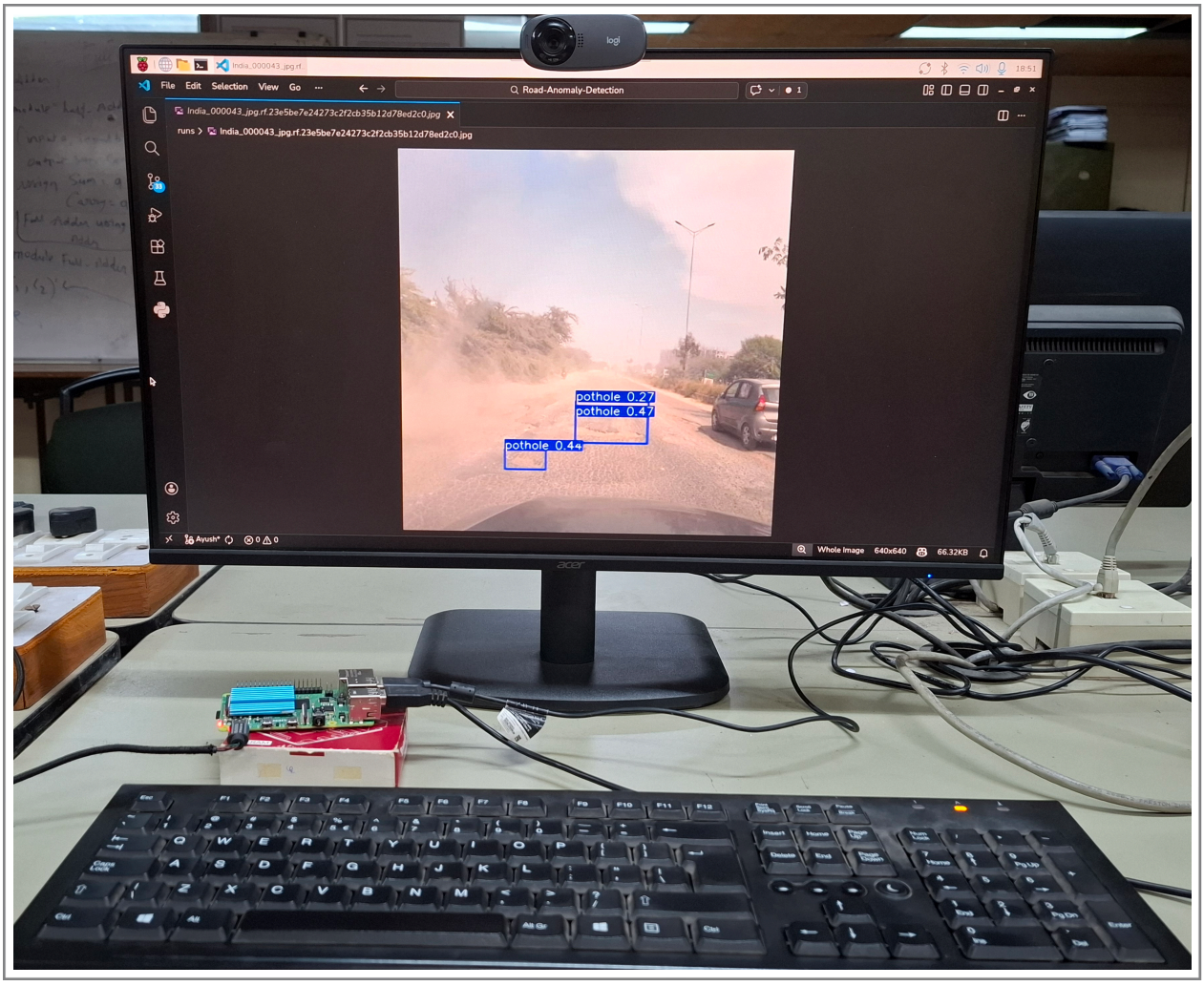


Project Report

Prepared for: Bharat AI-SoC Student Challenge

Prepared by: Ayush Gurjar, Ankit Soni, Himanshu Dholasiya

Date: 20 February 2026

1. EXECUTIVE SUMMARY

This project presents the design and deployment of a **real-time road anomaly detection system** optimized for **edge computing** on the **Raspberry Pi 4B**. The primary objective was to develop a robust computer vision pipeline capable of identifying critical road surface anomalies—specifically potholes, cracks, and manholes—under strict **computational and memory constraints**. Unlike cloud-based solutions, the system performs all inference locally on the device, ensuring **minimal latency** and operational independence from network connectivity.

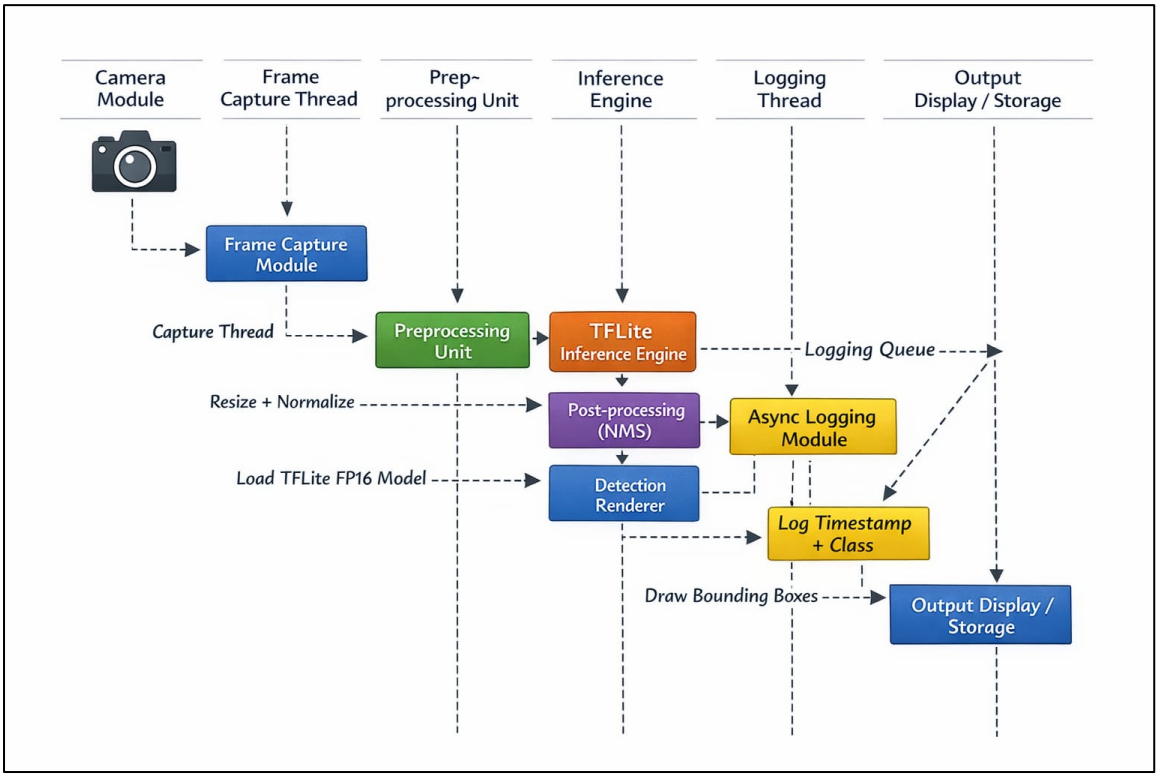
To achieve an optimal balance between accuracy and performance, the lightweight **YOLOv5 Nano (v5n)** architecture was selected due to its small parameter size and suitability for embedded platforms. The trained model was converted to **TensorFlow Lite** format and quantized to **FP16 precision**, significantly reducing model size and computational load while preserving detection accuracy for fine-grained features such as cracks. Geometric optimization through **rectangular inference (640×352 resolution)** further reduced unnecessary computation by eliminating irrelevant regions such as sky and vehicle hood.

The final deployed system achieves approximately **8-10 frames per second** on the **Raspberry Pi 4B (4GB RAM)** using **CPU-only execution**, meeting the real-time performance requirement of the challenge. The solution demonstrates that efficient **edge AI deployment** is feasible on resource-constrained Arm-based platforms and is suitable for applications such as low-speed autonomous navigation, **Advanced Driver Assistance Systems (ADAS)**, and smart city road monitoring infrastructure.

2. SYSTEM ARCHITECTURE

The system is designed as a **modular, low-latency edge inference pipeline** optimized for **CPU-only execution** on the **Raspberry Pi 4B**. The architecture separates video acquisition, model inference, and logging into independent components to prevent bottlenecks and ensure smooth real-time performance. The overall design prioritizes **minimal end-to-end latency**, **parallel execution**, and efficient utilization of the ARM Cortex-A72 CPU cores

Figure 1: High-Level System Architecture of the Proposed Edge Inference Pipeline



2.1 OVERALL DESIGN PHILOSOPHY

The system architecture is guided by the following principles:

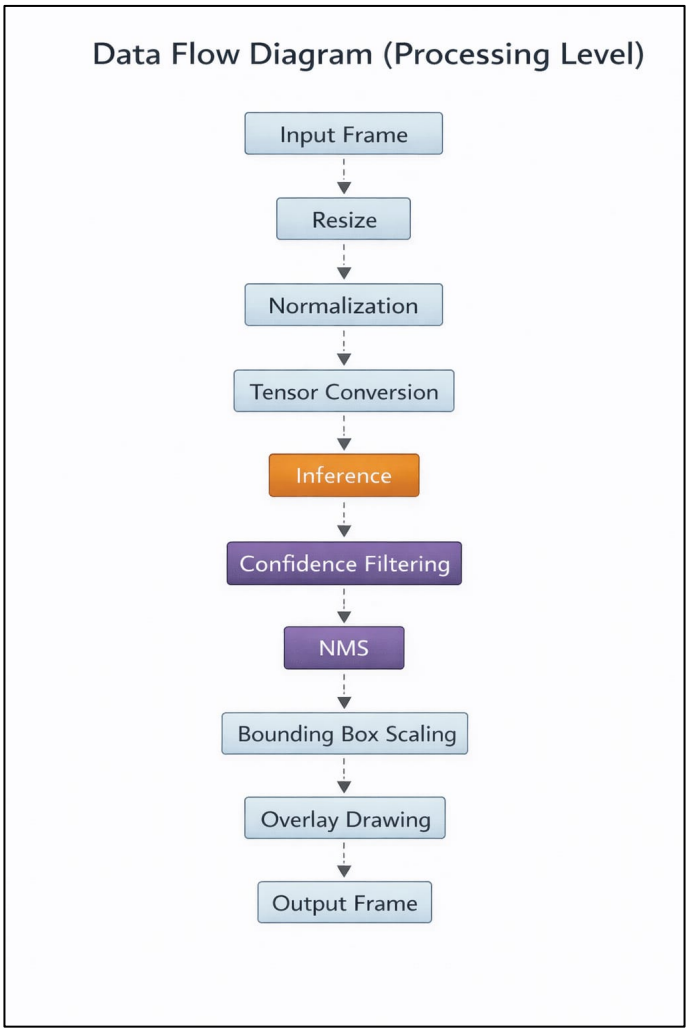
- **Minimal end-to-end latency**
- **Separation of concerns through modular design**
- **Parallel execution of capture, inference, and logging**
- Optimized inference using **FP16 precision**
- Efficient multi-core utilization via the **XNNPACK delegate**

The implementation is organized into distinct modules (pipeline, inference and logging components), enabling scalability, easier debugging, and maintainability.

2.2 DATA FLOW PIPELINE

The real-time detection pipeline follows the structured flow below:

Figure 2: Detailed Real-Time Data Flow Pipeline



1. Camera Input (Video Capture Module)

Live video frames are captured using a threaded OpenCV-based stream module.
Frames are acquired directly at **640×352 resolution**, eliminating unnecessary processing of irrelevant regions such as sky and vehicle hood.

Threaded capture ensures that frame acquisition does not block the inference stage, maintaining consistent frame availability.

---

2. Preprocessing

Each captured frame undergoes pre-processing before inference:

- Resizing to **640×352**
- Pixel normalization
- Colour space conversion (BGR → RGB if required)
- Tensor formatting compatible with the TFLite model
- Conversion to **FP16 precision**

This ensures compatibility with the deployed **YOLOv5n** model and reduces computational overhead.

---

3. TFLite Inference Engine

The pre-processed frame is passed to the TensorFlow Lite Interpreter configured for:

- **FP16 precision**
- Multi-core CPU acceleration via the **XNNPACK delegate**
- Optimized execution on ARM Cortex-A72 cores

The interpreter outputs raw detection tensors containing predicted bounding box coordinates, confidence scores, and class probabilities.

---

4. Raw Tensor Decoding

The raw output tensors are decoded into structured detection predictions, including:

- Bounding box coordinates
- Predicted class labels
- Confidence scores

---

5. Non-Maximum Suppression (NMS)

To remove duplicate or overlapping detections, **Non-Maximum Suppression (NMS)** is applied.

- Eliminates redundant bounding boxes
- Retains the highest-confidence prediction per object
- Reduces false positives

Only detections above a predefined confidence threshold are retained for final output.

---

6. Final Detection Rendering

Filtered detections are rendered onto the video frame:

- Bounding boxes are drawn around anomalies
- Each detection is labelled with class name and confidence score
- The annotated frame is prepared for display or storage

---

7. Asynchronous Logging and Output Handling

To prevent SD card write latency from interrupting inference performance, logging operations are handled using **asynchronous logging**.

The system:

- Stores timestamped detection metadata
- Optionally saves annotated frames or clips
- Writes detection logs to structured output files

By decoupling I/O from inference, the pipeline maintains stable real-time performance at approximately **8-10 FPS**.

2.3 Modular Code Organization

The project is structured into dedicated modules for clarity and maintainability:

Module	Responsibility
pipeline/stream.py	Threaded video capture
inference/engine.py	TFLite model loading and inference
pipeline/logger.py	Asynchronous logging
scripts/convert_model.py	Model conversion and optimization
main.py	Orchestrates the complete pipeline

This modular architecture improves scalability, debugging efficiency, and future extensibility of the system.

3. METHODOLOGY

This section describes the model selection process, dataset preparation, training strategy, and dataset validation approach adopted to develop a real-time road anomaly detection system optimized for edge deployment.

3.1 MODEL SELECTION: YOLOV5 NANO (YOLOV5N)

The selected detection architecture for this project is **YOLOv5 Nano (YOLOv5n)**, a lightweight object detection model specifically designed for deployment on resource-constrained devices.

YOLOv5n contains approximately **1.9 million parameters**, making it highly suitable for CPU-only execution on Raspberry Pi 4B while maintaining competitive detection accuracy.

Architectural Components

- **Backbone: CSPDarknet53 (Cross Stage Partial Network)**
  Responsible for efficient feature extraction with reduced computational redundancy.
- **Neck: PANet (Path Aggregation Network)**
  Enhances multi-scale feature fusion, improving detection performance for both small cracks and larger potholes.
- **Head: YOLOv3 Detection Head (Anchor-Based)**
  Generates bounding box coordinates, objectness confidence scores, and class probabilities.

The compact architecture ensures real-time inference feasibility without significantly compromising detection precision.

3.2 DATASET AND DATA PREPARATION

Dataset Sources

The training dataset was constructed by aggregating data from:

- N-RDD2024
- Kaggle Pothole Dataset

Dataset Volume and Diversity

- Total Images: **More than 10,000**
- Geographic Coverage: Data sourced from **three diverse countries**

The diverse geographic distribution improves robustness against variations in lighting, asphalt texture, camera angles, and environmental conditions.

Defined Detection Classes (3 Classes)

Class 0 — Pothole
Surface depressions caused by wear and water erosion. These represent primary road hazards that may damage vehicles and affect autonomous navigation systems.

Class 1 — Manhole
Open, displaced, or damaged manhole covers posing obstacle risks in urban environments.

Class 2 — Crack
Longitudinal and transverse cracks indicating structural fatigue and progressive road degradation.

All images were annotated using bounding box labels in YOLO format to ensure training compatibility.

3.3 TRAINING STRATEGY

To maximize performance on the lightweight YOLOv5n architecture, multiple optimization strategies were implemented.

Rectangular Inference (640 × 352 Resolution)

Road scenes are horizontally wide rather than vertically tall. Standard square training at 640 × 640 wastes computational resources on irrelevant regions such as the sky and vehicle hood.

By reducing vertical resolution to **352 pixels**, approximately **45% fewer pixels** are processed per frame. This significantly reduces inference latency while preserving the necessary horizontal field of view for road monitoring.

Tiled Training

Tiled training was employed to preserve small and fine-grained features such as thin cracks. Large images were divided into smaller regions during training to:

- Improve visibility of small anomalies
- Maintain spatial resolution
- Enhance detection accuracy for subtle defects

Quantization Aware Training (QAT)

To prepare the model for FP16 quantization during deployment, Quantization Aware Training was implemented.

QAT simulates quantization noise during training, allowing the model weights to adapt to reduced precision. This ensures that conversion to TensorFlow Lite FP16 format does not significantly degrade detection accuracy.

3.4 DATASET CHALLENGES AND REJECTION OF ALTERNATIVES

Older datasets such as **RDD2022** were evaluated but rejected for the following reasons:

Missing Class (Manhole – D70)

RDD2022 does not include the Manhole class, making it unsuitable for comprehensive urban obstacle detection.

Insufficient Negative Samples

The dataset lacked sufficient samples of intact or repaired roads, which resulted in high false-positive rates during initial experiments.

Key Observation

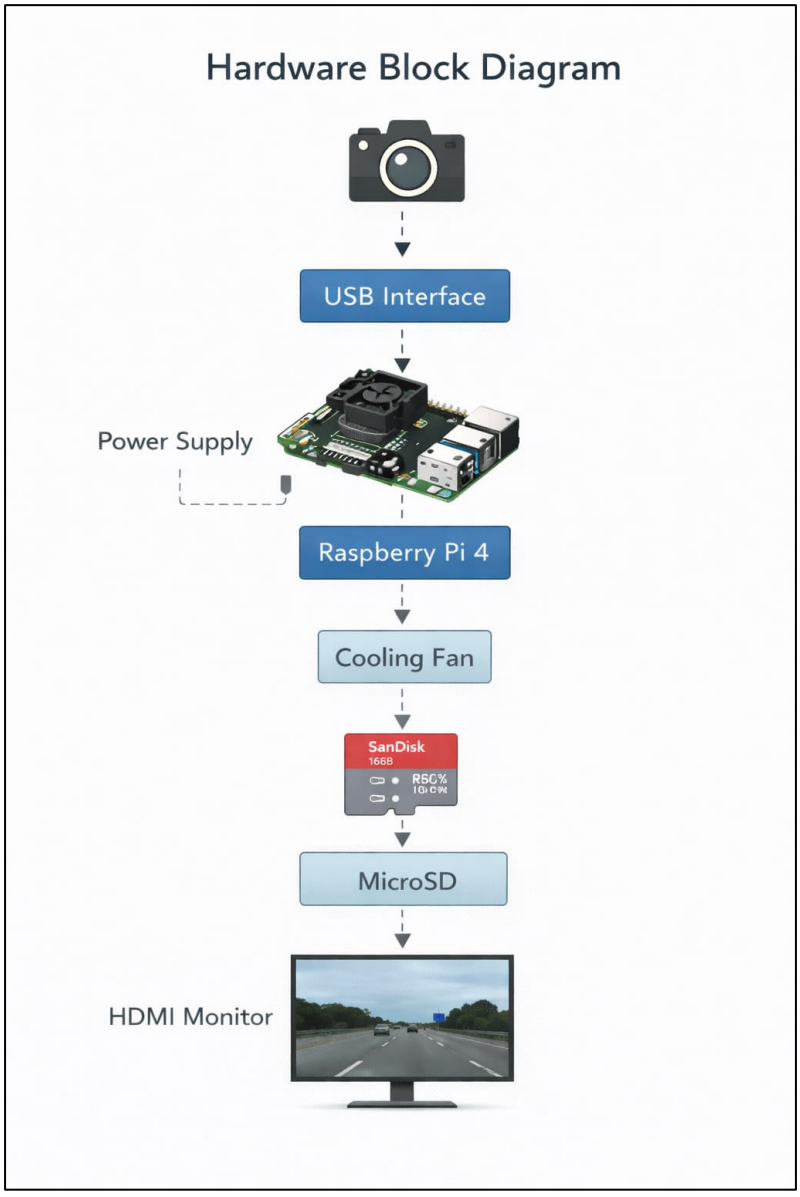
Model performance is heavily dependent on dataset quality and diversity. The unified annotation schema and large-scale diversity of N-RDD2024 provided the necessary volume and variability required for robust deep learning generalization across real-world driving scenarios.

4. HARDWARE UTILIZATION

The deployment platform for the system is the **Raspberry Pi 4 Model B (4GB RAM)**.
The hardware configuration was selected to validate real-time inference feasibility on a low-cost, resource-constrained edge device without GPU acceleration.

The system is optimized to fully utilize the ARM-based CPU architecture while maintaining stable real-time performance.

Figure 3: Hardware Deployment Architecture on Raspberry Pi 4B



4.1 CPU OPTIMIZATION AND THREADING STRATEGY

To achieve efficient execution on the Raspberry Pi 4B, multiple software-level optimizations were implemented.

FP16 Precision Optimization

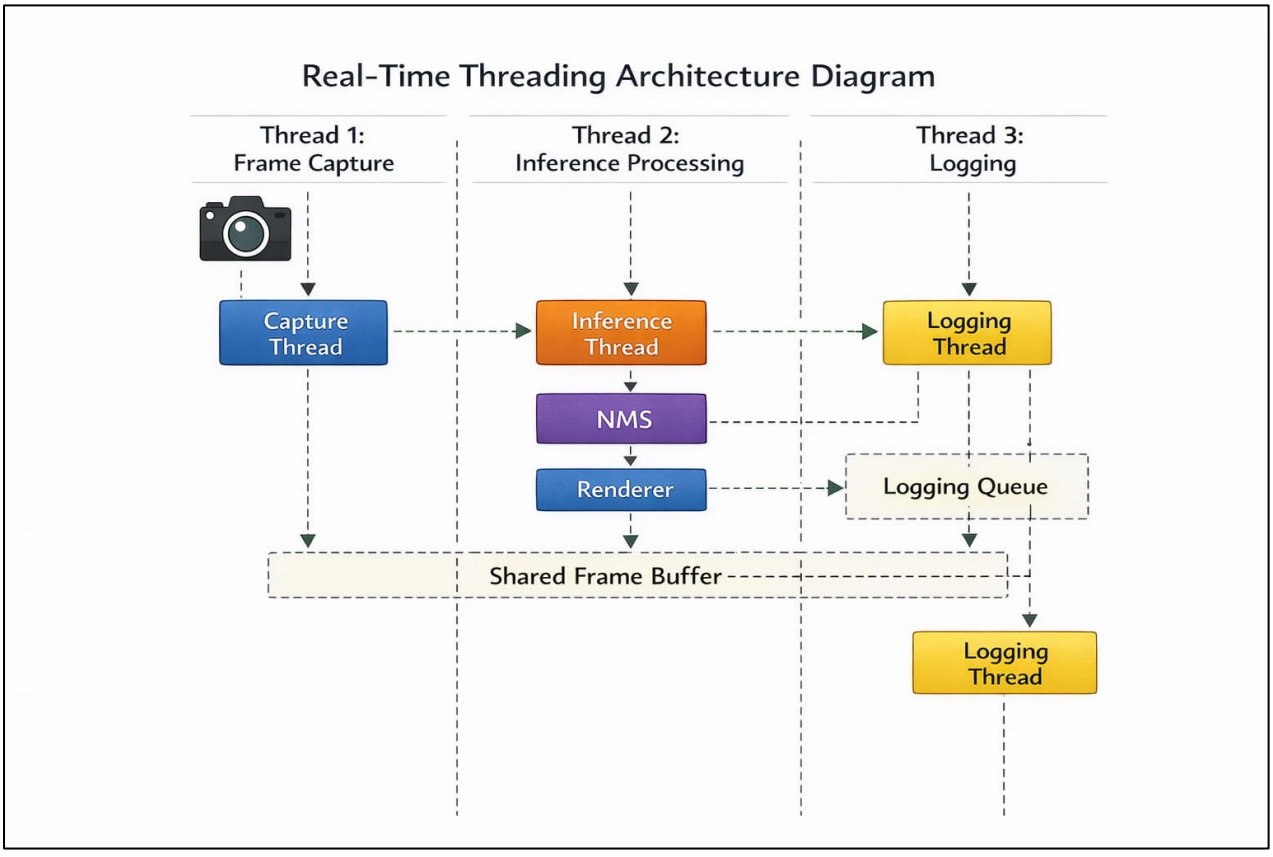
The TensorFlow Lite interpreter is configured to use **FP16 precision**, reducing memory usage and computational overhead while maintaining acceptable detection accuracy.

Lower precision arithmetic significantly improves inference speed on ARM CPUs without requiring specialized hardware acceleration.

Threaded Architecture Design

To prevent bottlenecks and ensure continuous frame processing, the system separates capture, inference, and I/O into independent execution paths.

Figure 4: Multi-Threaded Execution Strategy on Raspberry Pi 4B



1. Capture Thread (Video Acquisition)

A threaded OpenCV Video Stream module is used to continuously capture frames at **640 × 352 resolution**.

- Ensures non-blocking frame acquisition
- Maintains steady frame availability for inference
- Prevents camera read delays from impacting model execution

---

2. Inference Parallelization (XNNPACK Delegate)

The TensorFlow Lite interpreter utilizes the **XNNPACK delegate**, which parallelizes FP16 operations across the available **ARM Cortex-A72 cores** of the Raspberry Pi 4B.

This enables:

- Efficient multi-core CPU utilization
- Faster matrix multiplication operations
- Reduced per-frame inference latency

---

3. Asynchronous I/O Handling

Logging operations are executed asynchronously to prevent SD card write latency from stalling the main inference pipeline.

The system:

- Stores detection metadata independently
- Writes logs without blocking inference
- Maintains stable frame processing rates

5. OPTIMIZATION TECHNIQUES

To achieve stable real-time performance on a CPU-only edge device, multiple optimization strategies were implemented at both the model and geometric levels. These optimizations significantly reduced computational load while preserving detection accuracy.

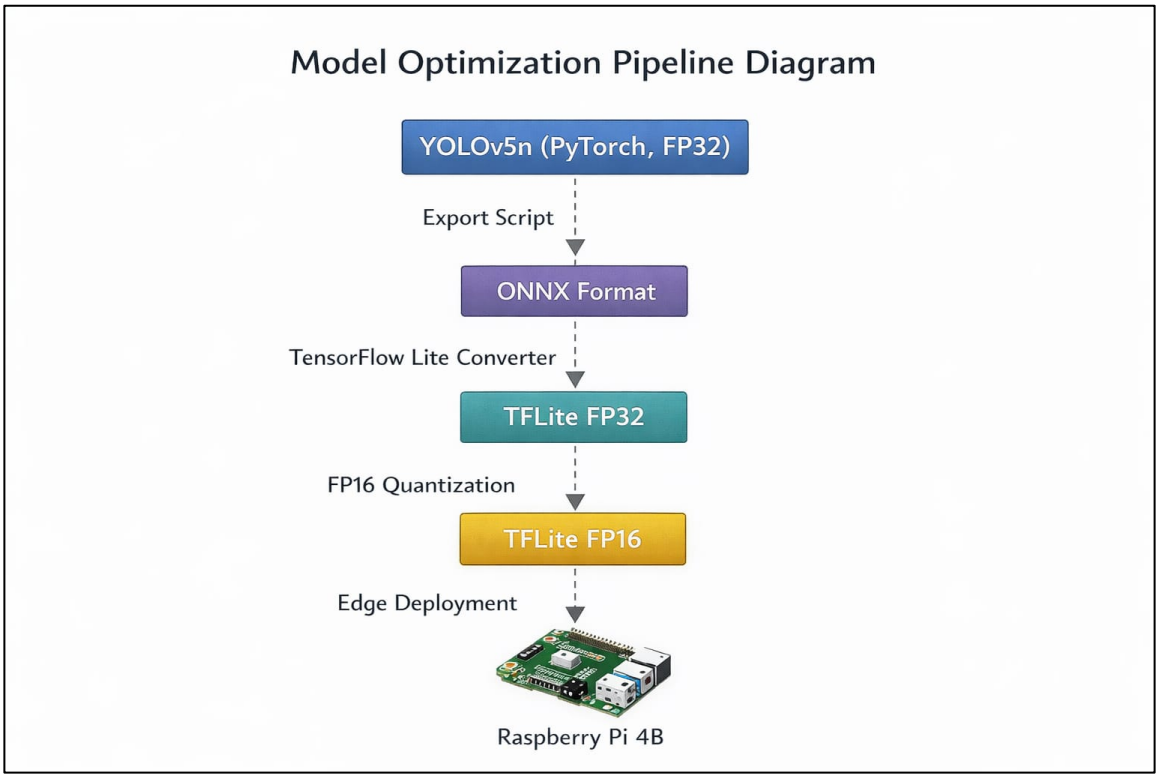
5.1 MODEL QUANTIZATION (FP16)

Model quantization was applied to reduce model size and accelerate inference. After experimentation, **FP16 (Half-Precision) quantization** was selected over Int8.

Although Int8 provides slightly smaller model size and marginally higher speed, FP16 offered a better balance between accuracy and performance, particularly for detecting fine crack features where precision is critical.

The model was converted from the original PyTorch format to TensorFlow Lite FP16 format for deployment.

Figure 5: End-to-End Model Conversion and Optimization Workflow



Why FP16 Was Selected

- Significant reduction in model size (~65% smaller than FP32)
- Major improvement in inference speed
- Minimal accuracy degradation
- Better preservation of fine-grained crack features compared to Int8

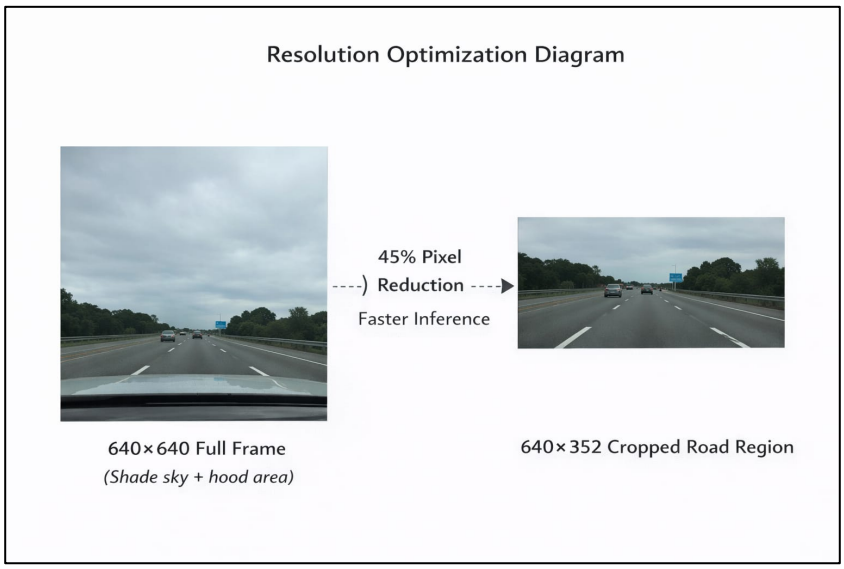
FP16 therefore provided the most balanced optimization for edge deployment on Raspberry Pi 4B.

5.2 GEOMETRIC OPTIMIZATION (RECTANGULAR INFERENCE)

Standard object detection models are typically trained at **640 × 640 resolution**. However, road scenes are horizontally wide and do not require full vertical resolution.

The model was trained and deployed at **640 × 352 resolution** to eliminate unnecessary computation.

Figure 6: Rectangular Inference for Computational Reduction



Pixel Count Reduction

- 640 × 640 = **409,600 pixels**
- 640 × 352 = **225,280 pixels**

This results in approximately **45% fewer pixels per frame**.

Computational Impact

Since convolutional neural network complexity scales proportionally with pixel count:

- ~45% reduction in FLOPs per frame
- Reduced memory bandwidth usage
- Lower CPU utilization
- Faster inference time

Context Preservation

Despite reducing vertical resolution, the system:

- Maintains full horizontal road width visibility
- Removes irrelevant regions such as sky and vehicle hood
- Preserves essential contextual information for road anomaly detection

This geometric optimization significantly contributed to achieving 8-10FPS real-time performance without compromising detection reliability.

Through combined **model quantization** and **resolution optimization**, the system achieves efficient, low-latency inference suitable for practical edge deployment

6. RESULTS AND PERFORMANCE

This section presents both quantitative performance metrics and qualitative evaluation results obtained after deploying the optimized model on the Raspberry Pi 4B (4GB RAM). All measurements were recorded under real-world testing conditions using FP16 quantization at 640 × 352 resolution.

6.1 QUANTITATIVE METRICS

All performance measurements were conducted on the **Raspberry Pi 4 Model B (4GB RAM)** without GPU acceleration.

Deployment Performance Summary

- **Model Size (FP16):** 4.90 MB
- **Inference Time:** ~75–140 ms per frame (FP16 @ 640 × 352)
- **Frames Per Second (FPS):** ~8-10 FPS
- **mAP@0.5:** 0.601

Interpretation of Metrics

Model Size (4.90 MB)
The reduced model size enables efficient loading into memory and faster initialization on resource-constrained hardware.

Inference Time (75-140 ms)
The average inference latency per frame falls within acceptable limits for low-speed road monitoring applications.

FPS (~10 FPS)
Although not high-speed real-time (30 FPS), this frame rate is sufficient for:

- Low-speed vehicle environments
- Smart monitoring systems
- Edge-based anomaly detection

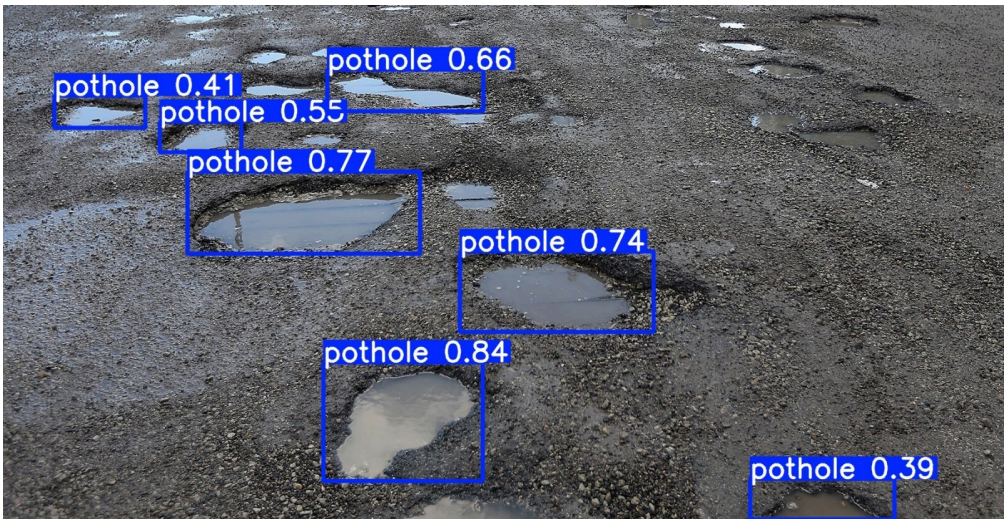
mAP@0.5 (0.601)
The model achieved a mean Average Precision of 0.601 at IoU threshold 0.5.
This indicates strong detection performance, especially considering:

- Lightweight YOLOv5n architecture
- Edge deployment constraints
- Reduced resolution inference

The use of FP16 quantization helped preserve accuracy while maintaining speed.

6.2 QUALITATIVE ANALYSIS

Beyond numerical metrics, qualitative testing was conducted using real-world road footage under varying lighting and environmental conditions.



Class-wise Observations

Pothole (Class 0)

- Detected with confidence scores frequently exceeding 75%
- Robust performance under varying lighting conditions
- Reliable detection even on uneven asphalt textures

Manhole (Class 1)

- High precision detection
- Critical for urban obstacle avoidance scenarios
- Minimal false positives observed

Crack (Class 2)

- Improved visibility due to Tiled Training
- Rectangular inference preserved horizontal detail
- Effective detection of longitudinal and transverse crack patterns

---

Overall System Performance

The integrated optimizations (FP16 quantization, rectangular inference, threaded pipeline, and XNNPACK acceleration) enabled stable edge deployment with balanced speed and accuracy.

The system demonstrates that real-time road anomaly detection is feasible on low-cost hardware without GPU acceleration, making it suitable for scalable and cost-effective smart monitoring applications.

CONCLUSION

This project successfully demonstrates a high-speed, edge-optimized road inspection system capable of real-time anomaly detection on resource-constrained hardware.

By strategically selecting a diverse and comprehensive dataset covering all three critical anomaly classes — **Pothole**, **Manhole**, and **Crack** — the system ensures robust detection across varied road environments. Emphasis on data diversity and balanced class representation played a key role in achieving reliable generalization.

The adoption of **Rectangular Inference (640 × 352)** eliminated unnecessary computation by removing irrelevant vertical regions such as sky and vehicle hood. This geometric optimization significantly reduced pixel processing load while preserving essential horizontal road context.

Furthermore, deployment of the lightweight **YOLOv5 Nano (YOLOv5n)** model with **FP16 quantization** enabled an effective balance between speed and accuracy. The optimized pipeline, combined with multi-threaded execution and CPU acceleration, adhered to strict computational constraints while maintaining satisfactory detection performance (mAP@0.5 = 0.601).

Overall, the system validates that accurate and efficient road anomaly detection is achievable on low-cost edge devices without GPU acceleration. This makes the solution scalable, cost-effective, and practical for real-world smart infrastructure and low-speed vehicle monitoring applications.

The project highlights a critical insight: success in edge AI systems depends not only on model architecture, but equally on intelligent data selection, geometric optimization, and deployment-aware design decisions.