QUALITY CONTROL AND DEFECT DETECTION USING IMAGE

ABSTRACT:

In today's manufacturing industry, ensuring product quality is a critical aspect of maintaining efficiency, reducing waste, and enhancing customer satisfaction. Traditional quality control methods rely heavily on manual inspection, which is time-consuming, prone to human error, and inconsistent. To address these challenges, this project focuses on developing an AI-driven automated defect detection system using Computer Vision (CV) and deep learning**.** The proposed system leverages Google Cloud's Vertex AI AutoML to train and deploy an image classification model capable of detecting and categorizing product defects with high accuracy.

The primary objective of this project is to implement real-time defect detection by integrating cloud-based AI solutions with camera systems. The project follows a structured workflow, including data collection, image preprocessing, model training, evaluation, and deployment**.** The dataset consists of images of screws, categorized into defective and non-defective classes. By utilizing Google Cloud Storage (GCS), Cloud Functions, and Cloud Monitoring, the system ensures seamless image processing, prediction, and logging for continuous performance tracking.

This project also explores early defect prediction by analyzing historical image data, enabling proactive quality control. The integration of edge computing and cloud-based inference ensures low-latency, high-speed detection, making it suitable for real-time manufacturing environments. The significance of this project lies in its ability **to** automate quality assurance, reduce manual labor costs, and improve defect detection accuracy, ultimately enhancing production efficiency.

By implementing this AI-driven solution, manufacturers can achieve higher product reliability, reduced operational costs, and improved quality standards, paving the way for smart, automated, and scalable quality control systems in modern industries.

CHAPTER 1

INTRODUCTION

1.1AIM &OBJECTIVE OF THE PROJECT

1.1.1 Aim

The primary aim of this project is to design and implement an AI-powered quality control system that can efficiently detect and classify product defects using advanced image analysis and deep learning techniques. By integrating computer vision and artificial intelligence, the system will ensure high-precision defect identification, real-time monitoring, and process automation to significantly enhance quality control in manufacturing. This AI-driven approach will minimize human errors, reduce operational costs, improve production efficiency, and ensure consistent product quality across large-scale manufacturing processes.

By leveraging cutting-edge machine learning models and cloud-based technologies, the project will facilitate seamless real-time analysis and defect classification, thereby streamlining industrial workflows. The ultimate goal is to transform traditional manual inspection processes into a highly accurate, automated, and data-driven quality assurance system that proactively identifies potential defects before they escalate, ensuring enhanced reliability and efficiency in production.

1.1.2 Objectives

To achieve the aim of this project, the following key objectives have been outlined:

1. **Automated Defect Detection** – Develop an AI-based computer vision model capable of automatically identifying and classifying product defects from image datasets with high accuracy and efficiency, reducing reliance on manual inspections
2. **Quality Assurance Monitoring** – Implement a real-time monitoring framework that continuously analyzes manufacturing images to ensure quality standards are met throughout the production lifecycle. This will minimize defects in final products and enhance overall process efficiency.

3. **Early Defect Prediction** – Utilize historical image data and deep learning models to predict the likelihood of defects occurring before they become critical, enabling proactive quality control measures and minimizing production losses.
4. **Real-Time Analysis & Cloud Integration** – Develop a real-time defect detection system by integrating AI models with edge computing, IoT-enabled cameras, or cloud-based solutions to enable instant quality assessments and faster decision-making in manufacturing environments

1.1.3 ORGANISATIONAL PROFILE:

Name of the Organization: BEAU ROI TECHNOLOGIES PRIVATE LIMITED

Beau Roi Technologies Private Limited is a pioneering company specializing in cutting-edge technology solutions across various domains, including artificial intelligence, machine learning, automation, and cloud computing. The company is dedicated to delivering innovative and efficient solutions to enhance business operations, streamline workflows, and optimize performance.

With a strong emphasis on research and development, Beau Roi Technologies has built a reputation for excellence by consistently pushing the boundaries of technological innovation. The company's expertise spans software development, intelligent automation, cloud-based applications, and enterprise solutions.

History

Founded in 2021, Beau Roi Technologies began as a startup with a vision to revolutionize digital transformation through advanced AI-driven solutions. Over the years, the company has grown into a recognized leader, catering to clients across diverse industries, including education, healthcare, finance, and manufacturing.

With a commitment to quality and innovation, the company has successfully deployed numerous projects, helping businesses leverage technology for improved efficiency and decision-making. Its expertise in cloud computing and artificial intelligence has enabled organizations to adopt scalable, intelligent solutions tailored to their specific needs.

Future Prospects:

BEAU ROI TECHNOLOGIES PRIVATE LIMITED continues to innovate in the field of AI-driven education technology. Plans for future enhancements include the integration of Generative AI for subjective answer evaluation, multilingual support, and adaptive learning analytics to personalize student assessments further.

Contact Details:

LOCATION:
M124, Cactus Corporate Coworking, #173, 6th Floor, Block B, TECCI Park,Old Mahabalipuram Road (OMR), Elcot Sez,Sholinganallur, Chennai - 600 119,Tamil Nadu, India.

WEBSITE:
https://beauroi.com/

1.2 PROJECT BACKGROUND:

1.2.1 Problem Statement:

Manufacturing industries face significant challenges in ensuring consistent product quality due to manual defect detection, which is often time-consuming, error-prone, and inefficient. traditional inspection methods rely on human workers, leading to inconsistencies, delays, and increased costs. additionally, early-stage defect identification is difficult, resulting in high rejection rates and wastage. there is a need for an automated, accurate, and scalable defect detection system to improve quality control in production lines.

1.2.2 Challenges:

1. **Variability in defects** – defects may appear in different shapes, sizes, and textures, making accurate detection complex.
2. **High processing speed requirement** – real-time defect detection demands fast image processing and analysis to avoid production delays.
3. **Data collection and annotation** – a large, high-quality dataset with correctly labeled defective and non-defective images is necessary for training ai models.

4. **Integration with manufacturing systems** – the ai-based system must seamlessly integrate with existing production lines and camera-based inspection units.
5. **False positives and false negatives** – ensuring the ai model minimizes misclassifications to avoid unnecessary rejections or overlooked defects.
6. **Cloud-based implementation** – real-time processing on cloud platforms must be optimized for speed, cost, and reliability.
7. **Scalability and maintenance** – the system should be scalable for different types of defects and require minimal retraining for new defect categories.
8. **Fast retrieval of cloud credentials** – ensuring seamless authentication and fast retrieval of credentials from google cloud to avoid delays in processing and maintaining security in real-time defect detection.

1.2.3 Proposed solution:

To address these challenges, an ai-driven automated defect detection system will be developed using vertex ai automl for image classification. the system will leverage deep learning models trained on labeled defect images to accurately classify products as defective or non-defective. the solution involves:

1. **Automated image analysis** – using google cloud-based automl vision to detect defects from real-time product images.
2. **Real-time defect detection** – implementing cloud functions to trigger instant predictions when images are uploaded to google cloud storage.
3. **Cloud monitoring and logging** – using cloud monitoring and logging to track system performance and improve accuracy over time.
4. **Edge computing integration** – deploying models on edge devices to reduce processing latency in real-time manufacturing environments.
5. **Optimized cloud security and credential management** – ensuring fast retrieval of authentication credentials in cloud-based processing to improve system efficiency.
6. **Continuous improvement** – model retraining using new defect data to improve accuracy and adapt to evolving manufacturing conditions.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM:

Existing quality control methods primarily depend on manual inspection or basic machine vision techniques. while these approaches may be effective for simple defect detection tasks, they often fall short in handling more complex scenarios that involve variations in screw size, orientation, and lighting conditions. manual inspection is inherently prone to human error, fatigue, and inefficiency, leading to inconsistencies in quality assessment. furthermore, reliance on human inspectors increases operational costs and slows down the production process.

Current automated defect detection solutions, though an improvement over manual methods, often lack the adaptability needed to handle a wide range of defect types and product variations. many of these systems require extensive calibration and fine-tuning to function correctly under different lighting conditions and manufacturing settings. this limits their scalability and makes it challenging to deploy them across diverse production environments. additionally, some machine vision systems struggle with real-time defect identification, leading to delays in quality control processes.

The need for a more advanced, scalable, and adaptable defect detection solution is evident. an ai-powered approach, capable of learning from vast amounts of image data and adapting to different defect patterns, is essential to improve quality control, reduce manufacturing defects, and enhance overall efficiency in industrial production.

2.2 PROPOSED SYSTEM:

The proposed system employs AutoML with NAS, transfer learning, and ensemble methods for defect detection, processing raw images of screws to identify anomalies. This end-to-end approach eliminates the need for manual inspection or traditional feature extraction techniques. The system uses diverse training datasets augmented with synthetic variations to improve robustness. By normalizing input images and employing advanced preprocessing techniques, the model achieves high accuracy in defect classification. Testing is conducted in simulated environments replicating real-world conditions, ensuring reliability and scalability.

2.3 FEASIBILITY STUDY:

The feasibility study thoroughly examines the practicality and viability of implementing the AI-powered automated defect detection system across multiple dimensions, including technical, operational, economic, and environmental aspects. This analysis ensures that the proposed system is not only innovative but also sustainable, efficient, and beneficial for industrial applications in the long run.

2.4.1 Technical Feasibility:

From a technical standpoint, the project leverages cutting-edge computer vision, deep learning, and cloud computing technologies to develop a robust defect detection system. The use of Google Cloud’s Vertex AI AutoML simplifies the training and deployment of the model, enabling high accuracy in identifying and classifying defective and non-defective products. The system is designed to integrate seamlessly with real-time image capture devices and cloud storage solutions, ensuring smooth operation in industrial settings. Additionally, the incorporation of edge computing capabilities allows for rapid on-site defect analysis, reducing latency and improving efficiency.

2.4.2 Operational Feasibility:

The operational feasibility of this system lies in its ability to enhance production efficiency while reducing manual labor dependency. Traditional quality control methods often require extensive human intervention, which can lead to errors and inconsistencies. By automating the defect detection process, this AI-driven system ensures uniformity, reliability, and speed in identifying product defects. The integration of real-time image processing allows industries to monitor production lines continuously, ensuring that defective products are detected and removed before they reach the market.

2.4.3 Economic Feasibility:

Implementing an AI-based defect detection system significantly reduces long-term operational costs. While the initial investment may include expenses related to cloud infrastructure, model training, and system deployment, the return on investment (ROI) is substantial. By minimizing product defects, reducing material wastage, and optimizing production processes, industries can experience a notable increase in cost savings and profitability. Additionally, the system's ability to prevent defective products from reaching customers helps avoid potential losses due to recalls, customer complaints, or reputational damage.

2.4.4 Environmental Feasibility:

From an environmental perspective, this project contributes to sustainable manufacturing practices by reducing waste and optimizing resource utilization. The AI-driven defect detection system minimizes the number of defective products discarded, thus preventing unnecessary material consumption and reducing industrial waste. By incorporating smart quality control measures, industries can work toward achieving eco-friendly production standards, aligning with global sustainability goals.

2.4.5 Schedule Feasibility

Schedule feasibility determines whether the AI-based automated defect detection system can be developed and deployed within the given timeframe. The proposed project is planned for three months (12 weeks) with structured phases to ensure timely completion.

Week1-2:Requirement Analysis & Data Collection

•Define project objectives, scope, and technical stack.

• Collect and preprocess screw image datasets for training.

Week 3-4: System Design

• Design the AI model architecture for defect detection.

• Plan cloud storage integration and real-time analysis pipeline.

Week 5-7: Implementation

• Develop and train the defect detection model using Vertex AI AutoML.

• Implement real-time image analysis using Google Cloud Functions.

• Integrate PyCharm with Google Cloud Storage for seamless data handling.

Week 8-9: Testing & Optimization
• Test model accuracy on real-world screw images.
• Optimize cloud functions for fast inference and low latency.

Week 10-12: Deployment & Final Review
• Deploy the AI model on Google Cloud for real-time defect detection.
• Conduct performance testing, refine logging, and finalize project documentation..

2.4 HARDWARE & SOFTWARE SPECIFICATIONS

The implementation of the Automated Defect Detection System requires a high-performance computing setup to handle deep learning model training, real-time image processing, and cloud integration. The system demands a powerful processor, sufficient RAM, and high-speed storage for smooth execution. Software requirements include Python, TensorFlow/PyTorch for model training, and OpenCV for image processing. Cloud tools such as Google Cloud Storage, Vertex AI, AutoML, and Cloud Functions enable efficient deployment, real-time analysis, and automated defect detection. PyCharm serves as the development environment, while Windows 10/11 ensures compatibility with AI frameworks and cloud services.

2.4.1 Hardware Requirements

A high-performance computing setup is essential for handling deep learning model training, real-time image processing, and cloud integration. Powerful processors, sufficient RAM, and fast storage ensure smooth execution and efficient defect detection.

• Processor: Intel Core i5 or higher (Recommended: Intel Core i7 or AMD Ryzen 7 for better performance)

A high-performance processor is crucial for ensuring smooth execution of AI models, handling real-time image processing, and managing cloud-based operations efficiently. Faster processors enhance the training speed of deep learning models and allow seamless defect detection in real time.

• **RAM: 8 GB or more (Recommended: 16 GB or higher for faster computations)**

Adequate memory is essential for efficiently managing large image datasets, performing intensive AI computations, and ensuring quick model inference. More RAM allows the system to process multiple high-resolution images simultaneously without performance bottlenecks

• **Storage: Minimum 500 GB HDD or SSD (Recommended: 1 TB SSD for faster data retrieval and model training)**

High-speed SSD storage significantly improves data retrieval speeds, enabling faster model training and inference. Storing large datasets and accessing images from storage becomes quicker, reducing latency in AI-based defect detection

2.4.2 Software Requirements

Quality control and defect detection using image utilizes Python with TensorFlow/PyTorch for AI model development, OpenCV for image processing, and Keras for neural network training. Cloud tools like Google Cloud Storage, Vertex AI, and AutoML facilitate real-time defect detection and seamless deployment.

2.4.2.1 PyCharm

A robust Integrated Development Environment (IDE) designed for Python development, providing advanced debugging capabilities, intelligent code completion, and seamless integration with AI libraries. It enhances productivity and streamlines AI model development.

2.4.2.2 Google Cloud Platform (GCP)

A cloud computing platform that enables scalable AI model deployment, efficient cloud storage management, and seamless execution of machine learning workflows. It provides the necessary infrastructure for AI-based defect detection.

2.4.2.3 Programming Language

Python

The primary programming language for AI and machine learning due to its simplicity, extensive support for deep learning libraries, and strong community support. It facilitates model development, training, and cloud-based inference.

2.4.2.4 Libraries

1. **TensorFlow/PyTorch**

Advanced deep learning frameworks used to build and train AI models for defect detection. TensorFlow offers high-level APIs and scalability, while PyTorch provides flexibility for creating custom model architectures.

2.**OpenCV**

A powerful computer vision library used for preprocessing images, performing defect localization, and extracting relevant features from product images to enhance model accuracy.

3.**Keras**

A high-level deep learning API integrated with TensorFlow, simplifying the process of designing, training, and deploying neural networks for defect classification.

4.**Time, OS**

Essential Python libraries used for time-based operations, system-level interactions, and file management in the AI workflow.

2.4.2.5 Cloud Tools

- **Google Cloud Storage (GCS)**

A scalable cloud storage solution for securely storing image datasets, real-time captured images, and AI model artifacts used in defect detection

- **Vertex AI**

A managed machine learning service in GCP that automates AI model training, hyperparameter tuning, and deployment, making it easier to implement defect detection solutions.

- **AutoML**

A no-code/low-code tool within Vertex AI that simplifies AI model training by automatically selecting the best algorithms and hyperparameters for image classification tasks.

- **Cloud Function**

A serverless execution environment that triggers real-time AI model predictions whenever an image is uploaded to the cloud storage bucket, ensuring automation in defect detection workflows.

- **Cloud Monitoring**

A cloud-based monitoring service that provides real-time insights into AI application performance, latency, and system health, ensuring smooth operation of defect detection models.

- **Cloud Logging**

A tool for capturing and analyzing event logs, system errors, and performance data to enhance debugging and optimize AI-based defect detection processes.

2.4.2.6 Operating System

Windows 10/11

A widely used and compatible operating system that supports AI development tools, cloud integration, and hardware acceleration for deep learning applications, ensuring an efficient environment for defect detection implementation.

Conclusion

The integration of advanced AI models with cloud-based tools ensures efficient and accurate defect detection in real-time. This approach enhances quality control processes, reducing manual effort and improving overall production efficiency.

CHAPTER 3

SYSTEM DESIGN

3.1 USE DIAGRAM

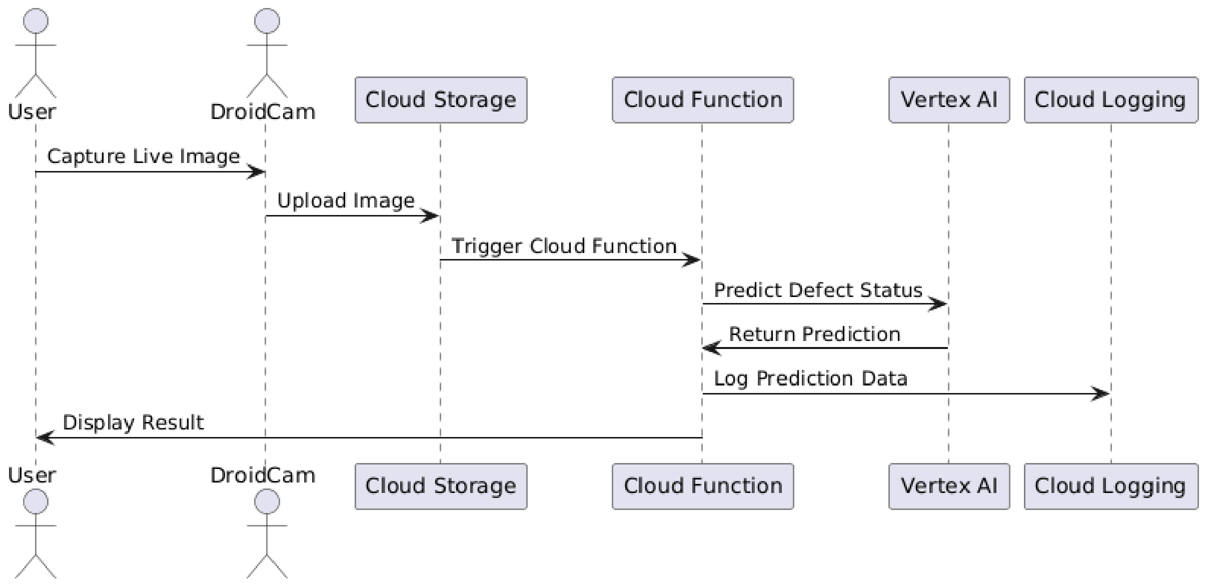


Fig 3.1 Use cases diagram

3.2 DATAFLOW DIAGRAM:

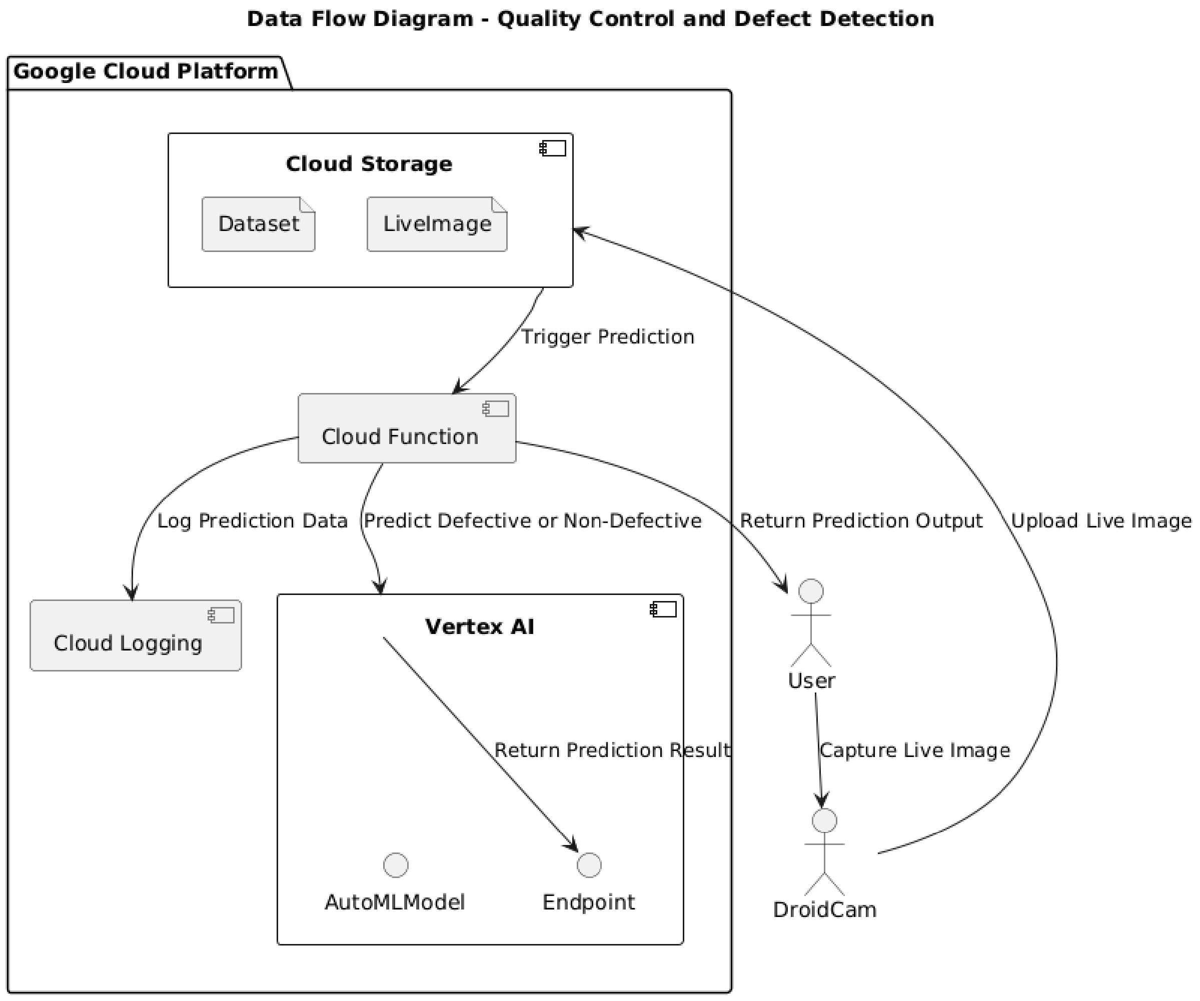


Fig 3.2 Dataflow Diagram

3.3 DATABASE DESIGN:

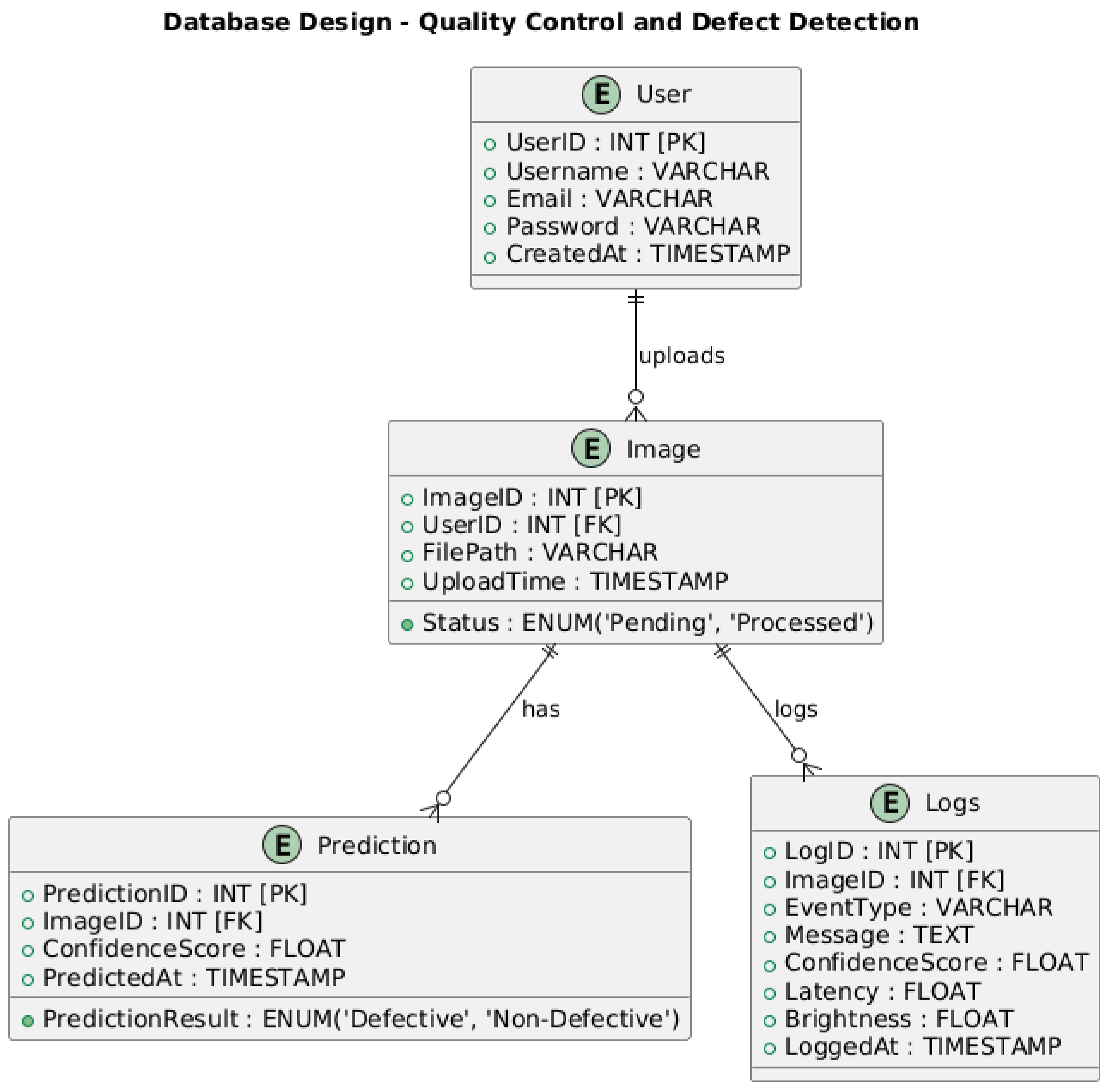


Fig 3.3 Database Diagram

CHAPTER 4:

IMPLEMENTATION

Quality Control and Defect Detection Using Image encompasses dataset collection, image preprocessing, model development, cloud integration, and rigorous testing. By adopting a systematic methodology, the project not only achieves high accuracy in defect detection but also ensures robust, real-time performance suitable for industrial applications.

4.1 PROJECT PHASES

The project was divided into a series of well-defined phases. Each phase built upon the previous steps to create a cohesive and efficient workflow. The following sections explain each phase in depth.

4.1.1 Dataset Collection

The foundation of any computer vision system is a high-quality dataset. For this project, our primary focus was on assembling a comprehensive repository of screw images that include both defective and non-defective samples. The dataset was curated from several sources:

- **Industrial Datasets:** We sourced images from publicly available industrial datasets known for their stringent quality controls. These images provided a baseline for standard screw configurations and common defect types.
- **Manual Capture:** To supplement the dataset, we utilized high-resolution cameras to manually capture images from different angles, under various lighting conditions, and at multiple resolutions. This approach ensured that the dataset was representative of real-world conditions encountered on production lines.
- **Quality Assurance:** Each image underwent a preliminary screening process to verify its relevance and quality. Criteria such as resolution, focus, and the clarity of defect features were rigorously evaluated. In cases where images did not meet quality standards, additional samples were captured.

This phase also involved annotating the dataset. Defects were carefully labeled, ensuring that the training set could effectively teach the model the distinguishing features of a defective screw. This step was crucial to enable the supervised learning approach adopted in the project.

4.1.2 Image Preprocessing

Preprocessing the images was a critical step to ensure that the subsequent machine learning model could operate with high accuracy and efficiency. The preprocessing pipeline included several key procedures:

- **Resizing:** Images were resized to a standardized dimension to balance the trade-off between computational efficiency and preservation of detail. A fixed resolution was chosen to match the input size requirements of the model architecture.
- **Normalization:** To reduce the variability due to lighting and contrast, normalization techniques were applied. This ensured that pixel intensity values were scaled appropriately, enhancing the model's ability to learn robust features.
- **Augmentation:** Data augmentation was employed to artificially expand the dataset and introduce variability. Techniques such as horizontal and vertical flipping, random rotations, brightness adjustments, and slight zoom operations were implemented. This augmentation helped simulate different environmental conditions (e.g., varying lighting, angle differences) that the model might encounter during real-time operation.
- **Noise Reduction:** In some instances, images contained artifacts or noise. A series of denoising filters, such as Gaussian blur, were applied to ensure that these artifacts did not interfere with feature extraction.

Each of these preprocessing steps was designed to reduce overfitting and improve generalization, enabling the model to perform reliably across diverse scenarios.

4.1.3 Data Splitting

For effective training and unbiased evaluation of the model, the dataset was partitioned into distinct subsets:

- **Training Set:** Approximately 80% of the images were allocated to the training set. This subset was used to train the machine learning model, allowing it to learn distinguishing features and patterns associated with both defective and non-defective screws.

- **Testing Set:** The remaining 20% of the images were reserved for testing. This set provided a reliable measure of the model's performance on unseen data. By holding out this portion of the dataset, we ensured that the evaluation metrics (accuracy, precision, recall, and F1-score) reflected the model's real-world performance.
- **Validation Considerations:** Although the standard split was 80-20, additional cross-validation techniques were also employed during experimentation to fine-tune hyperparameters. This iterative process allowed for robust model adjustments without compromising the integrity of the testing data.

4.1.4 Model Development

Instead of starting from scratch, the project leveraged Google Cloud's Vertex AI AutoML for model development. This decision was motivated by the need for a scalable, automated solution capable of managing complex image classification tasks. Key aspects of this phase included:

- **Automated Model Selection:** Vertex AI AutoML uses state-of-the-art Neural Architecture Search (NAS) to identify optimal model architectures based on the provided dataset. This automation greatly reduced the need for manual tuning and enabled rapid prototyping.
- **Transfer Learning:** The model utilized pre-trained networks that had already learned generalized image features. By fine-tuning these networks on our specific dataset of screw images, the project benefited from both efficiency and improved accuracy.
- **Hyperparameter Tuning and Optimization:** Vertex AI AutoML performed exhaustive hyperparameter tuning. This process included adjusting learning rates, batch sizes, and the number of training epochs to optimize model performance. Ensemble methods were also integrated to boost accuracy and reduce variance.
- **Scalability:** The use of Vertex AI ensured that the model could scale according to computational demands. The integration with cloud infrastructure allowed for efficient processing of large datasets and supported iterative training cycles.

4.1.5 Model Evaluation

The evaluation phase was crucial in determining the effectiveness of the defect detection system. The model was rigorously tested using several performance metrics:

- **Accuracy:** The overall percentage of correctly classified images served as the primary metric for model success.
- **Precision and Recall:** Precision measured the proportion of true positive defect detections against all predicted positives, while recall evaluated the model's ability to detect all actual defects. These metrics helped identify any bias in the model, particularly in avoiding false positives or negatives.
- **F1-Score:** As a harmonic mean of precision and recall, the F1-score provided a balanced view of the model's performance. High F1-scores were indicative of a robust model that maintained consistency across varying defect types and environmental conditions.
- **Confusion Matrix Analysis:** Detailed examination of the confusion matrix helped pinpoint specific classes where the model's performance could be improved. This analysis drove targeted refinements in preprocessing and model tuning.

4.1.6 Real-Time Analysis Setup

To transition from a controlled experimental setting to a production environment, a real-time analysis pipeline was developed. This involved integrating various hardware and software components to enable live defect detection:

- **Live Image Capture with DroidCam:** DroidCam was utilized to capture real-time images of screws during the manufacturing process.
- **Integration via PyCharm:** The captured images were processed using scripts developed in PyCharm and uploaded to a Cloud Storage bucket named "captured_images."
- **Triggering Cloud Functions:** Upon image upload, a Cloud Function was automatically triggered to retrieve, preprocess, and classify the image using Vertex AI AutoML.
- **Latency Considerations:** Optimization efforts ensured that the time between image capture and defect detection was within acceptable limits, thereby meeting the real-time performance requirements.

4.1.7 Cloud Integration and Performance Monitoring

- **Cloud Storage:** Centralized image repository for real-time analysis.
- **Cloud Functions:** Automated processing upon image upload.

- **Performance Monitoring:** Cloud Logging for tracking system health and optimization.

4.2 TESTING METHODOLOGY

Robust testing is essential to validate the functionality and efficiency of any automated system. In this project, a multi-tiered testing methodology was employed to assess both individual components and the overall system. The testing strategy encompassed the following elements:

4.2.1 Unit Testing

Unit testing was performed on each discrete component of the system to ensure that they operated as expected in isolation. Specific focus areas included:

Image Preprocessing Modules: Individual functions responsible for resizing, normalization, augmentation, and noise reduction were rigorously tested. The tests confirmed that each function produced the expected output for given inputs.

Cloud Function Execution: Standalone tests were conducted to verify that Cloud Functions correctly handled events triggered by image uploads. Simulated image files were used to check that these functions could execute without error and return the appropriate status messages.

Model Prediction Calls: The API calls to Vertex AI AutoML were tested to ensure that requests were correctly formatted and that the responses contained valid prediction scores.

By isolating each unit, the development team was able to quickly identify and resolve issues before integrating components into the full pipeline.

4.2.2 Integration Testing

Following successful unit tests, integration testing was carried out to verify that the various components could work together seamlessly. This involved:

End-to-End Pipeline Testing: The entire process—from image capture via DroidCam, through cloud upload, triggering of Cloud Functions, and finally receiving the prediction from the model—was tested under controlled conditions. This end-to-end test ensured that data flowed correctly between each system component.

Data Flow Verification: Detailed logging and monitoring were used to trace the flow of data, confirming that images were processed in the correct sequence and that the outputs from each stage matched expected results.

Inter-Component Communication: The integration tests also focused on ensuring that communication protocols (such as REST API calls and event triggers) functioned reliably even under variable network conditions.

4.2.3 Real-Time Testing

Real-time testing was crucial to validate that the system performed as expected in live operating conditions. This phase of testing included:

Live Environment Simulation: The system was deployed in an environment that closely mimicked real-world manufacturing settings. This included variable lighting conditions, different angles of image capture, and rapid consecutive image uploads.

Stress Testing: The system's capacity to handle a high volume of image uploads was evaluated. By simulating bursts of simultaneous image uploads, the tests ensured that Cloud Functions could scale dynamically and maintain performance without bottlenecks.

Latency Measurement: Response times were carefully monitored to ensure that predictions were delivered within the acceptable timeframe. This measurement was critical for confirming the system's real-time operational capability.

4.2.4 Performance Monitoring

Beyond initial testing, continuous performance monitoring was set up to ensure the system maintained its efficiency over time:

Cloud Logging: Every stage of the processing pipeline was integrated with Cloud Logging. This allowed for continuous collection of performance metrics, including processing times, error rates, and throughput.

Alert Systems: Automated alerts were configured to notify administrators if performance metrics deviated from expected values. This proactive approach enabled rapid response to any emerging issues.

Longitudinal Analysis: Over time, historical performance data was analyzed to identify trends and potential areas for further optimization. This analysis was instrumental in planning future upgrades and modifications to the system.

4.3 Test Cases

The final stage of the implementation was to verify the system through a series of structured test cases. Each test case was designed to probe different aspects of the pipeline—from handling of typical inputs to managing edge cases.

4.3.1 Cloud Function Triggering

Test Scenario: Verify that an image captured using DroidCam and uploaded to Cloud Storage triggers the Cloud Function.

Test Details:

A valid image file was captured and automatically uploaded to the “captured_images” bucket.

The Cloud Function was expected to detect the upload event, process the image, and forward it for prediction.

Outcome: The Cloud Function was successfully triggered, processed the image, and returned a prediction with a corresponding confidence score.

Evaluation: This test confirmed the correct configuration of event triggers and validated the end-to-end communication between Cloud Storage and Cloud Functions.

4.3.2 Handling Corrupted Image Files

Test Scenario: Evaluate system resilience by uploading a corrupted image file.

Test Details:

A deliberately corrupted file was introduced into the Cloud Storage bucket.

The Cloud Function’s error handling routines were monitored to determine if the system could gracefully manage the corrupted input.

Outcome: The error was detected early, logged appropriately, and the system continued to operate without interruption.

Evaluation: This case demonstrated the robustness of the error-handling mechanisms and the overall stability of the system under abnormal conditions.

4.3.3 Concurrent Image Uploads

Test Scenario: Assess the system’s ability to process multiple images simultaneously.

Test Details:

Multiple images were uploaded concurrently to simulate a high-load production environment.

The Cloud Functions were monitored to verify that each image was processed correctly, without any loss or misclassification.

Outcome: All images were successfully processed, and predictions were returned in parallel with minimal latency.

Evaluation: This test confirmed that the system’s architecture could handle concurrent operations and scale dynamically to meet increased demand.

4.3.4 Model Response Time

Test Scenario: Measure the response time of the prediction model.

Test Details:

A series of images were uploaded in rapid succession to gauge the time taken from image upload to prediction delivery.

Detailed latency logs were analyzed to determine if the system met real-time performance benchmarks.

Outcome: The system consistently returned predictions within the defined acceptable time limit, ensuring real-time operational performance.

Evaluation: This test was crucial to validate that the end-to-end pipeline met industrial real-time processing standards.

4.3.5 Classification Accuracy

Test Scenario: Validate the model’s classification accuracy using various screw images.

Test Details:

A diverse set of images, including various defect types and lighting conditions, were uploaded.

The model’s predictions were compared against manual annotations and ground truth labels.

Outcome: The model accurately classified the images with high confidence, correctly distinguishing between defective and non-defective screws.

Evaluation: The high accuracy and consistency across varied conditions confirmed that the training, augmentation, and model tuning steps were effective. Furthermore, any discrepancies observed during testing informed subsequent rounds of model refinement.

4.3 Test Cases

To ensure a robust defect detection system, several test cases were conducted:

4.3.1 Image Upload and Cloud Function Trigger:

- A test was conducted to check whether an image captured using DroidCam and uploaded to Cloud Storage triggered the Cloud Function.
- Expected Result: The function successfully processed the image and returned a confidence score.
- Actual Result: The system performed as expected, confirming the trigger functionality.

4.3.2 Handling corrupted image files:

- The system was tested by uploading a corrupted image file to observe its behavior.
- Expected Result: The error should be handled gracefully without causing a system crash.
- Actual Result: The system successfully logged an error message and prevented a crash.

4.3.3 Concurrent image processing

- Multiple images were uploaded simultaneously to verify if the Cloud Function processed all images correctly.
- Expected Result: The system should handle concurrent uploads efficiently.
- Actual Result: All images were processed without delays, confirming scalability.

4.3.4 Model response time evaluation

- A test was performed to measure the response time of the model.
- Expected Result: Predictions should be returned within an acceptable time limit to ensure real-time performance.
- Actual Result: The response time met the expected threshold, validating real-time functionality.

4.3.5 Classification accuracy validation

- Various screw images were uploaded to verify classification accuracy.
- Expected Result: The model should correctly classify defective and non-defective screws with high confidence.
- Actual Result: The model achieved high classification accuracy, confirming reliability.

CHAPTER 5:

CONCLUSION & ENHANCEMENT

5.1 Conclusion

This project successfully implemented an automated defect detection system that integrates advanced machine learning techniques with robust cloud-based infrastructure. Leveraging Google Cloud's Vertex AI AutoML, real-time image capture via DroidCam, and dynamic Cloud Functions, the system has established a scalable solution for quality control in industrial settings.

Key Accomplishments:

- **Seamless Integration:**
  The project achieved a robust, end-to-end pipeline by combining real-time image capture, automated image processing, and cloud-based predictive analytics. This seamless integration between hardware and cloud services has enabled an efficient quality control process.
- **Real-Time Performance:**
  By harnessing the capabilities of Cloud Logging and continuous monitoring, the system maintained a high level of responsiveness and stability. The ability to rapidly detect defects as they occur ensures that manufacturing lines can respond to issues immediately.
- **High Accuracy and Scalability:**
  Rigorous training, testing, and fine-tuning resulted in a defect detection model that delivers high classification accuracy. The scalable nature of cloud infrastructure ensures that the system can handle increased production volumes without compromising performance.
- **Proactive Maintenance and Monitoring:**
  The implementation of continuous performance monitoring through automated alerts and historical data analysis has led to proactive maintenance practices, reducing downtime and ensuring consistent system reliability.

5.2 Scope for Enhancements

While the current system forms a robust foundation, there remain several promising avenues for future enhancements. These improvements aim to further elevate the system's efficiency, accuracy, and adaptability in complex manufacturing environments.

5.2.1 Edge Computing Integration

- Deploying models on edge devices reduces latency and enables faster decision-making.
- Real-time processing is crucial for high-speed manufacturing.
- Minimizing cloud data transfer cuts costs and improves efficiency.

5.2.2 Model Optimization & Advanced Learning

- Expanding dataset diversity improves robustness.
- Hyperparameter tuning enhances classification accuracy.
- Active and reinforcement learning ensure continuous adaptation.
- Transfer learning from industry-specific datasets boosts defect detection.

5.2.3 Multi-Class Defect Detection

- Enables granular classification beyond binary defective/non-defective labels.
- Provides deeper insights into production quality.
- Supports automated sorting and intervention with robotics.

5.2.4 Enhanced Reporting & Analytics

- Real-time dashboards offer actionable insights.
- Predictive analytics forecast defect trends and maintenance needs.
- Automated reports streamline documentation and efficiency tracking.

5.2.5 Alternative Cloud & Hybrid Deployments

- Exploring AWS, Azure, or on-premise solutions for cost and performance benefits.
- Hybrid models balance performance, security, and cost-effectiveness.
- Ensuring data security and regulatory compliance.

5.2.6 Industrial System Integration

- IoT and sensor data provide a comprehensive production view.
- Predictive maintenance reduces downtime and optimizes efficiency.
- Robotic automation enables real-time corrective actions.

APPENDIX-1

CODING

PYTHON INTEGRATION WITH CAMERA

```
import cv2

import time

import os

# DroidCam URL (replace with your DroidCam IP and port)

droidcam_url = "http://192.168.43.1:4747/video"

# Open the video stream

cap = cv2.VideoCapture(droidcam_url)

if not cap.isOpened():

    print("Error: Could not open video stream.")

    exit()

# Interval in seconds (e.g., capture an image every 5 seconds)

interval = 5

# Folder to save images

save_folder = "captured_images"

if not os.path.exists(save_folder):

    os.makedirs(save_folder)

# Counter for image filenames

image_counter = 0

try:

    while True:

        # Capture frame-by-frame
        ret, frame = cap.read()
        if not ret:
            print("Error: Failed to capture image.")
            break# Convert the captured frame to grayscale
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
       # Save the grayscale frame as an image
        image_name = os.path.join(save_folder, f"image_{image_counter}.jpg")
        cv2.imwrite(image_name, gray_frame)
        print(f"Saved: {image_name}")
        # Increment the counter
        image_counter += 1
       # Wait for the specified interval
        time.sleep(interval)
      except KeyboardInterrupt:
          print("Image capture stopped by user.")
      finally:
          # Release the video stream and close all OpenCV windows
       cap.release()
       cv2.destroyAllWindows()
       print("Video stream closed.")
```

UPLOADING REAL TIME IMAGES TO GOOGLE CLOUD STORAGE:

```
import os
from google.cloud import storage
def upload_to_gcs(bucket_name, source_folder, destination_folder):
    client = storage.Client()
    bucket = client.bucket(bucket_name)
for image_name in os.listdir(source_folder):
        source_file_path = os.path.join(source_folder, image_name)
        destination_blob_name = f"{destination_folder}/{image_name}"
         blob = bucket.blob(destination_blob_name)
        blob.upload_from_filename(source_file_path)
        print(f"File {image_name} uploaded to {destination_blob_name}.")
# Replace with your bucket name and folder paths
bucket_name = "captured_images"
source_folder = "captured_images"
destination_folder = "images"  # Folder in the bucket
upload_to_gcs(bucket_name, source_folder, destination_folder)
```

CLOUD FUNCTION FOR PREDICTION USING VERTEX AI AUTOML MODEL:

```
import os
import base64
import logging
from google.cloud import aiplatform
from google.cloud import storage
# Set up logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def hello_gcs(event, context):
    """
    Triggered by a change to a Cloud Storage bucket.
```

```
Processes an image file, sends it to Vertex AI for prediction,
and logs the prediction results.
"""
bucket_name = event['bucket']
file_name = event['name']
logger.info(f"Processing file: {file_name} from bucket: {bucket_name}")

try:
    # Download the file from GCS
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(file_name)
    image_bytes = blob.download_as_bytes()
    logger.info(f"Downloaded image {file_name} successfully.")
    # Encode the image in base64
    encoded_image = base64.b64encode(image_bytes).decode("utf-8")
    # Prepare the instance payload for your model.
    instance = {"content": encoded_image}
    # Retrieve environment variables set in Cloud Function configuration
    project_id = os.environ.get("PROJECT_ID")
    region = os.environ.get("REGION")
    endpoint_id = os.environ.get("ENDPOINT_ID")
    # Initialize Vertex AI with your project and region
    aiplatform.init(project=project_id, location=region)
    # Connect to your deployed Vertex AI endpoint
    endpoint = aiplatform.Endpoint(endpoint_id)
```

```
# Call the prediction endpoint
response = endpoint.predict(instances=[instance])
# Log the prediction results
logger.info(f"Prediction results for {file_name}: {response.predictions}")
except Exception as e:
    logger.error(f"Error processing file {file_name}: {str(e)}")
    raise e  # Re-raise the exception to let Cloud Functions handle it
```

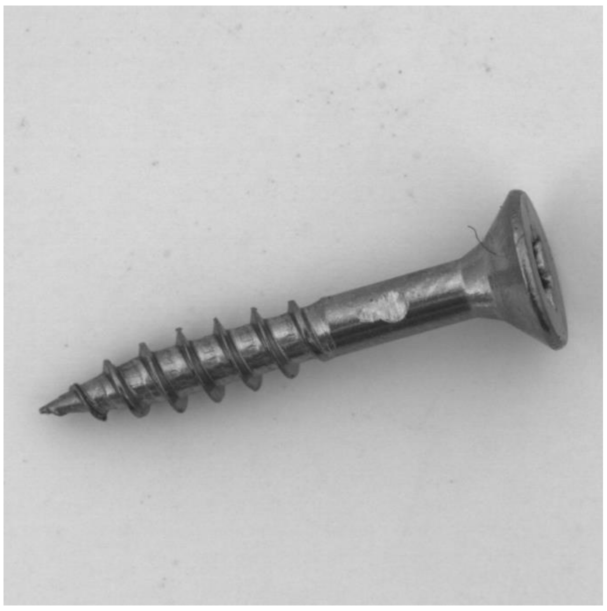
APPENDIX-2

SCREENSHOTS:

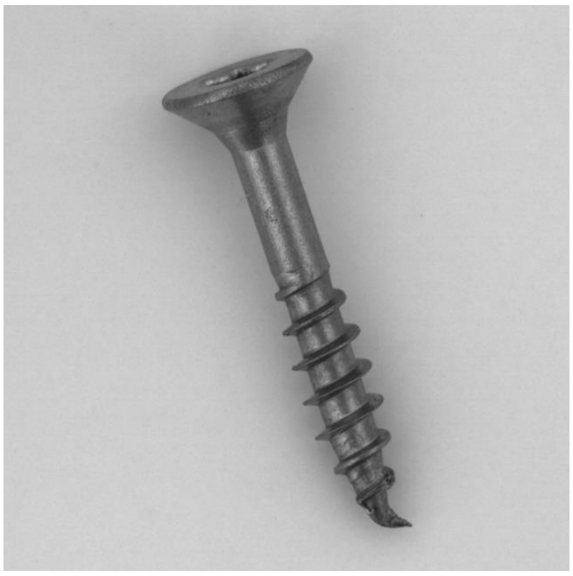
DATASET:

NON DEFECTIVE SCREWS

DEFECTIVE SCREW



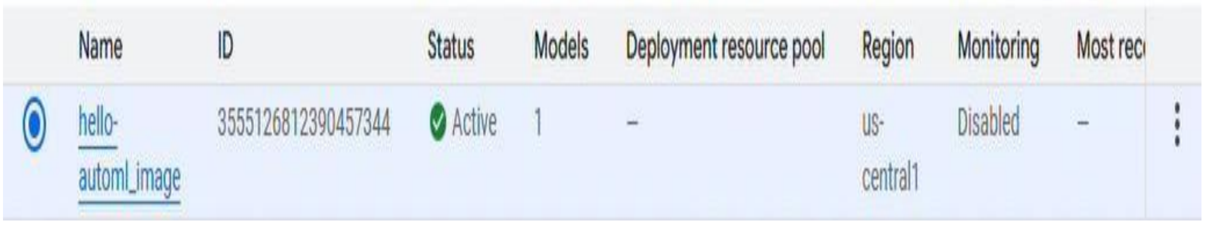
scratch neck



Manipulated front

Thread side

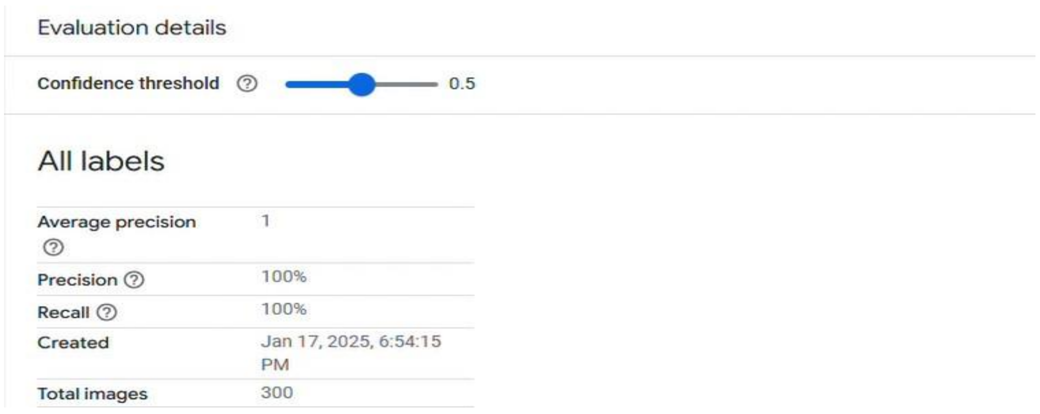
MODEL:



	Name	ID	Status	Models	Deployment resource pool	Region	Monitoring	Most rec
◉	hello-automl_image	3555126812390457344	✓ Active	1	–	us-central1	Disabled	–

Fig 3.9 AutoMLModel

EVALUATION PERFORMANCE:



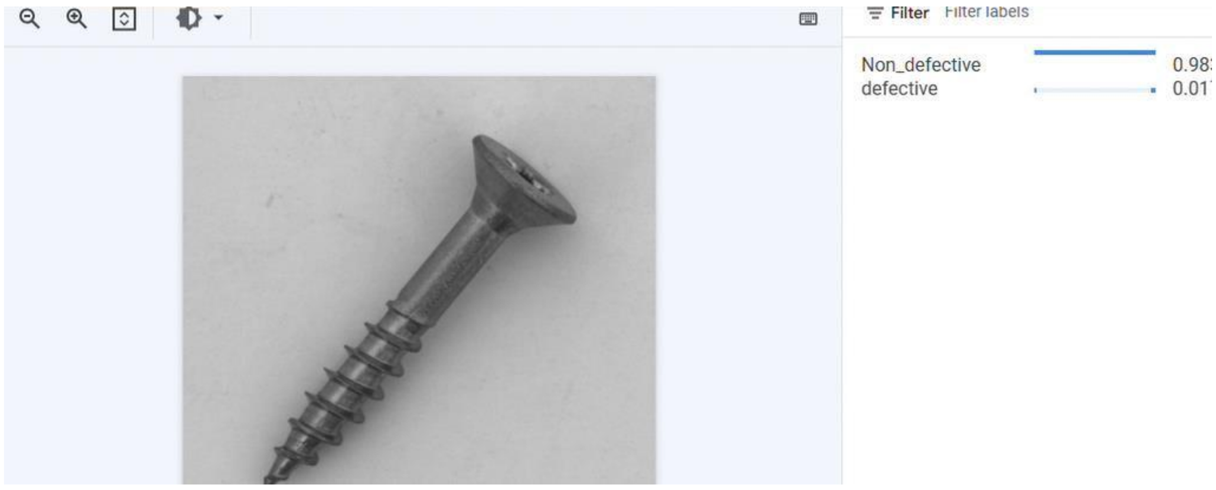
Evaluation details

Confidence threshold 0.5

All labels

Average precision	1
Precision	100%
Recall	100%
Created	Jan 17, 2025, 6:54:15 PM
Total images	300

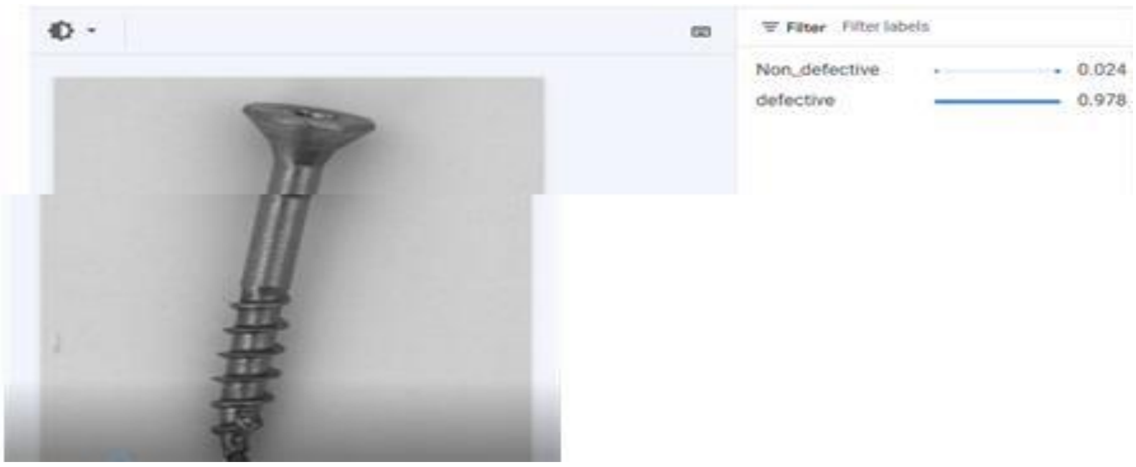
TESTING THE MODEL:



Filter Filter labels

Non_defective 0.98

defective 0.01



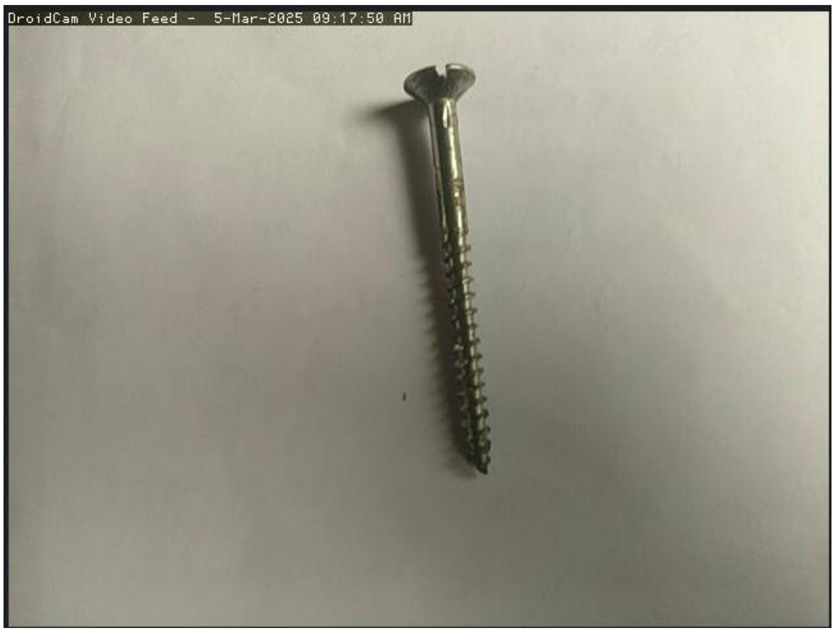
Filter Filter labels

Non_defective 0.024

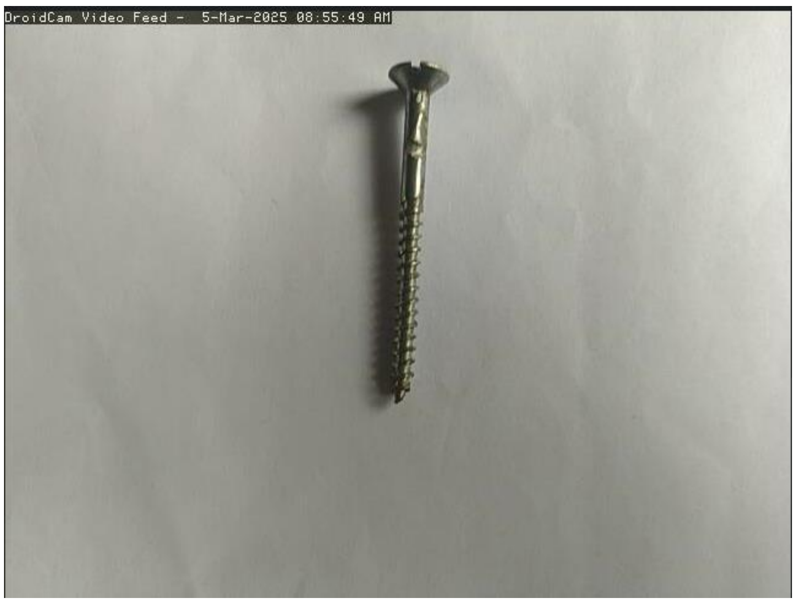
defective 0.978

REALTIME IMAGES:

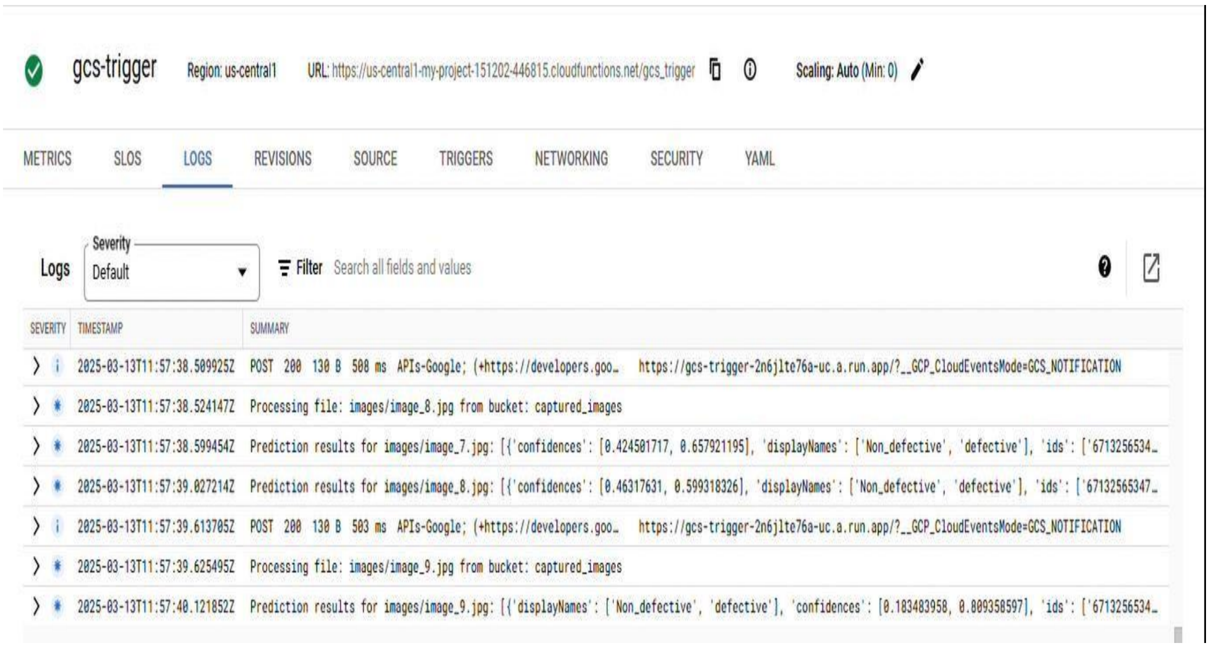
Non defective screw



Defective screw

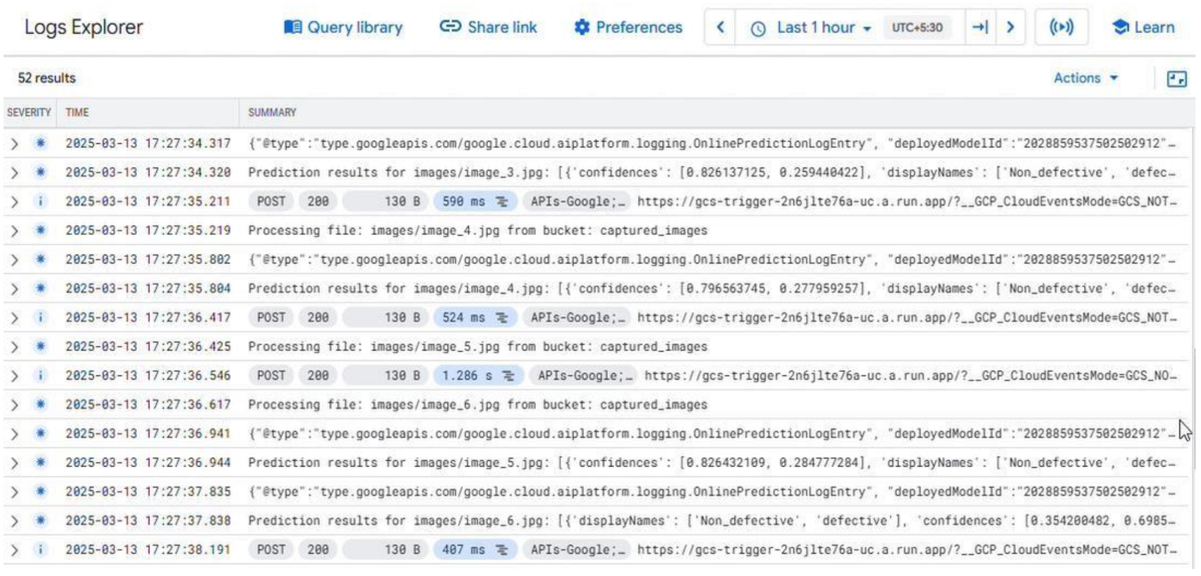


PREDICTION USING CLOUD FUNCTION:



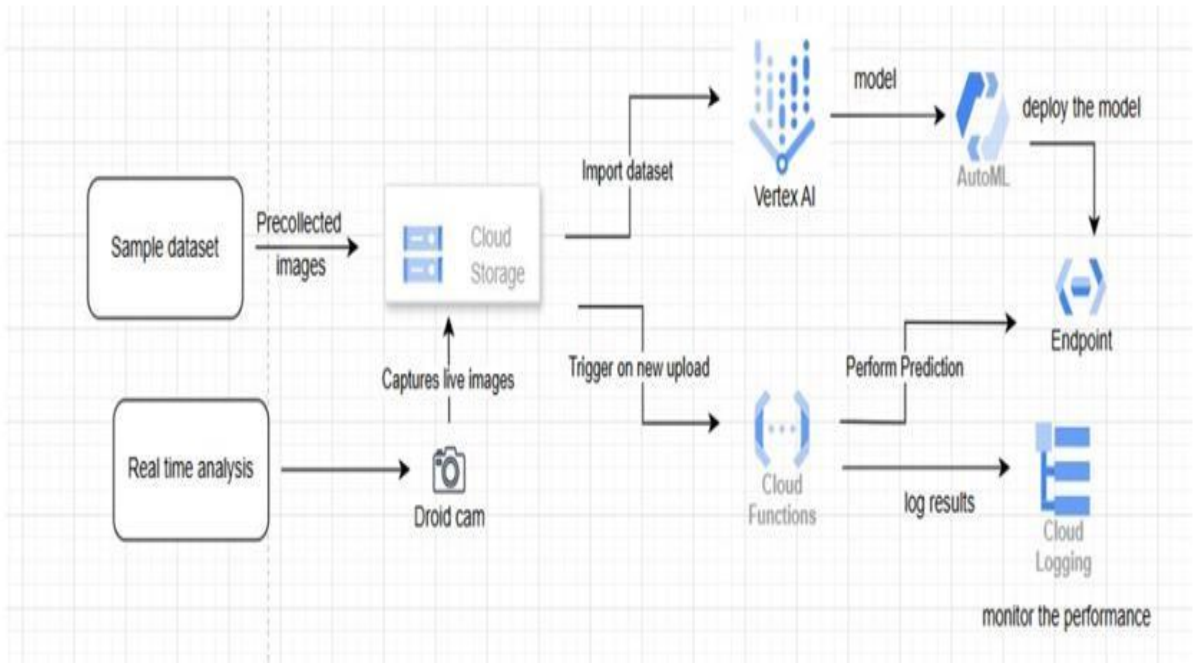
```
SEVERITY  TIMESTAMP                    SUMMARY
i  2025-03-13T11:57:38.509925Z  POST  200  130 B  508 ms  APIs-Google; (+https://developers.goo…   https://gcs-trigger-2n6jlte76a-uc.a.run.app/?__GCP_CloudEventsMode=GCS_NOTIFICATION
*  2025-03-13T11:57:38.524147Z  Processing file: images/image_8.jpg from bucket: captured_images
*  2025-03-13T11:57:38.599454Z  Prediction results for images/image_7.jpg: [{'confidences': [0.424501717, 0.657921195], 'displayNames': ['Non_defective', 'defective'], 'ids': ['6713256534…
*  2025-03-13T11:57:39.027214Z  Prediction results for images/image_8.jpg: [{'confidences': [0.46317631, 0.599318326], 'displayNames': ['Non_defective', 'defective'], 'ids': ['67132565347…
i  2025-03-13T11:57:39.613705Z  POST  200  130 B  503 ms  APIs-Google; (+https://developers.goo…   https://gcs-trigger-2n6jlte76a-uc.a.run.app/?__GCP_CloudEventsMode=GCS_NOTIFICATION
*  2025-03-13T11:57:39.625495Z  Processing file: images/image_9.jpg from bucket: captured_images
*  2025-03-13T11:57:40.121852Z  Prediction results for images/image_9.jpg: [{'displayNames': ['Non_defective', 'defective'], 'confidences': [0.183483958, 0.809358597], 'ids': ['6713256534…
```

MONITORING PERFORMANCE:



```
SEVERITY  TIME                     SUMMARY
*  2025-03-13 17:27:34.317  {"@type":"type.googleapis.com/google.cloud.aiplatform.logging.OnlinePredictionLogEntry", "deployedModelId":"2028859537502502912"…
*  2025-03-13 17:27:34.320  Prediction results for images/image_3.jpg: [{'confidences': [0.826137125, 0.259440422], 'displayNames': ['Non_defective', 'defec…
i  2025-03-13 17:27:35.211  POST  200  130 B  590 ms  APIs-Google;…  https://gcs-trigger-2n6jlte76a-uc.a.run.app/?__GCP_CloudEventsMode=GCS_NOT…
*  2025-03-13 17:27:35.219  Processing file: images/image_4.jpg from bucket: captured_images
*  2025-03-13 17:27:35.802  {"@type":"type.googleapis.com/google.cloud.aiplatform.logging.OnlinePredictionLogEntry", "deployedModelId":"2028859537502502912"…
*  2025-03-13 17:27:35.804  Prediction results for images/image_4.jpg: [{'confidences': [0.796563745, 0.277959257], 'displayNames': ['Non_defective', 'defec…
i  2025-03-13 17:27:36.417  POST  200  130 B  524 ms  APIs-Google;…  https://gcs-trigger-2n6jlte76a-uc.a.run.app/?__GCP_CloudEventsMode=GCS_NOT…
*  2025-03-13 17:27:36.425  Processing file: images/image_5.jpg from bucket: captured_images
i  2025-03-13 17:27:36.546  POST  200  130 B  1.286 s  APIs-Google;…  https://gcs-trigger-2n6jlte76a-uc.a.run.app/?__GCP_CloudEventsMode=GCS_NO…
*  2025-03-13 17:27:36.617  Processing file: images/image_6.jpg from bucket: captured_images
*  2025-03-13 17:27:36.941  {"@type":"type.googleapis.com/google.cloud.aiplatform.logging.OnlinePredictionLogEntry", "deployedModelId":"2028859537502502912"…
*  2025-03-13 17:27:36.944  Prediction results for images/image_5.jpg: [{'confidences': [0.826432109, 0.284777284], 'displayNames': ['Non_defective', 'defec…
*  2025-03-13 17:27:37.835  {"@type":"type.googleapis.com/google.cloud.aiplatform.logging.OnlinePredictionLogEntry", "deployedModelId":"2028859537502502912"…
*  2025-03-13 17:27:37.838  Prediction results for images/image_6.jpg: [{'displayNames': ['Non_defective', 'defective'], 'confidences': [0.354200482, 0.6985…
i  2025-03-13 17:27:38.191  POST  200  130 B  407 ms  APIs-Google;…  https://gcs-trigger-2n6jlte76a-uc.a.run.app/?__GCP_CloudEventsMode=GCS_NOT…
```

ARCHITECTURE:


Sample dataset
Precollected images
Cloud Storage
Import dataset
Vertex AI
model
AutoML
deploy the model
Endpoint
Captures live images
Trigger on new upload
Perform Prediction
Real time analysis
Droid cam
Cloud Functions
log results
Cloud Logging
monitor the performance