSOR(corrected\_image)  
 RETURN normalized\_tensor  
   
 // Defect detection inference (20-40ms)  
 defect\_detections = DETECT\_DEFECTS(processed\_image):  
 // Load TensorRT optimized model  
 model\_input = PREPARE\_MODEL\_INPUT(processed\_image)  
 raw\_predictions = CNN\_VISION\_TRANSFORMER\_INFERENCE(model\_input)  
   
 // Post-process predictions  
 filtered\_detections = FILTER\_BY\_CONFIDENCE\_THRESHOLD(raw\_predictions, 0.8)  
 classified\_defects = CLASSIFY\_DEFECT\_TYPES(filtered\_detections)  
 severity\_scored = ASSIGN\_SEVERITY\_LEVELS(classified\_defects)  
   
 RETURN severity\_scored  
   
 // Quality decision logic (1-5ms)  
 quality\_decision = MAKE\_QUALITY\_DECISION(defect\_detections):  
 critical\_defects = COUNT defects WHERE severity >= 3  
 major\_defects = COUNT defects WHERE severity == 2  
 minor\_defects = COUNT defects WHERE severity == 1  
   
 IF critical\_defects > 0:  
 RETURN "FAIL", MIN(confidence\_scores\_of\_critical\_defects)  
 ELIF major\_defects > 2:  
 RETURN "FAIL", MIN(confidence\_scores\_of\_major\_defects)  
 ELIF minor\_defects > 0:  
 RETURN "REVIEW", MIN(confidence\_scores\_of\_minor\_defects)  
 ELSE:  
 RETURN "PASS", 1.0  
   
 // Performance validation  
 total\_processing\_time = GET\_CURRENT\_TIMESTAMP() - start\_time  
 IF total\_processing\_time > 100\_milliseconds:  
 LOG performance\_warning("Processing time exceeded 100ms threshold")  
   
 RETURN inspection\_result(  
 product\_id, quality\_decision, defect\_detections,   
 total\_processing\_time, GET\_CURRENT\_TIMESTAMP()  
 )  
  
FUNCTION continuous\_model\_improvement():  
 WHILE system\_running:  
 // Collect uncertain predictions for human review  
 uncertain\_samples = IDENTIFY\_LOW\_CONFIDENCE\_PREDICTIONS(threshold=0.8)  
   
 FOR each sample IN uncertain\_samples:  
 QUEUE\_FOR\_OPERATOR\_REVIEW(sample)  
   
 // Retrain model when sufficient feedback collected  
 IF feedback\_samples\_count >= 1000:  
 new\_model = RETRAIN\_MODEL\_WITH\_FEEDBACK(feedback\_samples)  
 performance\_improvement = VALIDATE\_MODEL\_PERFORMANCE(new\_model)  
   
 IF performance\_improvement > current\_model\_performance:  
 DEPLOY\_NEW\_MODEL\_VERSION(new\_model)  
 UPDATE\_MODEL\_REGISTRY(new\_model, performance\_metrics)  
   
 SLEEP(model\_improvement\_interval)

### 3. Manufacturing Integration Logic

MANUFACTURING\_INTEGRATION\_SYSTEM:  
 FUNCTION initialize\_plc\_communication():  
 plc\_client = CREATE\_OPC\_UA\_CLIENT(plc\_endpoint\_url)  
   
 TRY:  
 CONNECT\_TO\_PLC(plc\_client)  
 SETUP\_DATA\_SUBSCRIPTIONS(plc\_client):  
 SUBSCRIBE\_TO("InspectionTrigger", callback=handle\_inspection\_trigger)  
 SUBSCRIBE\_TO("ProductionData", callback=handle\_production\_data)  
 SUBSCRIBE\_TO("LineSpeed", callback=handle\_line\_speed\_change)  
   
 RETURN plc\_client  
 CATCH connection\_error:  
 LOG\_ERROR("Failed to connect to PLC", connection\_error)  
 ACTIVATE\_OFFLINE\_MODE()  
 RETURN null  
   
 FUNCTION handle\_inspection\_trigger(trigger\_data):  
 product\_id = EXTRACT\_PRODUCT\_ID(trigger\_data)  
 batch\_number = EXTRACT\_BATCH\_NUMBER(trigger\_data)  
 line\_speed = EXTRACT\_LINE\_SPEED(trigger\_data)  
   
 // Validate trigger data  
 IF VALIDATE\_TRIGGER\_DATA(trigger\_data):  
 inspection\_request = CREATE\_INSPECTION\_REQUEST(  
 product\_id, batch\_number, line\_speed, GET\_CURRENT\_TIMESTAMP()  
 )  
   
 // Process inspection asynchronously  
 ASYNC\_PROCESS\_INSPECTION(inspection\_request, callback=send\_result\_to\_plc)  
 ELSE:  
 LOG\_WARNING("Invalid trigger data received", trigger\_data)  
   
 FUNCTION send\_result\_to\_plc(inspection\_result):  
 TRY:  
 // Write quality decision to PLC  
 WRITE\_TO\_PLC\_NODE("QualityResult", inspection\_result.overall\_quality)  
 WRITE\_TO\_PLC\_NODE("ConfidenceScore", inspection\_result.confidence\_score)  
 WRITE\_TO\_PLC\_NODE("DefectCount", LENGTH(inspection\_result.defects))  
 WRITE\_TO\_PLC\_NODE("ProcessingTime", inspection\_result.processing\_time\_ms)  
   
 // Trigger production line action  
 IF inspection\_result.overall\_quality == "FAIL":  
 WRITE\_TO\_PLC\_NODE("RejectProduct", TRUE)  
 ACTIVATE\_REJECT\_MECHANISM(inspection\_result.product\_id)  
 ELIF inspection\_result.overall\_quality == "REVIEW":  
 WRITE\_TO\_PLC\_NODE("FlagForReview", TRUE)  
 NOTIFY\_QUALITY\_OPERATOR(inspection\_result)  
   
 // Log successful communication  
 LOG\_INFO("Quality result sent to PLC", inspection\_result.product\_id)  
   
 CATCH plc\_communication\_error:  
 LOG\_ERROR("Failed to send result to PLC", plc\_communication\_error)  
 ACTIVATE\_MANUAL\_OVERRIDE\_MODE()  
 ALERT\_PRODUCTION\_SUPERVISOR(inspection\_result, plc\_communication\_error)  
  
FUNCTION production\_line\_synchronization():  
 WHILE production\_active:  
 current\_line\_speed = READ\_FROM\_PLC("LineSpeed")  
   
 // Adjust processing parameters based on line speed  
 IF current\_line\_speed > high\_speed\_threshold:  
 OPTIMIZE\_FOR\_HIGH\_SPEED\_PROCESSING():  
 REDUCE\_IMAGE\_RESOLUTION\_IF\_NECESSARY()  
 ENABLE\_BATCH\_PROCESSING\_MODE()  
 INCREASE\_CONFIDENCE\_THRESHOLD\_SLIGHTLY()  
 ELIF current\_line\_speed < low\_speed\_threshold:  
 OPTIMIZE\_FOR\_HIGH\_ACCURACY():  
 USE\_MAXIMUM\_IMAGE\_RESOLUTION()  
 ENABLE\_DETAILED\_ANALYSIS\_MODE()  
 LOWER\_CONFIDENCE\_THRESHOLD\_FOR\_SENSITIVITY()  
   
 SLEEP(line\_speed\_monitoring\_interval)

### 4. Quality Analytics and Statistical Process Control

QUALITY\_ANALYTICS\_ENGINE:  
 FUNCTION update\_statistical\_process\_control(inspection\_result):  
 // Add new data point to SPC charts  
 quality\_measurement = EXTRACT\_QUALITY\_MEASUREMENT(inspection\_result)  
   
 // Update X-bar and R charts  
 sample\_mean = CALCULATE\_SAMPLE\_MEAN(quality\_measurement)  
 sample\_range = CALCULATE\_SAMPLE\_RANGE(quality\_measurement)  
   
 x\_bar\_data.APPEND(sample\_mean)  
 r\_data.APPEND(sample\_range)  
   
 // Check for out-of-control conditions  
 control\_violations = CHECK\_CONTROL\_CHART\_RULES():  
 violations = []  
   
 // Rule 1: Point beyond control limits  
 control\_limits = CALCULATE\_CONTROL\_LIMITS(x\_bar\_data, r\_data)  
 IF sample\_mean > control\_limits.upper OR sample\_mean < control\_limits.lower:  
 violations.APPEND("OUT\_OF\_CONTROL\_POINT")  
   
 // Rule 2: 7 consecutive points on same side of center line  
 recent\_points = GET\_LAST\_N\_POINTS(x\_bar\_data, 7)  
 IF ALL\_ABOVE\_CENTER\_LINE(recent\_points) OR ALL\_BELOW\_CENTER\_LINE(recent\_points):  
 violations.APPEND("PROCESS\_SHIFT")  
   
 // Rule 3: 7 consecutive trending points  
 IF CONSECUTIVE\_TREND\_DETECTED(recent\_points, 7):  
 violations.APPEND("PROCESS\_TREND")  
   
 // Rule 4: 2 out of 3 points beyond 2-sigma limits  
 recent\_3\_points = GET\_LAST\_N\_POINTS(x\_bar\_data, 3)  
 beyond\_2\_sigma = COUNT\_POINTS\_BEYOND\_2\_SIGMA(recent\_3\_points, control\_limits)  
 IF beyond\_2\_sigma >= 2:  
 violations.APPEND("PROCESS\_INSTABILITY")  
   
 RETURN violations  
   
 // Generate alerts for control violations  
 FOR each violation IN control\_violations:  
 alert = CREATE\_SPC\_ALERT(violation, sample\_mean, GET\_CURRENT\_TIMESTAMP())  
 SEND\_ALERT\_TO\_QUALITY\_MANAGER(alert)  
 LOG\_QUALITY\_ALERT(alert)  
   
 FUNCTION calculate\_process\_capability\_indices():  
 recent\_measurements = GET\_RECENT\_MEASUREMENTS(time\_window=24\_hours)  
   
 IF LENGTH(recent\_measurements) >= 30: // Minimum sample size  
 process\_mean = CALCULATE\_MEAN(recent\_measurements)  
 process\_std = CALCULATE\_STANDARD\_DEVIATION(recent\_measurements)  
   
 // Get specification limits from product requirements  
 upper\_spec\_limit = GET\_UPPER\_SPECIFICATION\_LIMIT()  
 lower\_spec\_limit = GET\_LOWER\_SPECIFICATION\_LIMIT()  
   
 // Calculate Cp (Process Capability)  
 cp = (upper\_spec\_limit - lower\_spec\_limit) / (6 \* process\_std)  
   
 // Calculate Cpk (Process Capability Index)  
 cpu = (upper\_spec\_limit - process\_mean) / (3 \* process\_std)  
 cpl = (process\_mean - lower\_spec\_limit) / (3 \* process\_std)  
 cpk = MIN(cpu, cpl)  
   
 // Interpret capability results  
 capability\_assessment = ASSESS\_PROCESS\_CAPABILITY(cp, cpk):  
 IF cpk >= 1.33:  
 RETURN "EXCELLENT\_CAPABILITY"  
 ELIF cpk >= 1.0:  
 RETURN "ADEQUATE\_CAPABILITY"  
 ELIF cpk >= 0.67:  
 RETURN "MARGINAL\_CAPABILITY"  
 ELSE:  
 RETURN "INADEQUATE\_CAPABILITY"  
   
 // Update capability dashboard  
 UPDATE\_CAPABILITY\_DASHBOARD(cp, cpk, capability\_assessment)  
   
 // Alert if capability deteriorates  
 IF cpk < minimum\_acceptable\_cpk:  
 SEND\_CAPABILITY\_ALERT(cpk, capability\_assessment)  
  
FUNCTION generate\_quality\_reports():  
 WHILE system\_running:  
 current\_time = GET\_CURRENT\_TIMESTAMP()  
   
 // Generate hourly quality summary  
 IF current\_time.minute == 0: // Top of each hour  
 hourly\_data = COLLECT\_HOURLY\_QUALITY\_DATA()  
 hourly\_report = GENERATE\_HOURLY\_REPORT(hourly\_data):  
 total\_inspections = COUNT\_INSPECTIONS(last\_hour)  
 defect\_rate = CALCULATE\_DEFECT\_RATE(last\_hour)  
 average\_confidence = CALCULATE\_AVERAGE\_CONFIDENCE(last\_hour)  
 processing\_performance = CALCULATE\_AVERAGE\_PROCESSING\_TIME(last\_hour)  
   
 RETURN quality\_summary(  
 total\_inspections, defect\_rate, average\_confidence,   
 processing\_performance, GET\_CURRENT\_TIMESTAMP()  
 )  
   
 SEND\_REPORT\_TO\_STAKEHOLDERS(hourly\_report)  
 STORE\_REPORT\_IN\_DATABASE(hourly\_report)  
   
 // Generate daily quality report  
 IF current\_time.hour == 0 AND current\_time.minute == 0: // Midnight  
 daily\_data = COLLECT\_DAILY\_QUALITY\_DATA()  
 daily\_report = GENERATE\_COMPREHENSIVE\_DAILY\_REPORT(daily\_data)  
 SEND\_DAILY\_REPORT\_TO\_MANAGEMENT(daily\_report)  
   
 SLEEP(60) // Check every minute

### 5. Edge Computing and System Health Monitoring

EDGE\_COMPUTING\_SYSTEM:  
 FUNCTION monitor\_system\_health():  
 WHILE system\_running:  
 // Monitor hardware resources  
 cpu\_usage = GET\_CPU\_UTILIZATION()  
 memory\_usage = GET\_MEMORY\_UTILIZATION()  
 gpu\_usage = GET\_GPU\_UTILIZATION()  
 gpu\_temperature = GET\_GPU\_TEMPERATURE()  
 disk\_usage = GET\_DISK\_UTILIZATION()  
   
 // Check for resource constraints  
 resource\_alerts = CHECK\_RESOURCE\_THRESHOLDS():  
 alerts = []  
   
 IF cpu\_usage > 80:  
 alerts.APPEND("HIGH\_CPU\_USAGE")  
 OPTIMIZE\_CPU\_INTENSIVE\_PROCESSES()  
   
 IF memory\_usage > 85:  
 alerts.APPEND("HIGH\_MEMORY\_USAGE")  
 CLEANUP\_UNUSED\_MEMORY\_BUFFERS()  
   
 IF gpu\_usage > 90:  
 alerts.APPEND("HIGH\_GPU\_USAGE")  
 OPTIMIZE\_GPU\_INFERENCE\_BATCHING()  
   
 IF gpu\_temperature > 75:  
 alerts.APPEND("HIGH\_GPU\_TEMPERATURE")  
 ACTIVATE\_THERMAL\_THROTTLING()  
   
 IF disk\_usage > 90:  
 alerts.APPEND("HIGH\_DISK\_USAGE")  
 ARCHIVE\_OLD\_INSPECTION\_IMAGES()  
   
 RETURN alerts  
   
 // Handle critical resource issues  
 FOR each alert IN resource\_alerts:  
 LOG\_SYSTEM\_ALERT(alert)  
 TAKE\_CORRECTIVE\_ACTION(alert)  
   
 IF alert == "CRITICAL":  
 NOTIFY\_SYSTEM\_ADMINISTRATOR(alert)  
   
 // Monitor network connectivity  
 network\_status = CHECK\_NETWORK\_CONNECTIVITY():  
 plc\_connection = TEST\_PLC\_CONNECTION()  
 database\_connection = TEST\_DATABASE\_CONNECTION()  
 external\_api\_connection = TEST\_EXTERNAL\_API\_CONNECTION()  
   
 IF NOT plc\_connection:  
 ACTIVATE\_OFFLINE\_MODE()  
 BUFFER\_QUALITY\_DECISIONS\_LOCALLY()  
   
 RETURN network\_connectivity\_status(  
 plc\_connection, database\_connection, external\_api\_connection  
 )  
   
 // Update system health dashboard  
 system\_health = COMPILE\_SYSTEM\_HEALTH\_STATUS(  
 cpu\_usage, memory\_usage, gpu\_usage, gpu\_temperature,  
 disk\_usage, network\_status, resource\_alerts  
 )  
   
 UPDATE\_HEALTH\_DASHBOARD(system\_health)  
   
 SLEEP(system\_monitoring\_interval)  
  
FUNCTION handle\_system\_failures():  
 WHILE system\_running:  
 TRY:  
 MONITOR\_CRITICAL\_PROCESSES()  
   
 CATCH process\_failure:  
 failure\_type = IDENTIFY\_FAILURE\_TYPE(process\_failure)  
   
 SWITCH failure\_type:  
 CASE "CAMERA\_FAILURE":  
 SWITCH\_TO\_BACKUP\_CAMERA()  
 NOTIFY\_MAINTENANCE\_TEAM("Camera failure detected")  
   
 CASE "MODEL\_INFERENCE\_FAILURE":  
 RELOAD\_INFERENCE\_MODEL()  
 IF reload\_failed:  
 SWITCH\_TO\_BACKUP\_MODEL()  
   
 CASE "PLC\_COMMUNICATION\_FAILURE":  
 ATTEMPT\_PLC\_RECONNECTION()  
 IF reconnection\_failed:  
 ACTIVATE\_MANUAL\_OVERRIDE\_MODE()  
 ALERT\_PRODUCTION\_SUPERVISOR()  
   
 CASE "DATABASE\_FAILURE":  
 SWITCH\_TO\_LOCAL\_STORAGE\_MODE()  
 QUEUE\_DATA\_FOR\_SYNC\_WHEN\_RECOVERED()  
   
 DEFAULT:  
 LOG\_UNKNOWN\_FAILURE(process\_failure)  
 ATTEMPT\_GRACEFUL\_RESTART()  
   
 // Log failure and recovery actions  
 LOG\_FAILURE\_EVENT(failure\_type, recovery\_actions, GET\_CURRENT\_TIMESTAMP())  
  
FUNCTION data\_backup\_and\_recovery():  
 WHILE system\_running:  
 // Backup critical data every 4 hours  
 IF current\_time.hour % 4 == 0 AND current\_time.minute == 0:  
 BACKUP\_QUALITY\_DATABASE()  
 BACKUP\_MODEL\_CONFIGURATIONS()  
 BACKUP\_SYSTEM\_CONFIGURATIONS()  
   
 // Verify backup integrity  
 backup\_verification = VERIFY\_BACKUP\_INTEGRITY()  
 IF NOT backup\_verification.success:  
 ALERT\_SYSTEM\_ADMINISTRATOR("Backup verification failed")  
   
 // Archive old inspection images daily  
 IF current\_time.hour == 2 AND current\_time.minute == 0: // 2 AM daily  
 old\_images = FIND\_IMAGES\_OLDER\_THAN(retention\_period=30\_days)  
 ARCHIVE\_TO\_LONG\_TERM\_STORAGE(old\_images)  
 DELETE\_ARCHIVED\_IMAGES\_FROM\_LOCAL\_STORAGE(old\_images)  
   
 SLEEP(3600) // Check every hour

### 6. Security and Compliance Implementation

SECURITY\_SYSTEM:  
 FUNCTION implement\_industrial\_cybersecurity():  
 // Network segmentation  
 CONFIGURE\_NETWORK\_SEGMENTATION():  
 ISOLATE\_OT\_NETWORK\_FROM\_IT\_NETWORK()  
 IMPLEMENT\_FIREWALL\_RULES\_FOR\_OT\_IT\_COMMUNICATION()  
 ENABLE\_NETWORK\_MONITORING\_AND\_INTRUSION\_DETECTION()  
   
 // Device authentication  
 SETUP\_CERTIFICATE\_BASED\_AUTHENTICATION():  
 GENERATE\_DEVICE\_CERTIFICATES()  
 CONFIGURE\_MUTUAL\_TLS\_AUTHENTICATION()  
 IMPLEMENT\_CERTIFICATE\_ROTATION\_POLICY()  
   
 // Data encryption  
 IMPLEMENT\_DATA\_ENCRYPTION():  
 ENCRYPT\_DATA\_AT\_REST\_WITH\_AES\_256()  
 ENCRYPT\_DATA\_IN\_TRANSIT\_WITH\_TLS\_1\_3()  
 IMPLEMENT\_SECURE\_KEY\_MANAGEMENT()  
   
 // Access control  
 CONFIGURE\_ROLE\_BASED\_ACCESS\_CONTROL():  
 DEFINE\_USER\_ROLES(operator, quality\_manager, engineer, admin)  
 IMPLEMENT\_LEAST\_PRIVILEGE\_ACCESS()  
 ENABLE\_SESSION\_MANAGEMENT\_AND\_TIMEOUTS()  
   
 // Audit logging  
 ENABLE\_COMPREHENSIVE\_AUDIT\_LOGGING():  
 LOG\_ALL\_USER\_ACTIONS()  
 LOG\_ALL\_SYSTEM\_EVENTS()  
 LOG\_ALL\_QUALITY\_DECISIONS()  
 IMPLEMENT\_TAMPER\_EVIDENT\_LOGGING()  
  
FUNCTION compliance\_monitoring():  
 WHILE system\_running:  
 // ISO 9001 compliance checks  
 iso\_compliance = CHECK\_ISO\_9001\_COMPLIANCE():  
 VERIFY\_QUALITY\_MANAGEMENT\_PROCESSES()  
 VALIDATE\_DOCUMENTATION\_COMPLETENESS()  
 CHECK\_CONTINUOUS\_IMPROVEMENT\_ACTIVITIES()  
   
 // Six Sigma compliance  
 six\_sigma\_compliance = CHECK\_SIX\_SIGMA\_COMPLIANCE():  
 VALIDATE\_STATISTICAL\_PROCESS\_CONTROL()  
 VERIFY\_PROCESS\_CAPABILITY\_MEASUREMENTS()  
 CHECK\_DEFECT\_REDUCTION\_INITIATIVES()  
   
 // Generate compliance reports  
 IF compliance\_reporting\_due:  
 compliance\_report = GENERATE\_COMPLIANCE\_REPORT(  
 iso\_compliance, six\_sigma\_compliance  
 )  
 SUBMIT\_COMPLIANCE\_REPORT\_TO\_AUTHORITIES(compliance\_report)  
   
 SLEEP(compliance\_check\_interval)