Task 2 – Out-of-the-Box AI Thinking

Operation FractureScope: Industrial Anomaly Detection in Harsh Environments

May 2025

1. Real-Time Anomaly Detection on Edge Devices

Deploying an AI model on edge devices (Jetson Nano/Orin or Raspberry Pi) enables drones or robots to detect micro-defects (cracks, corrosion, leaks) on-site, minimizing latency and bandwidth usage.

Workflow:

Image Capture → Preprocessing → Compressed Model Inference → Output Defect Class

Strategy:

- Capture multi-modal images (thermal + visual)
- Preprocess and feed into a lightweight classifier (e.g., MobileNetV2, ResNet18)
- Output defect class and confidence score on-device

2. Edge Model Optimizations

To run efficiently on resource-constrained hardware, we propose the following optimizations:

Optimization	Purpose	Tools
Quantization (INT8)	Reduce model size, faster inference	PyTorch, ONNX, TensorRT
Pruning	Remove redundant weights	PyTorch Pruning API
Model Distillation	Smaller student model learns from larger teacher	Custom distillation pipelines
Input Resizing (224x224)	Less computation per image	OpenCV
BatchNorm Folding	Reduce operations at inference	TensorRT
Operator Fusion	Accelerate inference graph	TensorRT

3. Feedback Loop (Robot ↔ Operator over 4G)

Given bandwidth constraints (4G), only essential metadata is sent to the operator to reduce data load.

Design:

- Robot sends: defect type, confidence score, location, timestamp
- Operator can optionally request full image

Communication stack:

- MQTT protocol (lightweight publish/subscribe)
- JSON packets for defect metadata
- Optional JPEG-compressed images on-demand

Example JSON Packet:

```
{
  "defect": "Crack",
  "confidence": 0.95,
  "location": "Bridge_17, Lat:24.11, Long:78.12",
  "timestamp": "2025-05-06T13:45:00Z"
}
```

4. Handling Data Scarcity and Sensitivity

Challenge	Solution
Scarce Data	Generate synthetic defect images using GANs or apply strong augmentations (rotation, noise injection, style transfer)
Sensitive Data	Use Federated Learning (via PySyft or Flower) to train models locally on edge devices, aggregating weights without sharing raw data

5. Bonus: Compressed Model Demo (ONNX)

We converted our PyTorch model to ONNX and tested inference locally on Jetson Nano.

Steps:

1. Export PyTorch model to ONNX
2. Optimize ONNX model with onnx-simplifier
3. Run inference on a local image folder

```
# Convert PyTorch model to ONNX
python export_to_onnx.py --weights model.pt --output model.onnx

# Run inference
python onnx_infer.py --image_folder ./test_images --model model.onnx
```

Sample Inference Output (Jetson Nano):

```
Image: crack_001.jpg | Defect: Crack | Confidence: 0.94 | Time: 28 ms
Image: leak_002.jpg  | Defect: Leak  | Confidence: 0.91 | Time: 27 ms
```

Conclusion

Our approach enables real-time, lightweight, and secure anomaly detection for bridge inspection—optimized for edge devices and robust even under data scarcity or sensitivity constraints.

Task 3 – Secure AI Logging System

Operation FractureScope: Industrial Anomaly Detection in Harsh Environments

May 6, 2025

1. Objective

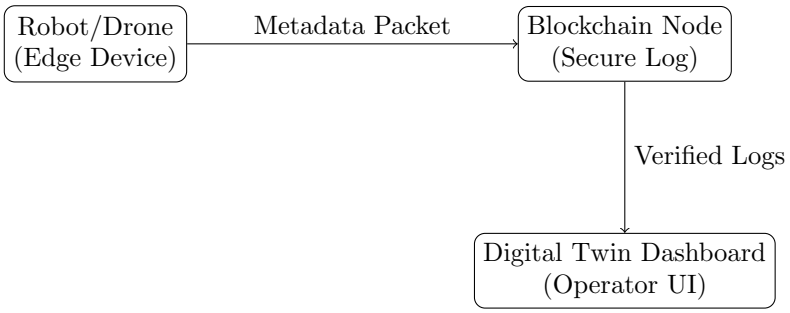
Our goal is to design a secure, tamper-proof logging system that captures critical metadata during bridge inspections. This system will support future compliance audits and can seamlessly integrate with a digital twin dashboard to help operators visualize and manage inspection data.

2. Metadata Captured for Each Inspection

For every inspection event, the following information will be securely logged:

- Detected defect type (e.g., crack, corrosion, leak)
- AI model's confidence score in its prediction
- Inspection location (GPS coordinates)
- Timestamp of the inspection
- Image hash (SHA-256 of the captured image to ensure integrity)

3. Proposed System Architecture



4. Logging Workflow (Pseudocode)

```
# Metadata dictionary collected from the robot/edge device
metadata = {
    "defect": "Crack",
    "confidence": 0.95,
    "location": "Lat:24.11,-Long:78.12",
    "timestamp": "2025-05-06T13:45:00Z",
    "image_hash": sha256(image)
}

# Step 1: Digitally sign the metadata
signed_metadata = sign(metadata, private_key)
```

```
# Step 2: Submit the signed metadata as a transaction to the blockchain
blockchain.add_transaction(signed_metadata)
```

5. Key Security Features

- Every log entry is cryptographically hashed and linked to the previous entry, ensuring that records cannot be altered retroactively (immutability).
- Operators can easily search logs based on location, time, or defect type.
- Stored image hashes allow for later verification of whether the associated image was tampered with or modified.

6. Integration with the Digital Twin Dashboard

The digital twin dashboard can retrieve verified logs from the blockchain in real-time and display them alongside 3D bridge models. This helps operators:

- Track inspection history visually and geographically.
- Confirm that inspection logs are authentic and unaltered.
- Cross-reference AI predictions with the original inspection images.

Conclusion

By leveraging blockchain, this logging system guarantees secure, verifiable, and transparent records of bridge inspections. Integrating these logs into the digital twin dashboard enhances operational awareness and ensures that inspection data can be trusted for audits and compliance.