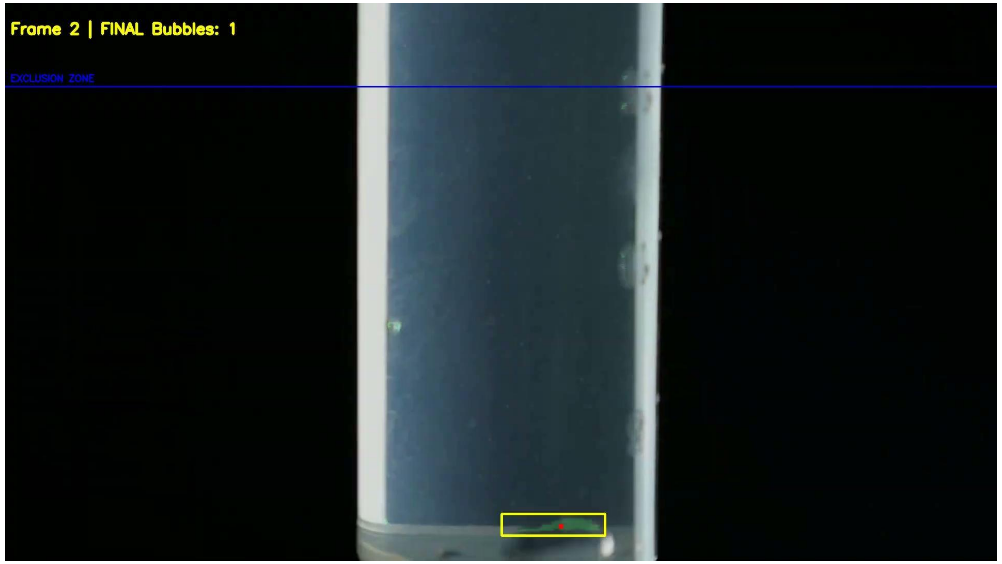
Raspi5 Bubble Detection

Final project • OpenCV and Trained CNN on Laptop • CNN on Raspberry Pi 5

TTL AI
Landon Campbell — integration & performance
Thomas Keyes — open CV software & CNN training
Tiger Zhang — CV pipeline & CNN training/deployment



Problem & Motivation

- Mountain bike brakes can trap microscopic air bubbles during bleeding
- Bubbles compress → inconsistent lever feel and reduced braking performance
- Manual detection is visual + subjective → need quantitative feedback
- Goal: on-device processing for a bike-shop friendly tool (no cloud)

Fig. 1 Mountain bike brake bleed with bubble release

Problem & Motivation Cont.

- Additional Applications:
  - Medical Setting: Current ultrasonic sensors are inconsistent
    - Air in IV or blood transfusion lines can cause severe complications
  - Industrial fluid & fuel systems - detect air/cavitation
  - Manufacturing & packaging - resin, beverage, or hydraulic quality control
  - Energy & environment - fuel cells, filtration, water loops
  - Generic optical flow sensor - particles, droplets, foam, or contaminants

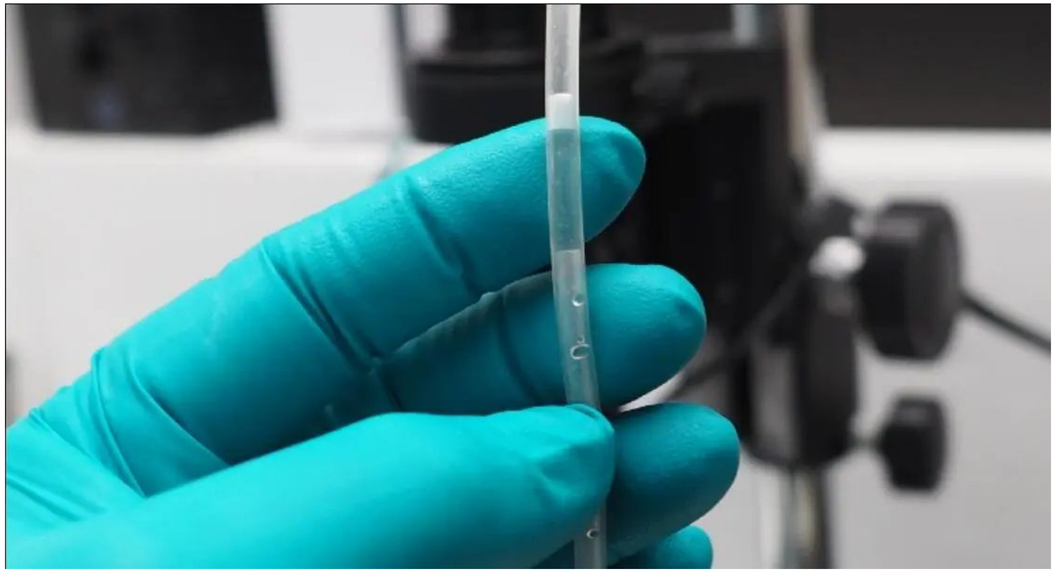


Fig 2. Air bubble in an IV line

Targets (What "Good" Looks Like)

- How we score it
  - Count stability (per-frame + cumulative
  - False positives (static artifacts removed)
  - Runtime: Pi CPU vs desktop CPU/GPU (ms/frame, FPS)
  - FP32 vs INT8 latency
  - Pixel mask (Dice) vs manual GT

Fig 3. Camera lighting and syringe setup

System Architecture

Laptop
OpenCV capture + baseline
CNN segmentation training (SmallUNet)
Produces probability map

→

Raspberry Pi 5
CNN running on live video feed, post processing for video overlay

→

Outputs
Mask + boxes
Bubble stats + logs

- Same post process on both paths: morphology + connected components + tracking filters
- Interfaces are swappable (CV model ↔ CNN model) for A/B testing
- Designed for edge-first: Pi handles camera/IO, live CNN application

Optics & Lighting: Make Bubbles Visible

- Controlled backlighting / dark background for high contrast
- Fixed geometry: camera + tube/syringe + shroud to kill clutter
- Stable exposure and repeatable ROI
- Black background created the cleanest training + inference domain

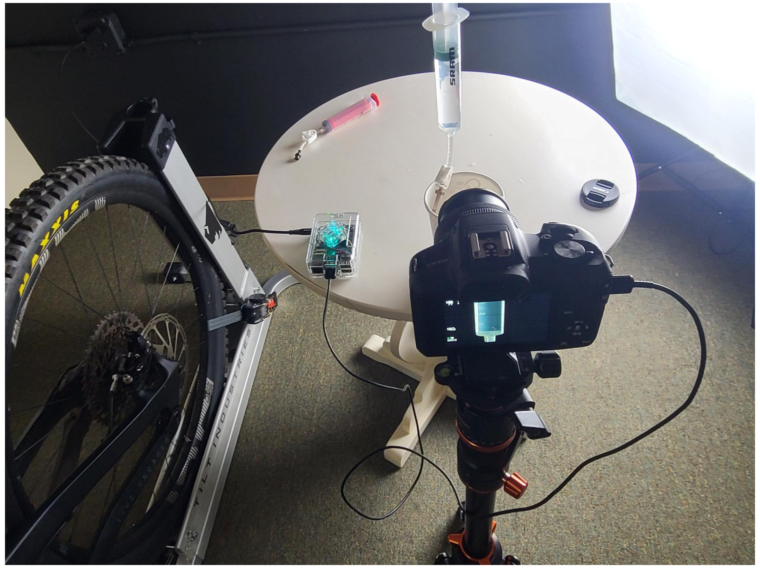


Fig 4. Camera connected to raspi 5 for live CNN running on syringe bubbles

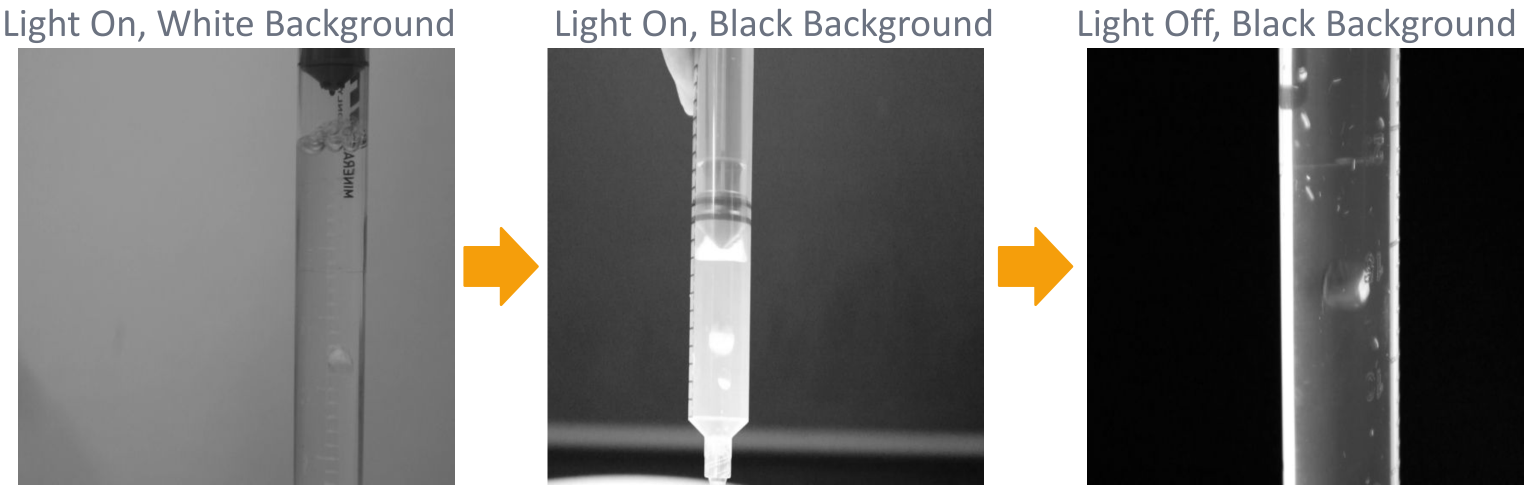


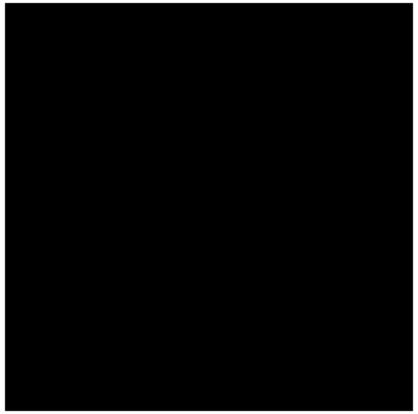
Fig 5. Lighting progression of physical setup

Dataset (Real Recorded Syringe Videos)

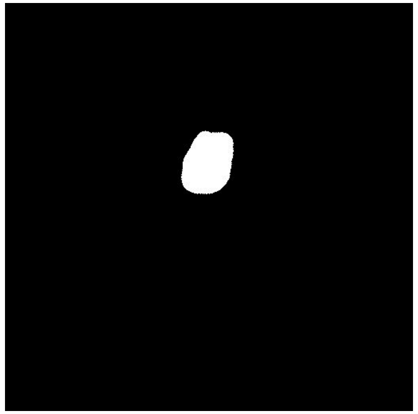
3 controlled videos:
- AIH_Bubbles.mp4 (190 frames)
- AIH_Bubbles2.mp4 (282 frames)
- AIH_Bubbles3.mp4 (183 frames)

Total: 9,344 image–mask pairs
- 8,070 manual masks (ground truth)
- 1,274 auto-masks (bootstrapped)

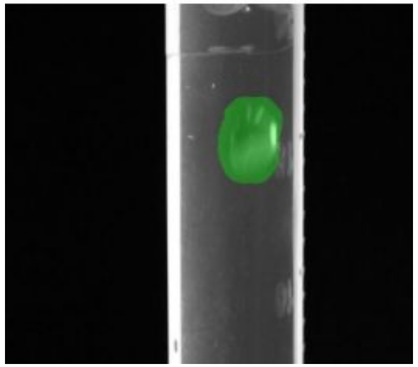
We generate multiple crops per frame → 9,344 training samples from 3 videos.



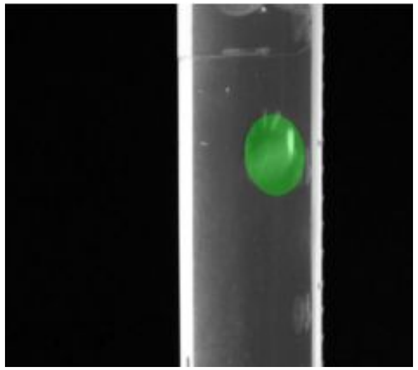
OpenCV auto mask segmentation



Manual mask correction



Manual mask correction overlay



CNN predicted mask overlay

- Manual masks mark full bubble volume (not just highlights)
- Auto-masks speed iteration and expand coverage

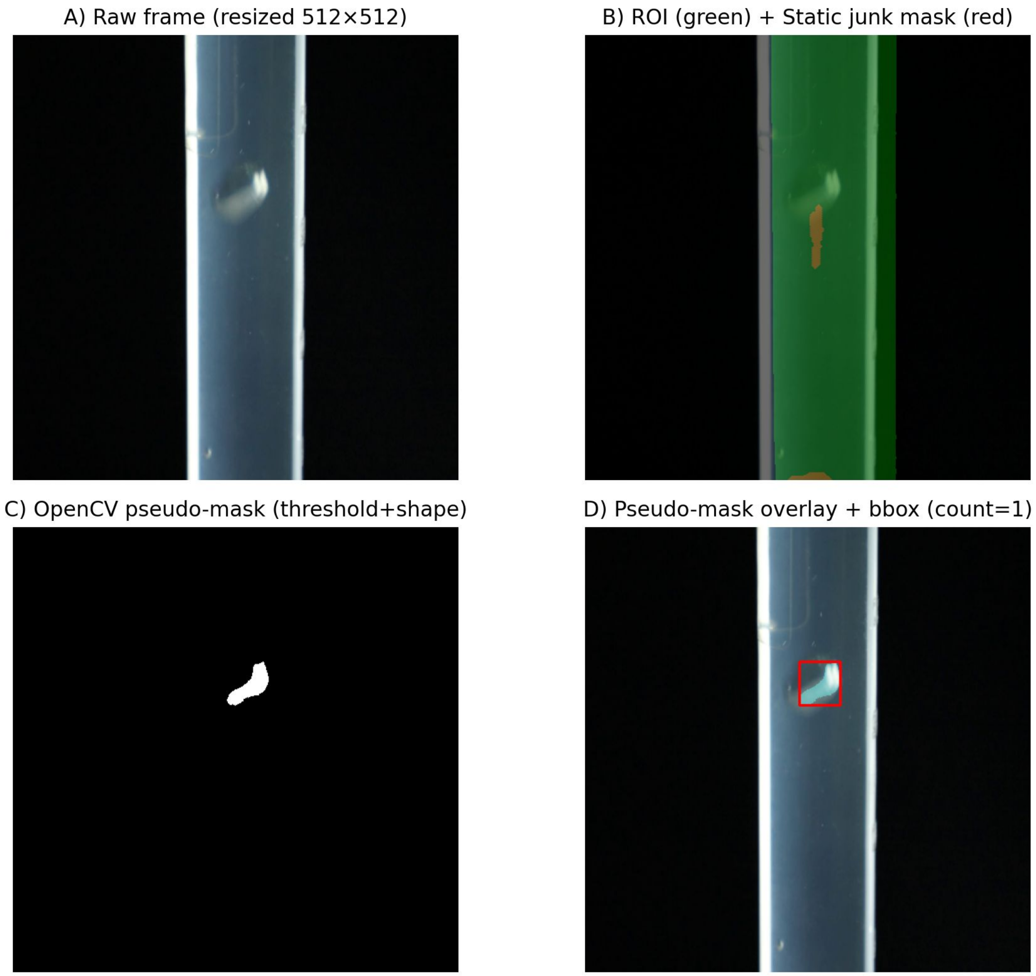
OpenCV Bootstrapping: Pseudo-Labels ⟶ Faster CNN Training

Goal: scale labeled data without hand-masking every frame

CV pipeline: (1) isolate syringe ROI ⟶ (2) threshold bright regions ⟶ (3) keep bubble-like blobs (circularity filter)

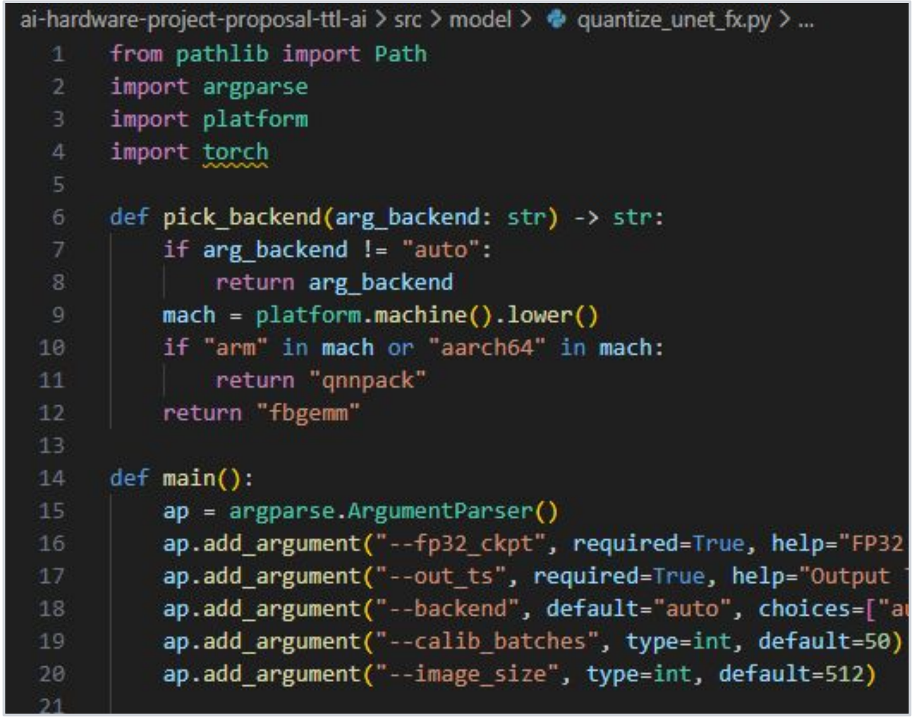
Output: 1,274 pseudo-masked samples from Bubbles3 used to expand coverage quickly

Key limitation: reflections + syringe markings can create false positives ⟶ CNN is needed for robustness



CNN Segmentation (SmallUNet) + Training Recipe

- Architecture: Small U-Net-like encoder/decoder with skip connections (~233K params)
- Training: 256×256 crops with weighted sampling (5× / 3× / 1×)
- Loss: Focal Loss (α=0.25, γ=2.0) for extreme class imbalance
- Augmentations: random crops, flips/rotations, color jitter/brightness
- Training on Apple M1 (MPS); early stop at epoch 36/100




```python
from pathlib import Path
import argparse
import platform
import torch

def pick_backend(arg_backend: str) -> str:
    if arg_backend != "auto":
        return arg_backend
    mach = platform.machine().lower()
    if "arm" in mach or "aarch64" in mach:
        return "qnnpack"
    return "fbgemm"

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--fp32_ckpt", required=True, help="FP32
    ap.add_argument("--out_ts", required=True, help="Output
    ap.add_argument("--backend", default="auto", choices=["a
    ap.add_argument("--calib_batches", type=int, default=50)
    ap.add_argument("--image_size", type=int, default=512)
```

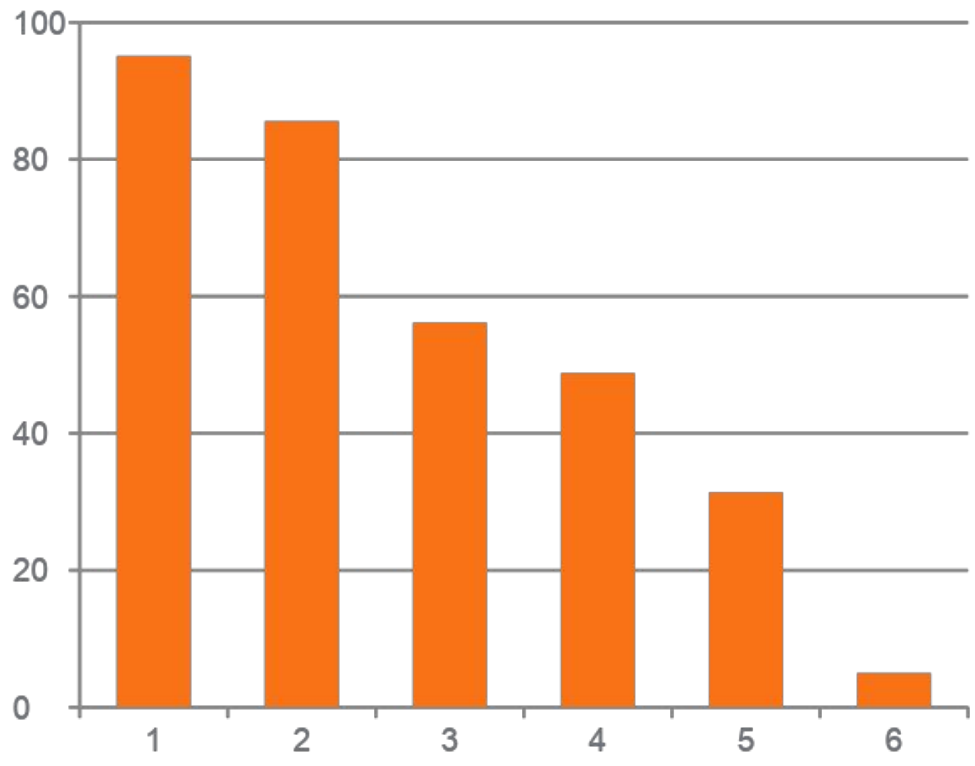
Fig 5. Snippet of quantized int8 UNET model

Validation vs manual GT: Pixel Dice is modest due to full-volume labels + refraction; system-level counting accuracy is the real KPI.

Key Insight: Raw CNN Output ≠ Reliable Detections

- On Bubbles3 (black background), masks closely match manual labels
- On Bubbles1/2, generalizes reasonably but needs heavy postprocess
- Temporal filtering is essential: motion tracking removes static glare/artifacts

False Positive Rate drops with stabilization (Bubbles3)



Postprocess reduced FP by ~91% (95.1% → ~5%)

Post Processing Pipeline (What Makes It Stable)

Stage A) Suppress known junk

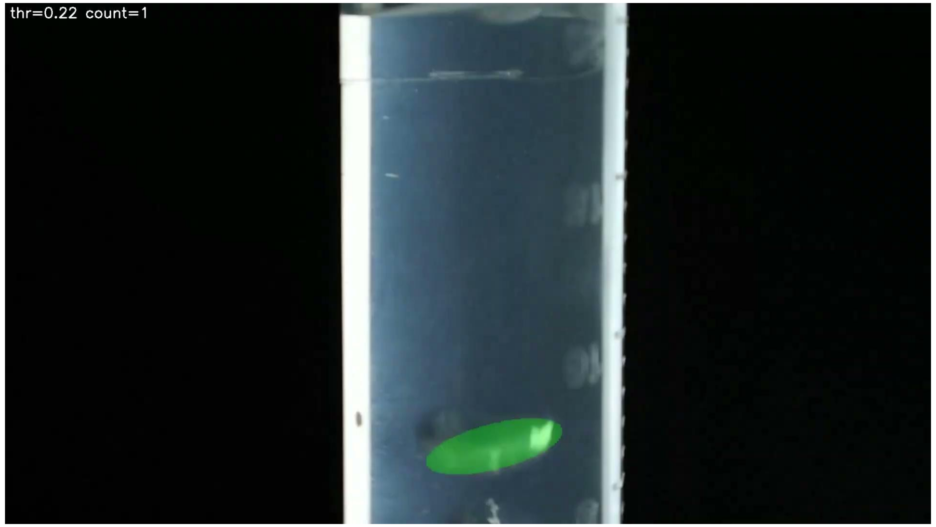
• ROI Mask: 256×256 tiles, stride 128 (stitch probability map)
• Static Junk Mask: Static variance filter: low variance (<10) → reject stuck detections

Stage B) Make blobs "bubble-like"

- Morphology close + fill holes: 15px elliptical dilation to merge refractive clusters
- Connected components: group contours into bubble instances

Stage C) Reject non-bubbles

•Ellipse/shape filters (circularity/solidity/axis-size): min area 3000 px; exclude top 15% zone
•Distance-to-edge gating (avoid syringe walls)



Final Validation Across All Real Videos

Video	Frames	Detections	Avg/Frame	Max	Std	GT estimate
Bubbles.mp4	190	7	0.04	1	0.19	Few actual bubbles
Bubbles2.mp4	282	113	0.40	3	0.59	Moderate bubbles
Bubbles3.mp4	183	350	1.91	7	1.17	Most bubbles
TOTAL	655	470	0.72	—	—	Validated

- Ran the same CNN + postprocess pipeline on Raspberry Pi 5 CPU, desktop CPU, and desktop GPU
- Benchmarked FP32 vs INT8 end-to-end latency + model time
- Verified bubble-count correctness vs manual masks (labeled frames) and on unseen frames
- Current demo: Pi camera capture + real-time overlay

Takeaway: counts match qualitative expectations (B1 few, B2 moderate, B3 most)

Hardware: Benchmarking & Deployment Status

Platform	Precision	Threads	Stride (every_n)	Model median (ms)	Core median (ms)	Total median (ms)	Total p95 (ms)	Total FPS
Raspberry Pi 5	FP32	4	1	809.50	897.62	1379.34	2888.28	0.72
Raspberry Pi 5	INT8	4	1	297.60	398.08	606.70	1514.53	1.65
Desktop CPU	FP32	8	2	27.08	31.67	41.87	47.08	23.88
Desktop GPU	FP32	8	2	12.49	16.74	16.97	19.34	58.92
Desktop CPU	INT8	8	2	25.78	30.16	30.14	32.54	33.18
Desktop GPU	INT8	8	2	2.78	7.38	16.22	21.53	61.66

Precision	Dice raw (mean)	Dice post (mean)	IoU raw (mean)	IoU post (mean)	Bubble count |err| (mean)	Bubble count |err| (median)	n	thr
FP32	0.337	0.627	0.233	0.546	0.430	0.0	79	0.22
INT8	0.320	0.658	0.217	0.576	0.405	0.0	79	0.22

Wrap-Up & Next Steps

- Working end-to-end pipeline: capture → segmentation → stabilization → bubble counts
- Big lesson: temporal filtering is mandatory for reliable deployment
- Next: expand labeled real footage; px→mm calibration; void-fraction metrics
- Next: tighten Pi benchmarking and publish latency/FPS + power numbers