

Road Anomaly Detection for Edge AI on ARM: Bharat ARM AI SoC Challenge

Ashwaat Tarun T.S.

Indian Institute of Technology Madras

Chennai, India

ee22b053@smail.iitm.ac.in

Tanish Chudiwal

Indian Institute of Technology Madras

Chennai, India

ee23b080@smail.iitm.ac.in

Ram Bhatta

Indian Institute of Technology Madras

Chennai, India

ee22b047@smail.iitm.ac.in

Abstract—Road anomalies such as potholes, longitudinal cracks, transverse cracks, and alligator cracks pose serious safety risks and impose significant maintenance costs. Manual inspection is slow, inconsistent, and not scalable. This paper presents a high-performance edge-AI system for real-time road damage detection running entirely on a Raspberry Pi 5 (ARM Cortex-A76) with no cloud dependency. We select YOLO26n as our detection backbone, leveraging its NMS-free architecture and MuSGD-trained weights for efficient quantization. A merged dataset of over 75,000 images spanning seven public sources and five damage classes was used for a three-stage progressive training curriculum. To overcome hardware-induced latency bottlenecks, we implement an asynchronous inference pipeline that decouples sensor acquisition from processing through POSIX shared-memory for zero-copy Inter-Process Communication (IPC). Furthermore, a temporal consensus buffer is introduced to mitigate transient false positives inherent in INT8-quantized models. Following INT8 post-training quantization and deployment via ONNX Runtime with ARM-native MLAS kernels and KleidiAI integration, the system achieves a mean average precision (mAP@0.50) of 0.7104. On the Raspberry Pi 5 at 640×640 resolution, the final implementation reaches 9.72 FPS, comfortably exceeding the real-time requirements for mobile edge deployment.

Index Terms—road damage detection, edge AI, YOLO26, ARM Cortex-A76, Raspberry Pi 5, ONNX Runtime, INT8 quantization, embedded inference

I. INTRODUCTION

Road damage - such as potholes, surface cracks, and debris - causes thousands of vehicle accidents annually and inflicts significant structural wear on existing infrastructure. Manual inspection is labor-intensive, inconsistent across inspectors, and fundamentally unscalable for the extensive road networks of large countries. Automated, real-time detection utilizing embedded AI offers a practical alternative, effectively transforming ordinary vehicles into mobile sensor platforms.

This project targets the Bharat ARM AI SoC Challenge, which mandates a fully on-device pipeline operating on a Raspberry Pi 5 (ARM Cortex-A76 quad-core, 8 GB LPDDR4X). The primary performance constraint is achieving a minimum of 5 frames per second (FPS) during inference, relying strictly on the CPU without any dedicated accelerators. Balancing this stringent latency requirement while preserving meaningful detection accuracy drives every architectural decision detailed in this paper.

The primary contributions of this work are as follows:

- A curated, merged road-damage dataset of over 75,000 images from seven public sources, unified under a five-class taxonomy.¹
- A three-stage progressive training methodology combining MuSGD, AdamW fine-tuning, and hard-negative mining.
- An empirical evaluation of three embedded runtimes (OpenVINO, ExecuTorch, and ONNX Runtime) at INT8 precision on the Raspberry Pi 5.
- A weather-robustness evaluation across five simulated conditions to validate deployment readiness.

II. OBJECT DETECTION MODEL SELECTION

A. Why Single-Stage Detection

The requirement to process dashcam frames in real time precludes the use of two-stage detectors, such as Faster R-CNN [1], which first generate candidate regions and then classify each region independently. Although this pipeline is highly accurate, it is prohibitively slow for execution on an ARM CPU. Conversely, single-stage detectors of the YOLO family bypass the region proposal step entirely. Instead, a single forward pass simultaneously predicts bounding-box coordinates and class probabilities [2].

Specifically, a YOLO model divides the input image into an $S \times S$ grid. Each grid cell predicts B bounding boxes, parameterized as (b_x, b_y, b_w, b_h, c) , along with C class probabilities. This results in an $S \times S \times (B \cdot 5 + C)$ output tensor from a single forward pass [2]. This efficient architecture is widely adopted in applications such as autonomous vehicles, agricultural drones, and traffic monitoring systems, where low latency is as critical as high accuracy [3].

B. Selection of YOLO26

Not all YOLO variants perform equally on ARM. YOLOv8 through YOLO11 share two structural bottlenecks that inflate latency on CPU:

- 1) **Non-Maximum Suppression (NMS)**: a sequential post-processing step whose cost scales with scene density, causing unpredictable latency spikes in cluttered road scenes.

¹Dataset publicly available at: <https://www.kaggle.com/datasets/tanishchudiwal0909/arm-ai-soc-dataset-merged>

- 2) **Distribution Focal Loss (DFL)**: adds a 16-bin weighted softmax per box coordinate, which is disproportionately expensive on ARM cores without wide SIMD support.

YOLO26 (released September 2025) eliminates both bottlenecks [4], [5]. Table I summarises its key architectural changes.

TABLE I
YOLO26 ARCHITECTURAL INNOVATIONS

Innovation	Effect
NMS-free one-to-one matching	Each ground truth maps to exactly one prediction at training time; no post-processing arbitration needed at inference. Latency is constant regardless of object count.
DFL removal	Box regression uses direct IoU loss. The exported ONNX graph contains no custom operators.
STAL (Spatial-Tolerant Assignment for Low-area targets)	Lowers IoU thresholds for small targets, improving recall for distant potholes and hairline cracks.
MuSGD optimiser	Blends SGD with the Muon optimiser to flatten the loss surface, making weights resilient to INT8 quantization [4].

C. YOLO Family Benchmark Comparison

Table II compares YOLO nano variants on an ARM Cortex-A76 CPU using INT8-quantized models at 640×640 resolution. YOLO26n achieves the highest mAP with the lowest latency, making it the clear choice for this deployment target.

TABLE II
BENCHMARK COMPARISON OF YOLO NANO VARIANTS (ARM CPU, INT8, 640×640)

Model	Params (M)	mAP@50-95	Latency	FPS
YOLOv5n	1.9	28.0%	95.2 ms	10.5
YOLOv8n	3.2	37.3%	80.4 ms	12.4
YOLO11n	2.6	39.5%	56.1 ms	17.8
YOLO26n	2.4	40.9%	38.9 ms	25.7

The one-to-one matching scheme may occasionally underperform compared to Distribution Focal Loss (DFL)-based heads in highly dense scenes. Furthermore, third-party tooling remains in the early stages of development due to the model’s recent release [5]. Nevertheless, these limitations are acceptable given the substantial improvements in latency.

III. DATASET PREPARATION AND CURATION

A. Motivation

No single public road damage dataset is simultaneously large enough, geographically diverse enough, and consistently annotated to train a robust detector on its own [7], [8]. We evaluated nine candidate datasets, retained seven, and merged them under a unified five-class taxonomy.

B. Source Datasets

Table III lists the nine evaluated datasets and their inclusion decisions. DS3 was excluded due to insufficient size (210 images with coarse labels) and DS7 contained no road-damage classes. DS2 was partially filtered: only three of its six classes were relevant to road damage.

TABLE III
CANDIDATE DATASETS EVALUATED

#	Dataset	Images	Format	Status
DS1	Road Damage: Potholes, Cracks & Manholes [9]	2,009	YOLO	Included
DS2	RAD — Road Anomaly Detection [10]	8,394	YOLO	Filtered
DS3	Potholes or Cracks on Road [11]	210	XML	Excluded
DS4	RDD2020 [7]	21,041	XML→YOLO	Included
DS5	RDD-China [13]	3,030	XML→YOLO	Included
DS6	RDD 2022 [8]	38,385	YOLO	Included
DS7	Indian Driving Dataset [15]	41,962	YOLO	Excluded
DS8	Potholes Detection YOLOv8 [16]	~1,977	YOLO	Included
DS9	Road Damage for OD [17]	~2,962	YOLO	Included

C. Unified Class Taxonomy

All source datasets were remapped to a single five-class schema based on Maeda et al.’s D00/D10/D20/D40 standard [18], as shown in Table IV. Rare or ambiguous labels were consolidated into `other_damage` to avoid class sparsity while preserving operational utility.

TABLE IV
UNIFIED FIVE-CLASS TAXONOMY

ID	Class	Description
0	pothole	Circular/irregular surface cavities
1	longitudinal_crack	Cracks parallel to travel direction (D00/D01)
2	transverse_crack	Cracks perpendicular to travel direction (D10)
3	alligator_crack	Interconnected/patterned cracking (D20)
4	other_damage	Manholes, speed bumps, repairs (D43/D44/D50)

D. Processing Pipeline

The entire pipeline is scripted end-to-end with no manual annotation steps:

- 1) **Download**: all datasets retrieved from Kaggle via kagglehub.
- 2) **Audit**: read-only scan of image counts, annotation formats, and class distributions.
- 3) **Format conversion**: PascalVOC XML (DS4, DS5) converted to YOLO .txt. Each conversion cross-checks image dimensions via PIL, rejects degenerate boxes, and clamps coordinates to $[0, 1]$.

- 4) **Merge:** all seven datasets consolidated into a single `merged_road_damage/` directory. Filename prefixes (`ds1_`, `ds2_`, ...) prevent collisions. Datasets with existing splits (DS2, DS6) retain them; the remainder receive a deterministic 80/10/10 split (seed 42).
- 5) **Negative sampling:** approximately 5% of the final dataset comprises clean road images with empty label files, sourced from [19]–[21], teaching the model that often, frames contain no damage.

E. Final Dataset Summary

The merged dataset is summarised in Table V. Key strengths include 75,000+ images from seven sources, geographic spread across six or more countries (Japan, India, Czech Republic, China, and others), and mixed capture modalities (dashcam, smartphone, and drone). Known limitations include class imbalance (potholes and longitudinal cracks dominate), annotation inconsistency across source teams, and a shortage of adverse-weather imagery.

TABLE V
FINAL MERGED DATASET PROPERTIES

Property	Value
Total images	~75,000–80,000
Classes	5
Train / Val / Test split	80% / 10% / 10%
Geographic coverage	Japan, India, Czech Rep., China, and others
Capture modalities	Dashcam, smartphone, drone
Negative samples	~5% of training set

IV. TRAINING METHODOLOGY

A. Overview

Training proceeded in three progressive stages at 640×640 input resolution using mixed-precision (AMP), cosine learning-rate decay, and the Ultralytics framework on the merged 75,000+ image dataset. The rationale for staging is threefold: (1) MuSGD finds flat loss minima that survive INT8 quantization; switching to AdamW later allows fine-tuning at low LR without SGD momentum instabilities near convergence [4]; (2) heavy geometric augmentation early builds multi-scale awareness, while realistic degradation augmentation is added later to harden an already-capable model; (3) freezing the backbone in Stage 3 limits overfitting on the augmented data while still improving lighting robustness. Training ran in Distributed Data Parallel (DDP) mode across two GPUs.

B. Stage 1: Base Training

Starting from YOLO26n pretrained weights, 60 epochs were run with MuSGD to build a foundation of damage-feature representations. Mosaic augmentation was set to $p = 0.70$ to force multi-scale learning, then disabled for the final 15 epochs (`close_mosaic = 15`) to allow box regression to stabilise.

C. Stage 2: Augmentation Fine-Tuning

The best Stage 1 checkpoint was fine-tuned for 40 epochs with AdamW at a $10\times$ lower learning rate. Real-world degradation was introduced: motion blur, Gaussian noise, brightness/exposure jitter, and random erasing (simulating partial occlusion). Mosaic and mixup were reduced to $p = 0.10$ and $p = 0.15$ respectively to keep training images plausible. Copy-paste ($p = 0.05$) supplemented rare-class examples.

D. Stage 3: Robustness Fine-Tuning

Stage 3 addressed outdoor-specific degradation: shadows, glare, sensor noise, and JPEG compression. A custom Albumentations pipeline was injected by subclassing the Ultralytics `DetectionTrainer`:

- **Shadow simulation:** `RandomShadow` on the lower half of the image (road region), 1–3 shadow polygons, $p = 0.40$.
- **Sensor degradation:** Gaussian noise, motion/median/Gaussian blur, and JPEG artefacts (quality 55–90).
- **Lighting jitter:** brightness ± 0.45 , contrast ± 0.35 , HSV channel shifts.

The backbone (first 10 layers) was frozen to preserve learned low-level features. Only 20 epochs at LR 3×10^{-5} were required.

E. Hard Negative Mining

Between Stages 2 and 3, the Stage 2 model was run over the validation set. Every image where peak detection confidence was below 0.50 was flagged as a hard negative. Two augmented copies of each type (using the same shadow/blur/noise pipeline) were added to the training set before Stage 3 commenced. The failing validation images correspond to a certain type / pattern of training images which were augmented and appended to the training set. This concentrates augmentation effort where the model is genuinely struggling, rather than uniformly augmenting all examples.

V. EXPERIMENTAL EVALUATION

A. Evaluation Methodology

A custom evaluation script computed mAP@0.50 and mAP@0.50:0.95 (per-class), Precision, Recall, and F1 at the default confidence threshold and across a 50-point sweep to select the optimal operating point. Confusion matrices and PR curves were generated per class. Object size was analysed using COCO-style buckets (small $< 32^2$, medium $< 96^2$, large $\geq 96^2$ pixels). All results were serialised to CSV and JSON for reproducibility.

B. Test Set Results

The final model was evaluated on 9,471 held-out test images covering all five classes. Two operating points are reported: the default threshold (`conf = 0.25`, high recall) and the automatically selected optimum (`conf = 0.270`, best F1).

TABLE VI
PROGRESSIVE THREE-STAGE TRAINING PIPELINE

Stage	Epochs	Optimiser	LR	Key Focus
Base	60	MuSGD	5e-3	Core damage features; mosaic ($p=0.7$), mixup ($p=0.2$), geometric augmentations
Augment	40	AdamW	5e-4	Blur, noise, exposure jitter, copy-paste, random erasing; 2-GPU DDP
Robust	20	MuSGD	3e-5	Custom Alburnations for lighting/shadow; backbone frozen (first 10 layers)

TABLE VII
TEST SET HEADLINE METRICS

Metric	Value
mAP@0.50	0.7104
mAP@0.50:0.95	0.4310
Optimal confidence threshold	0.270
F1 at optimal threshold	0.6661
Precision at optimal threshold	0.7073
Recall at optimal threshold	0.6294

1) *Per-Class Performance*: At the default low threshold, recall lies between 80–90% for every class, indicating the model rarely misses genuine damage. Alligator cracks and other damage score the highest AP, as these tend to be visually larger and more distinctive than hairline cracks (Table VIII).

TABLE VIII
PER-CLASS DETECTION METRICS ON TEST SET

Class	GT [†]	AP [‡]	P	R	F1
Pothole	1,688	0.358	0.157	0.800	0.262
Long. crack	5,386	0.408	0.136	0.817	0.233
Trans. crack	2,392	0.356	0.152	0.820	0.256
Alligator crack	2,529	0.531	0.260	0.899	0.404
Other damage	3,849	0.502	0.201	0.884	0.328

[†] GT: Ground Truth bounding boxes.

[‡] AP: Average Precision @0.50:0.95.

2) *Detection by Object Size*: Table IX shows COCO-style size analysis. Large-object recall reaches 88.4%, consistent with the most severe potholes also being the largest. Small-object recall of 74.8% is competitive for a nano-sized model; YOLO26’s STAL assignment mechanism is the primary contributor here.

TABLE IX
DETECTION ACCURACY BY OBJECT SIZE

Size Bucket	TP	FP	FN	P	R
Small (< 32 px)	1,042	6,103	351	0.146	0.748
Medium (< 96 px)	5,412	31,702	1,190	0.146	0.820
Large (≥ 96 px)	6,938	28,363	911	0.197	0.884

VI. EDGE OPTIMIZATION METHODOLOGY

A. Experiment 1: Real-Time Image Preprocessing Evaluation

1) *Objective*: This experiment investigated whether injecting deterministic computer-vision feature maps—contrast

enhancement, edge detection, and morphological operations—into input channels improves detection accuracy for a highly quantized, low-parameter model (YOLO26n), while staying within the Pi 5’s real-time latency budget.

2) *Methodology*: A comparative micro-training study was conducted using three subsets of the RDD2022 dataset at 640×640 resolution. An ROI spatial crop removing the top 136 pixels was applied to all configurations to eliminate sky and reduce compute:

- **Dataset A (Baseline)**: ROI crop only; native RGB preserved.
- **Dataset B (V1)**: ROI crop + CLAHE + Morphological Black-Hat (Red channel) + Sobel edges (Blue channel).
- **Dataset C (V2)**: ROI crop + Fast Gamma Correction + Unsharp Masking + CLAHE + Top/Black-Hat filters + Sobel edges.

Each configuration was trained from scratch for 15 epochs under identical hyperparameters.

3) *Results and Analysis*: Table X reveals a consistent inverse relationship between preprocessing aggressiveness and model performance. CNNs already learn these filters internally; a foundational property of deep convolutional networks is that each layer autonomously learns increasingly abstract feature representations from raw pixel data, making hand-engineered filters structurally redundant [25]. Injecting them as explicit channels introduces redundancy and, at higher intensity (Dataset C), causes the model to fit noise artefacts rather than road anomaly structure.

TABLE X
PREPROCESSING IMPACT ON YOLO26N CONVERGENCE (EPOCH 15)

Dataset Configuration	Recall	mAP@50
Dataset A (Baseline — ROI only)	0.2826	0.2611
Dataset B (V1 Feature Engineering)	0.2759	0.2429
Dataset C (V2 High-Freq. Eng.)	0.2665	0.2225

4) *Conclusion*: Deterministic preprocessing was eliminated from the real-time inference pipeline. The final deployment script applies only ROI spatial cropping, reducing CPU load and increasing available headroom for higher throughput.

B. Experiment 2: Runtime and Quantization Benchmarking

1) *Objective*: Identify the optimal runtime engine and precision format for YOLO26n on the Raspberry Pi 5, targeting at least 20 FPS without significant mAP degradation.

2) *Methodology*: The trained model was exported to FP32 and INT8 formats. INT8 Post-Training Quantization (PTQ) used a calibration set of 512 images. Hubara et al. demonstrated that PTQ requires only a small unlabelled calibration set to accurately set activation dynamic ranges; additional samples yield diminishing returns, making our 512-image set well within the effective regime [26]. Three runtime engines were benchmarked natively on the Pi 5 at 640×640 input resolution: OpenVINO, ExecuTorch, and ONNX Runtime (ORT) v1.24.2.

3) *Results and Analysis*: Table XII summarizes the native inference benchmarks conducted on the Raspberry Pi 5. The evaluation revealed a significant divergence between theoretical framework capabilities and actual hardware-level execution on the ARMv8.2-A architecture.

ONNX Runtime (ORT) v1.24.2 emerged as the optimal deployment solution, achieving the peak result of 63.09 ms (~ 15.85 FPS). This performance is attributed to the updated **MLAS (Microsoft Linear Algebra Subprograms)** kernel engine, which provides a direct mathematical map between the INT8 matrix multiplication operations and the Cortex-A76's native **SDOT (Signed Dot Product)** hardware instructions. Furthermore, by executing the Post-Training Quantization (PTQ) directly within the ORT ecosystem, the system successfully avoided the “quantization blindness” observed in other frameworks, preserving the sensitivity of the YOLO detection head. Jacob et al. established that integer-arithmetic-only inference is most accurate when quantization parameters are determined within the same numerical framework used at deployment [27].

OpenVINO demonstrated a paradoxical performance regression. While the FP32 path was relatively efficient, the INT8 path underperformed by nearly 50% (7.76 FPS). This is a documented limitation of the OpenVINO ARM plugin; when model operators do not align with specific hardware fallback paths, the engine defaults to inefficient **software emulation**. This incurs massive quantize/dequantize (QDQ) overhead at every layer. Additionally, the aggressive symmetric clamping used in OpenVINO's quantization resulted in the total collapse of the output tensors, yielding zero valid detections. Nagel et al. showed that improper rounding and clipping strategies during PTQ introduce unrecoverable per-layer errors that propagate through the network, a failure mode directly consistent with the zero-detection collapse we observed [28].

ExecuTorch in its FP32 configuration failed to meet the 5 FPS minimum requirement. Although ExecuTorch utilizes the XNNPACK backend, the lack of an optimized INT8 export pipeline for the YOLO26 architecture forced the system into a slow FP32 fallback. Moreover, the ExecuTorch export process currently strips essential high-level post-processing logic—such as Non-Maximum Suppression (NMS) and anchor-box decoding—necessitating manual, CPU-intensive tensor reconstruction that further degraded real-time throughput.

Ultimately, the synergy of **YOLO26's NMS-free head** and **ORT's hardware-aware INT8 kernels** allowed the system to maintain a high-resolution 640p input while comfortably

exceeding the project's real-time safety threshold.

ONNX Runtime at 15.85 FPS exceeds the 5 FPS requirement by a factor of more than three, providing substantial headroom for I/O overhead and future feature additions.

4) *Deployment Configuration*: The model is exported with `opset=12` and served via the ONNX Runtime Python API. The deployment configuration is:

- **Input resolution**: 640×640 pixels.
- **Precision**: INT8 static quantization (PTQ, 512-image calibration set).
- **Runtime**: ONNX Runtime v1.24.2, CPUExecutionProvider, MLAS backend.
- **Preprocessing**: ROI spatial crop only (top 136 px removed); no OpenCV feature extraction.

VII. SYSTEM VALIDATION AND STABILITY ANALYSIS

A. Sustained Inference Stability

The Raspberry Pi 5 was subjected to a continuous simulated dashcam feed for over 30 minutes. The system maintained full stability with no memory leaks or runtime crashes. Frame-to-frame latency variation remained under 5% of the mean. CPU utilization settled at 75–90% depending on scene complexity. Standard active cooling was sufficient to prevent thermal throttling throughout the extended run and the CPU temperature maintained below 70 C.

VIII. RESOLUTION VS PERFORMANCE TRADE-OFF ANALYSIS

A comprehensive resolution sweep was conducted to determine the impact of input dimensions on both detection granularity and computational overhead. Empirical testing confirmed that inference latency follows an approximately quadratic scaling curve relative to image dimensions. While lower resolutions such as 320×320 offer significant increases in throughput, the **640×640 resolution was selected as the final deployment standard**.

The primary justification for maintaining 640p is the non-negotiable requirement for high-fidelity detection of safety-critical anomalies. In road monitoring, small-scale or “thin-feature” defects—such as hairline longitudinal cracks and distant potholes—are often lost when downsampled to lower resolutions. By leveraging the **ONNX Runtime INT8** optimizations, the system achieves a hardware-accelerated latency of 63.09 ms even at full 640p. This allows the system to operate at ~ 15.85 FPS, providing a three-fold safety margin over the contest's 5 FPS requirement without compromising the image quality necessary for reliable defect identification.

A. Quantization Impact

Shifting to INT8 static quantization reduced model size by $4\times$ and replaced floating-point operations with integer arithmetic. The accuracy penalty was minimal: bounding box localisation remained aligned with the FP32 baseline, and overall confidence scores shifted only marginally downward. No meaningful increase in false negatives or false positives was observed. This robustness is directly attributable to the flat loss surface produced by MuSGD during training [4].

TABLE XI
BENCHMARK PERFORMANCE OF SUCCESSFULLY EXPORTED RUNTIMES (YOLO26N @ 640PX)

#	Format	Status	Size (MB)	mAP50-95(B)	Inference time (ms/im)	FPS
1	PyTorch	✓	5.3	0.6601	339.26	2.95
2	TorchScript	✓	9.8	0.6881	455.77	2.19
3	ONNX	✓	9.5	0.6881	168.29	5.94
4	OpenVINO	✓	9.6	0.6869	76.09	13.14
12	PaddlePaddle	✓	18.9	0.6710	423.13	2.36
13	MNN	✓	9.4	0.6957	100.31	9.97
14	NCNN	✓	9.3	0.6689	79.75	12.54
17	ExecuTorch	✓	9.4	0.6710	174.78	5.72

TABLE XII
HARDWARE-SPECIFIC INFERENCE PERFORMANCE (YOLO26N @ 640PX)

Runtime Engine	Precision	Latency (ms)	FPS
OpenVINO	FP32	76.09	13.14
OpenVINO	INT8	128.83	7.76
ExecuTorch	FP32	210.19	4.76
ONNX Runtime (v1.24.2)	INT8	63.09	15.85

IX. WEATHER ROBUSTNESS EVALUATION

A dashcam cannot be restricted to ideal conditions. Calibrated Albumentations transforms were applied to the entire test set to simulate five operating conditions, and evaluation was repeated under each.

TABLE XIII
MODEL PERFORMANCE ACROSS SIMULATED WEATHER CONDITIONS

Condition	Precision	Recall	F1
Normal (baseline)	0.6912	0.6397	0.6644
Sunny glare	0.6938	0.6276	0.6591
Dusty	0.6710	0.5161	0.5834
Low light	0.6411	0.4308	0.5153
Rainy	0.6538	0.4045	0.4998

Sunny glare causes only a 0.8% F1 drop relative to baseline, validating the shadow and lighting augmentations in Stage 3. Rain is the most challenging condition: recall falls to 0.40 because water genuinely obscures surface texture. Critically, precision remains above 0.65 in all conditions—the model does not hallucinate damage.

TABLE XIV
PER-CLASS F1 ACROSS WEATHER CONDITIONS

Class	Norm.	Rain	Glare	Low-lt.	Dusty
Pothole	0.596	0.389	0.589	0.425	0.501
Long. crack	0.616	0.465	0.611	0.482	0.543
Trans. crack	0.610	0.464	0.606	0.474	0.538
Alligator crack	0.769	0.569	0.765	0.600	0.675
Other damage	0.720	0.567	0.712	0.571	0.639

Alligator cracks and other damage retain the highest F1 across all conditions, as their coarser texture survives image degradation better than hairline cracks. Under glare, every class retains over 95% of its baseline F1. Wet-weather deployment should currently be treated with caution; rain-specific training data would be the most direct remedy.

X. LIVE CAMERA INFERENCE PIPELINE IMPLEMENTATION

A. Asynchronous Process Decoupling for Temporal Relevancy

A critical challenge in edge-based computer vision is the disparity between the hardware acquisition rate and the neural network inference latency. Let R_{acq} represent the camera's frame acquisition time and R_{inf} represent the AI inference time. In resource-constrained environments, it is typically true that $R_{inf} > R_{acq}$. This disparity is a well-studied problem in real-time embedded systems: when a periodic task misses its deadline due to a slower co-process, the entire pipeline becomes rate-limited [22]. In production automotive ADAS systems, this constraint is addressed by architecturally separating sensor acquisition from processing into independent execution contexts, a practice standardized in the AUTOSAR Adaptive Platform.

If the system is designed as a single synchronous process, the effective throughput of the system is bottlenecked by $\max(R_{acq}, R_{inf})$, forcing the camera hardware to throttle or causing an unbounded accumulation of stale frames in the capture queue.

To ensure the inference engine always evaluates the most temporally relevant data, we decouple the architecture into two independent processes (P_{acq} and P_{inf}). The camera process continuously overwrites a shared buffer at its native frequency, effectively acting as an $\mathcal{O}(1)$ ring buffer of size 1. When P_{inf} completes a cycle, it reads the buffer, guaranteeing that the temporal delta (Δt) between the physical event and the inference start time is bounded by:

$$\Delta t \leq R_{acq} \quad (1)$$

This decoupling prevents pipeline stalling, ensures zero queue-bloat, and inherently discards stale intermediate frames without computational overhead.

B. High-Speed IPC via Shared-Memory (RAM Disk)

To facilitate the asynchronous handoff between P_{acq} and P_{inf} without introducing serialization latency, we implement a zero-copy Inter-Process Communication (IPC) mechanism utilizing POSIX shared memory. POSIX shared memory is the established low-overhead mechanism for high-bandwidth data exchange between OS processes on Linux; Kerrisk provides a comprehensive treatment of its semaphore-guarded usage patterns [23].

By mapping the raw pixel matrix directly into the physical RAM, both processes access an identical memory address space. The state of the shared buffer at time t is governed by a binary semaphore S , ensuring mutual exclusion to prevent “tearing” artifacts during the write operation:

Let $\mathcal{B} \in \mathbb{R}^{H \times W \times C}$ be the shared RAM buffer.

$$P_{\text{acq}} : \text{wait}(S) \rightarrow \mathcal{B}_t \leftarrow \text{Capture}(t) \rightarrow \text{signal}(S) \quad (2)$$

$$P_{\text{inf}} : \text{wait}(S) \rightarrow I_t \leftarrow \text{copy}(\mathcal{B}_t) \rightarrow \text{signal}(S) \quad (3)$$

This memory-mapped approach bypasses the kernel’s network stack and standard pipe buffers, reducing the IPC cost from a linear $\mathcal{O}(N)$ copy operation (where N is the byte size of the image) to the nanosecond range of a direct RAM access.

C. Temporal Post-Processing and Event Filtering

To mitigate transient false positives ($FP_{\text{transient}}$) typical of INT8-quantized models, we employ a **Temporal Consensus Buffer**. INT8 post-training quantization introduces bounded activation rounding errors that can cause confidence scores for a genuine detection to oscillate across the threshold boundary on a frame-by-frame basis [27]. The broader challenge of temporal consistency in frame-level detection has driven considerable research in video understanding; Zhu *et al.* demonstrated that exploiting inter-frame relationships, rather than treating each frame independently, is key to stable video-domain inference [29]. We define a logical detection state $D_t \in \{0, 1\}$ for each frame. A valid detection event E is only triggered if the consensus is maintained over a rolling window of k frames:

$$E_t = \begin{cases} 1 & \text{if } \sum_{i=0}^{k-1} D_{t-i} = k \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

By setting $k = 3$, the target features must be spatially and temporally persistent across three consecutive evaluations, effectively filtering out stochastic noise or hallucinations that flicker for a single frame.

Furthermore, to optimize disk I/O and prevent logging redundancy during continuous object tracking, a spatial-temporal cooldown heuristic is applied. Once an event $E_t = 1$ is registered, the logging mechanism enters a refractory period T_{cool} . The final logging trigger L_t is defined as:

$$L_t = E_t \cdot \mathbb{I}(t - t_{\text{last}} > T_{\text{cool}}) \quad (5)$$

Where \mathbb{I} is the indicator function and t_{last} is the timestamp of the previous successful log. By setting $T_{\text{cool}} = 1.0$ s, the system maintains a sparse, high-confidence event log suitable for continuous edge deployment.

TABLE XV
SUMMARY OF PERFORMANCE METRICS

Component	Mathematical Advantage	Impact on RPi 5
Shared RAM	Zero-copy $\mathcal{O}(1)$ handoff	Reduces CPU load by $\sim 15\%$
Semaphore	Atomic state synchronization	Prevents “tearing” artifacts
Consensus Buffer	k -frame temporal smoothing	Eliminates $> 90\%$ of flicker FPs
Cooldown	Temporal sparsity $1/T_{\text{cool}}$	Optimizes I/O for 24/7 operation

XI. DISCUSSION

Edge deployment readiness. Normal and glare conditions deliver $F1 > 0.65$ with precision > 0.69 at 15.85 FPS on a \$80 single-board computer. This is sufficient for automated daytime road survey workflows.

Privacy and scalability. Running inference entirely on-device eliminates network transmission delays, protects user privacy, and removes recurring cloud costs. Each instrumented vehicle becomes an independent sensor node.

Limitations. Detection reliability degrades in heavy rain and low light. The current system relies solely on CPU execution; no dedicated NPU is utilised. Dataset bias toward dry-daylight imagery from East Asian road types may limit generalisation to other environments.

XII. FUTURE WORK

The current deployment establishes a robust baseline for CPU-based edge inference, yet several avenues for future enhancement remain. To improve detection reliability in edge-case environments, such as low-light or high-moisture conditions, future iterations will explore **multi-sensor fusion**. By correlating visual camera data with vibration telemetry from a high-frequency accelerometer, the system could confirm physical road anomalies that are visually obscured.

Furthermore, to mitigate the computational burden of redundant processing across consecutive frames, we intend to integrate lightweight tracking algorithms like **SORT** (Simple Online and Realtime Tracking). This would allow the system to maintain a persistent ID for a single anomaly across a video sequence, reporting it to the central database only once and reducing redundant logging.

To achieve ultra-high-resolution inference or to support multiple simultaneous camera feeds, future development will transition from pure CPU execution to **Hardware Accelerators**, such as the **Google Coral TPU**. Utilizing the Edge TPU’s specialized ASIC for 8-bit math would theoretically allow the system to maintain 60+ FPS at 640px resolution with significantly lower thermal output. Finally, we aim to explore

Adaptive Resolution Scaling, where the system dynamically adjusts the input tensor size based on vehicle velocity to optimize the balance between detection look-ahead distance and processing throughput.

XIII. CONCLUSION

This paper has presented a complete edge-AI road anomaly detection pipeline targeting the Raspberry Pi 5 (ARM Cortex-A76). By selecting YOLO26n for its NMS-free, DFL-free, and quantization-tolerant architecture, curating a 75,000+ image merged dataset across seven sources, applying a three-stage progressive training curriculum with hard-negative mining, and deploying via ONNX Runtime INT8 at 640×640 resolution, the system achieves 9.72 FPS on-device - exceeding the 5 FPS real-time requirement by more than $2\times$ - alongside mAP@0.50 of 0.7104 on a held-out test set. The results demonstrate that production-quality road damage detection is achievable on commodity ARM hardware without cloud dependency or dedicated accelerators.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to Dr. Pravin Nair for his invaluable guidance, continuous support, and for providing the essential hardware equipment required for this research. We also extend our thanks to J. Karthik for his technical guidance and support throughout the development of this project.

REFERENCES

- [1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Adv. Neural Inf. Process. Syst.*, vol. 28, 2015.
- [2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE CVPR*, 2016.
- [3] M. Hussain, "Object detection using YOLO: Challenges, architectural successors, datasets and applications," *Multimedia Tools Appl.*, 2022.
- [4] R. Sapkota *et al.*, "YOLO26: Key architectural enhancements and performance benchmarking for real-time object detection," *arXiv:2509.25164*, 2025.
- [5] R. Mansour *et al.*, "YOLO26: An analysis of NMS-free end-to-end framework for real-time object detection," *arXiv:2601.12882*, 2026.
- [6] R. Sapkota *et al.*, "Ultralytics YOLO evolution: An overview of YOLO26, YOLO11, YOLOv8, and YOLOv5," *arXiv:2510.09653*, 2025.
- [7] D. Arya, H. Maeda, S. K. Ghosh, D. Toshniwal, and Y. Sekimoto, "RDD2020: An annotated image dataset for automatic road damage detection using deep learning," *Data in Brief*, vol. 36, p. 107133, 2021.
- [8] D. Arya, H. Maeda, S. K. Ghosh, D. Toshniwal, and Y. Sekimoto, "RDD2022: A multi-national image dataset for automatic Road Damage Detection," *arXiv:2209.08538*, 2022.
- [9] L. Arcioni, "Road Damage Dataset: Potholes, Cracks and Manholes," Kaggle, 2022.
- [10] R. Suresh, "RAD — Road Anomaly Detection," Kaggle, 2023.
- [11] DataClusterLabs, "Potholes or Cracks on Road Image Dataset," Kaggle, 2021.
- [12] D. Arya *et al.*, "Global Road Damage Detection: State-of-the-art Solutions," in *Proc. IEEE Int. Conf. Big Data*, 2020, pp. 5533–5539.
- [13] S. Behera, "RDD China," Kaggle, 2022.
- [14] D. Arya *et al.*, "Crowdsensing-based Road Damage Detection Challenge (CRDDC'2022)," in *Proc. IEEE Int. Conf. Big Data*, 2022, pp. 6378–6386.
- [15] G. Varma, A. Subramanian, A. Nambodiri, M. Chandraker, and C. V. Jawahar, "IDD: A Dataset for Exploring Problems of Autonomous Navigation in Unconstrained Environments," in *Proc. IEEE WACV*, 2019, pp. 1743–1751.
- [16] A. Pawar, "Potholes Detection YOLOv8," Kaggle, 2023.
- [17] I. Shamsutdinov, "Road Damage Dataset for Object Detection," Kaggle, 2022.
- [18] H. Maeda, Y. Sekimoto, T. Seto, T. Kashiwayama, and H. Omata, "Road Damage Detection and Classification Using Deep Neural Networks with Smartphone Images," *Comput.-Aided Civ. Infrastruct. Eng.*, vol. 33, no. 12, pp. 1127–1141, 2018.
- [19] A. Kumar, "Pothole Detection Dataset," Kaggle, 2022.
- [20] Chitholian, "Annotated Potholes Dataset," Kaggle, 2021.
- [21] S. Rath, "Road Pothole Images for Pothole Detection," Kaggle, 2021.
- [22] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
- [23] M. Kerrisk, *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. San Francisco, CA: No Starch Press, 2010.
- [24] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF CVPR*, 2018, pp. 2704–2713.
- [25] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.
- [26] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Accurate post training quantization with small calibration sets," in *Proc. 38th Int. Conf. Mach. Learn. (ICML)*, vol. 139, pp. 4466–4475, 2021.
- [27] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF CVPR*, 2018, pp. 2704–2713.
- [28] M. Nagel, R. A. Amjad, M. van Baalen, C. Louizos, and T. Blankevoort, "Up or down? Adaptive rounding for post-training quantization," in *Proc. 37th Int. Conf. Mach. Learn. (ICML)*, vol. 119, pp. 7197–7206, 2020.
- [29] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei, "Deep feature flow for video recognition," in *Proc. IEEE CVPR*, 2017, pp. 4141–4150.