Task 1: Multi-Modal AI Pipeline

Operation FractureScope – EdgeX AI Challenge

1 Objective

Design and implement a mini AI pipeline that:

- Ingests multi-modal images (visible, grayscale, thermal—simulated via augmentations).
- Trains a detector/classifier for at least three defect types (cracks, corrosion, leaks).
- Fuses thermal and visual information for improved performance.
- Employs custom preprocessing, augmentations, and justification of architectural and training choices.

2 Data Ingestion & Preprocessing

2.1 Synthetic Texture Generation

- Backgrounds synthesized by sampling Gaussian noise ($\mu = 128, \sigma = 18$) on a 512×512 canvas.
- Smoothed via a 7×7 Gaussian blur to mimic surface textures.

2.2 Defect Simulation

a) **Cracks (class 0):** Generated as branched polylines with 4–8 segments, thickness 1–2 px.

b) **Corrosion (class 1):** Multiple filled ellipses of random sizes and orientations, blended at $\alpha = 0.5$.

c) **Leaks (class 2):** Blurred circular masks with radius 20–40 px, applied via low-alpha overlay.

2.3 Multi-Modal Construction

From each RGB image:

- **Visible (RGB):** Raw synthetic image.
- **Grayscale:** Computed via `cv2.cvtColor(..., COLOR_BGR2GRAY)` and replicated to three channels.
- **Thermal:** Applied `cv2.applyColorMap(gray, COLORMAP_JET)`.

These three representations were *stacked* channel-wise to form a 9-channel input tensor.

3 Model Architecture

3.1 Base Detector: YOLOv8n

- Chosen for its balance of real-time inference speed and accuracy.
- Pretrained on COCO dataset; only detection head reinitialized for three custom classes.

3.2 Input Adaptation

- Modified first convolutional layer to accept 9 input channels instead of 3.
- Enabled transfer learning by freezing early backbone layers during initial epochs.

3.3 Fusion Rationale

Stacking channels allows the network to learn cross-modal correlations:

- Thermal colormap highlights interior intensity variations, aiding "leak" detection.
- Grayscale emphasizes shape and texture edges, improving "crack" localization.
- RGB retains color cues for "corrosion" blobs.

4 Training Procedure

4.1 Configuration

- **Dataset:** 1,200 synthetic images (1,000 train / 200 val), YOLOv8 folder structure.
- **Hyperparameters:**
  - Epochs: 50
  - Image size: 512×512
  - Batch size: 16
  - Optimizer: Adam (default settings)
- **Checkpointing:** Save every epoch; best model ('best.pt') backed up immediately.
- **Augmentations:** Mosaic, random horizontal flips, brightness/hue jitter native to YOLOv8.

4.2 Loss Function

YOLOv8 uses a composite loss:

$$\mathcal{L} = \lambda_{\text{box}}\mathcal{L}_{\text{CIoU}} + \lambda_{\text{cls}}\mathcal{L}_{\text{BCE}} + \lambda_{\text{dfl}}\mathcal{L}_{\text{DFL}}$$

where λ coefficients are set per YOLOv8 defaults. *Justification:*

- CIoU loss accounts for bounding-box overlap and aspect ratio.
- BCE classification loss handles multi-class detection.
- DFL (Distribution Focal Loss) improves localization precision on small defects.

5 Evaluation Metrics

5.1 Synthetic Validation

Metrics reported at end of training:

- mAP@0.5 : 0.745
- mAP@0.5-0.95 : 0.714
- Precision (all classes) : 0.725
- Recall (all classes) : 0.823

5.2 Hand-Labeled Real Data

A subset of 20 real images was annotated and evaluated via:

- Precision, Recall, mAP@0.5 (using YOLOv8's `val` method)

6 Task 2: Out-of-the-Box AI Thinking

Write a short technical note covering four key points: real-time anomaly detection on edge devices, model optimizations, 4G feedback loop design, and data scarcity solutions.

6.1 Edge Inference Architecture

To achieve real-time detection on resource-constrained hardware (e.g., Jetson Nano/Orin, Raspberry Pi), we deploy the compressed YOLOv8n model as an ONNX engine using ONNX Runtime or TensorRT:

- **Pre-processing:** Capture frame $\rightarrow$ resize to 512×512 $\rightarrow$ normalize $\rightarrow$ stack RGB, grayscale, thermal into a 9-channel tensor.
- **Inference engine:** ONNX Runtime (CPU) or TensorRT (GPU FP16/INT8), batch size=1, warm-up followed by per-frame inference.
- **Post-processing:** Non-max suppression (NMS), confidence thresholding (0.25), class mapping.
- **Benchmark (CPU only, 4 threads):** FP32 ONNX inference latency of 68.8/image (14.5 FPS).

6.2 Model Optimizations

We apply several techniques to shrink model size and accelerate inference:

- **Quantization:** Post-training dynamic INT8 quantization yields 4x smaller weights and potential 2–4× speedup on supported runtimes.
- **Pruning:** Structured filter/channel pruning (e.g., 20% channel removal) to reduce FLOPs, compatible with standard kernels.
- **Mixed Precision & Fusion:** FP16 inference on Nvidia devices; layer fusion (Conv+BN+Act) via TensorRT optimizes throughput.

6.3 4G Feedback Loop Design

Only essential metadata and thumbnails are transmitted over 4G:

- **Payload:** JSON packet with timestamp, GPS, image hash, detected classes, confidence, and normalized bboxes.
- **Thumbnail:** Compressed 64×64 JPEG for quick preview.
- **Transport:** MQTT (QoS=1) or HTTPS POST with retry/backoff and local caching on connectivity loss.
- **Dashboard Integration:** Web UI plots defects on a digital-twin map; users can request full-resolution images on demand.

6.4 Data Scarcity & Federated Learning

When real data is limited or sensitive, we leverage:

- **Synthetic Data Augmentation:** Domain randomization (textures, lighting, GAN style transfer) to enrich training diversity.
- **Federated Fine-Tuning:** On-device gradient updates on private samples, with secure server aggregation and optional differential privacy.

7 Task 3: Secure AI Logging System

7.1 Objective

Design a tamper-proof, blockchain-based logging system that captures and stores inspection metadata (location, image reference, predictions, timestamp) and integrates seamlessly into a digital-twin dashboard for audit and compliance.

7.2 Blockchain-Based Log Design

- **Immutable Chain:** Each log entry is a block containing `index`, `timestamp`, `prev_hash`, `data`, and `hash`.
- **Lightweight On-Chain Data:** Store only compact metadata and an off-chain image reference (IPFS CID or S3 key) to minimize on-chain storage.
- **Off-Chain Storage:** Large assets (full-resolution images) live in object storage (e.g. S3, IPFS, MinIO); on-chain records contain only `image_ref`.

7.3 Data Schema

Each block's `data` field follows:

```
{
  "location":  {"lat": float, "lon": float},
  "image_ref": "CID_OR_S3_KEY",
  "predictions": [
    {"class":"Crack","conf":0.92,"bbox":[x1,y1,x2,y2]},
    ...
  ]
}
```

The block header:

```
{
  "index":  int,
  "timestamp":"YYYY-MM-DDTHH:MM:SSZ",
```

```
  "prev_hash":"64-hex-chars",
  "data":{...},
  "hash":  "SHA256(hex of index+prev_hash+data)"
}
```

7.4 On-Device Logging Pseudocode

```python
import hashlib, json, time
# from storage_client import StorageClient
# from blockchain_client import ChainClient

def sha256_hex(b: bytes) -> str:
    return hashlib.sha256(b).hexdigest()

# Initialize clients (pseudo-code)
# storage = StorageClient(bucket="bridge-inspection")
# chain   = ChainClient(node_url)

prev_hash = chain.get_latest_hash() or "0"*64

def log_inspection(frame_bytes, preds, gps):
    # 1. Upload image off-chain     returns CID or key
    image_ref = storage.upload_bytes(frame_bytes)

    # 2. Assemble block content
    block = {
      "index":     chain.next_index(),
      "timestamp": time.strftime("%Y-%m-%dT%H:%M:%SZ", time.gmtime()
         ),
      "prev_hash": prev_hash,
      "data": {
        "location":     gps,
        "image_ref":    image_ref,
        "predictions": preds
      }
    }

    # 3. Compute and attach block hash
    h = sha256_hex(json.dumps(block, sort_keys=True).encode())
    block["hash"] = h

    # 4. Submit to blockchain
    chain.submit_transaction(json.dumps(block))

    # 5. Update prev_hash
    prev_hash = h
```

Listing 1: Create and submit a new block

7.5 Dashboard Integration

- **Backend:** Periodically query the chain for new blocks, store in a time-series or document database.
- **Frontend:** A digital-twin web UI (e.g., Three.js) that:
  - Renders the 3D bridge model.
  - Plots defect markers at GPS locations from each block.
  - On click, fetches the off-chain image (via `image_ref`) and displays predictions and timestamp.
- **Auditability:** Full chain history preserved for compliance reviews.