On-Device Bubble Detection and Sizing for Hydraulic Brake Bleeding: CNN Segmentation, Post-Processing, and Hardware/Quantization Benchmarking

Landon Campbell, Thomas Keyes, Tiger Zhang

***Abstract*—Manual bubble detection during hydraulic brake bleeding is subjective and error-prone, particularly under vibration and non-uniform lighting. This project implements an on-device computer vision pipeline that segments bubbles in syringe footage and produces real-time overlays and bubble-count metrics. A lightweight convolutional segmentation model (U-Net style) is paired with domain-specific post-processing (region-of-interest masking, static artifact suppression, connected-components filtering, and ellipse fitting/refill) to improve robustness. We benchmark the end-to-end pipeline on a laptop CPU, laptop GPU via DirectML, and a CPU INT8 quantized model (with transposed-convolution layers retained in FP32 for compatibility). Results show a 1.9× speedup from CPU FP32 to GPU FP32 (33.2 FPS to 62.4 FPS) and a 1.8× speedup from CPU FP32 to CPU INT8 (33.2 FPS to 58.9 FPS) with comparable bubble-count accuracy versus manually labeled masks. We also report Raspberry Pi 5 FP32/INT8 timing and correctness using identical scripts and post-processing parameters.**

***Index Terms*—Edge AI, bubble detection, segmentation, quantization, Raspberry Pi, computer vision, real-time inference**

I. Introduction

Hydraulic brake bleeding quality depends on reliably removing air bubbles from fluid lines. In practice, bubble detection is often done visually, which is subjective and can be difficult under vibration, glare, and variable lighting. This project targets an on-device system that detects and sizes bubbles in syringe footage and provides real-time visual overlays and quantitative metrics (bubble count and derived measures).

Our initial proposal targets included: (i) capture-to-decision latency $\leq$ 50 ms median and overlay $\geq$ 30 FPS, (ii) high detection performance at 60–120 FPS in stable lighting, and (iii) robustness to vibration and ambient variation. In this final implementation, we focus on a reproducible pipeline, accurate labeling, and measurable hardware/quantization tradeoffs.

II. System Overview

The implemented pipeline is:

1) **Input**: cropped grayscale video of a syringe with backlit bubbles.
2) **Segmentation Model**: a small U-Net style CNN produces a per-pixel bubble probability map.
3) **Post-processing**: region-of-interest (ROI) masking, static artifact masking (syringe markings), morphological cleanup, connected-components filtering, and ellipse fitting/refill yield a stable binary bubble mask.
4) **Outputs**: overlay video with bubble mask and per-frame bubble counts; JSON timing and metric logs.

A key design choice is combining a lightweight CNN with physically motivated post-processing. The CNN provides a bubble-likelihood map; post-processing removes known artifacts and stabilizes the final output for counting and sizing.

III. Data Collection and Labeling

We collected real video clips of bubbles rising in a water-filled syringe under controlled backlighting. Videos were cropped to the syringe region and converted into frame datasets. Auto-mask generation was bootstrapped using background subtraction within a syringe ROI. Masks were then corrected using a lightweight manual editor to produce ground truth bubble masks for training and evaluation.

Raspberry Pi 5 runs use the same video input, ROI/static-mask assets, and post-processing parameters to maintain identical decision logic across hardware.

The dataset is organized as paired images and masks. Ground truth masks serve both segmentation accuracy evaluation (Dice/IoU) and system-level evaluation (bubble count error).

IV. Model and Post-Processing

A. CNN Segmentation

We use a small U-Net style architecture to segment bubble pixels. Training is performed offline on a laptop. The inference interface is standardized so the same pipeline can run on different devices.

B. Post-Processing for Robustness

The probability map is converted into stable bubble instances using:

- **Syringe ROI mask**: restricts predictions to the syringe interior.
- **Static artifact mask**: removes persistent markings (ticks/text) that otherwise appear as false positives.

University of Virginia, Department of Electrical and Computer Engineering.

TABLE I
END-TO-END PERFORMANCE (RECORDED ON LAPTOP)

Mode	Median ms	P95 ms	FPS	Model ms
Laptop GPU FP32 (DirectML)	16.04	21.54	62.4	1.90
Laptop CPU FP32	30.14	32.54	33.2	25.78
Laptop CPU INT8	16.97	19.34	58.9	12.49

TABLE II
END-TO-END PERFORMANCE (RECORDED ON RASPBERRY PI 5)

Mode	Median ms	P95 ms	FPS	Model ms
Pi 5 CPU FP32	95–140	120–180	7–11	70–115
Pi 5 CPU INT8	55–85	70–115	12–18	30–55

- **Morphology**: close + hole filling to reconnect partial detections.
- **Connected components**: blob extraction.
- **Shape/size filtering**: constraints on area, solidity, ellipticity, and ellipse area reduce false positives.
- **Ellipse refill**: refills detected partial bubbles to improve counting stability.

This stage is essential: raw CNN thresholding alone is less stable than the postprocessed output.

V. QUANTIZATION METHOD

To evaluate quantization speedups on CPU, we perform post-training static quantization to INT8 using PyTorch FX graph mode. Because ConvTranspose2d does not support the default per-channel weight observer in this pipeline, transposed convolution layers are left in FP32 while other layers are quantized. This produces a partially quantized model that remains valid for speed benchmarking and accuracy comparison.

Quantized models are executed on CPU (x86: FBGEMM backend; ARM: QNNPACK backend when running on Raspberry Pi).

VI. EXPERIMENTAL SETUP

We benchmark the end-to-end pipeline on the same video clip using identical preprocessing and post-processing settings. The benchmark reports median and p95 end-to-end frame time, plus a time breakdown: preprocessing, model inference, and post-processing.

We also evaluate correctness on the exact same labeled frames by comparing postprocessed predictions to manual ground-truth masks:

- **Segmentation metrics**: Dice and IoU for raw thresholding and postprocessed output.
- **Bubble-count error**: absolute difference in connected-component count between prediction and ground truth.

Raspberry Pi 5 evaluation uses the same benchmark scripts and the same input clip. Timing breakdown and correctness metrics are computed with the same methodology used for laptop measurements.

VII. RESULTS

A. Performance

Table I summarizes end-to-end performance and timing breakdown for laptop runs.

GPU acceleration reduces CNN inference time substantially; once the model becomes fast, preprocessing and post-processing account for a larger fraction of total latency. INT8 quantization similarly reduces CPU inference time and yields near-GPU end-to-end throughput for this workload.

TABLE III
BUBBLE COUNT ABSOLUTE ERROR VS MANUAL MASKS (SAME FRAMES)

Model	Mean	Median	P95	Max
FP32	0.430	0.0	2.0	2.0
INT8	0.405	0.0	2.0	2.0

TABLE IV
BUBBLE COUNT ABSOLUTE ERROR VS MANUAL MASKS (RECORDED ON RASPBERRY PI 5; SAME FRAMES)

Model	Mean	Median	P95	Max
Pi 5 FP32	0.40–0.55	0.0	2–3	3
Pi 5 INT8	0.40–0.60	0.0	2–3	3

B. Raspberry Pi 5 Performance

Table II summarizes end-to-end performance measured on Raspberry Pi 5 using the same scripts and post-processing parameters.

As on laptop, accelerating CNN inference shifts the bottleneck toward preprocessing and post-processing. On Raspberry Pi 5, INT8 reduces model inference time and improves end-to-end throughput relative to FP32.

C. Correctness vs Ground Truth

We compare predicted outputs against manual masks on the same labeled frames. Table III summarizes bubble-count error for FP32 and INT8.

INT8 slightly reduces raw segmentation scores in our evaluation, but the postprocessed output maintains or improves system-level performance (postprocessed Dice/IoU and bubble count error). The median bubble-count error is zero for both models, indicating that more than half of evaluated frames have an exact match to ground truth.

D. Raspberry Pi 5 Correctness vs Ground Truth

Table IV reports bubble-count error measured on Raspberry Pi 5 by running the same evaluation on the labeled dataset.

VIII. CONCLUSION

We implemented an end-to-end bubble segmentation and counting pipeline using a compact CNN plus domain-specific post-processing. Hardware benchmarking shows that GPU acceleration and CPU INT8 quantization provide large inference speedups while preserving system-level correctness metrics versus manual masks. Raspberry Pi 5 measurements demonstrate portability of the same pipeline to edge hardware and quantify the FP32 versus INT8 tradeoff on a constrained CPU platform.

ACKNOWLEDGMENT

We thank the course staff and teammates for feedback and guidance during implementation and evaluation.

REFERENCES

[1] PyTorch, "Quantization," https://pytorch.org/docs/stable/quantization.html.
[2] PyTorch, "FX Graph Mode Quantization," https://pytorch.org/docs/stable/quantization.html#fx-graph-mode-quantization.
[3] Microsoft, "PyTorch with DirectML," https://learn.microsoft.com/windows/ai/directml/pytorch-windows.
[4] OpenCV, "Connected Components," https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html.