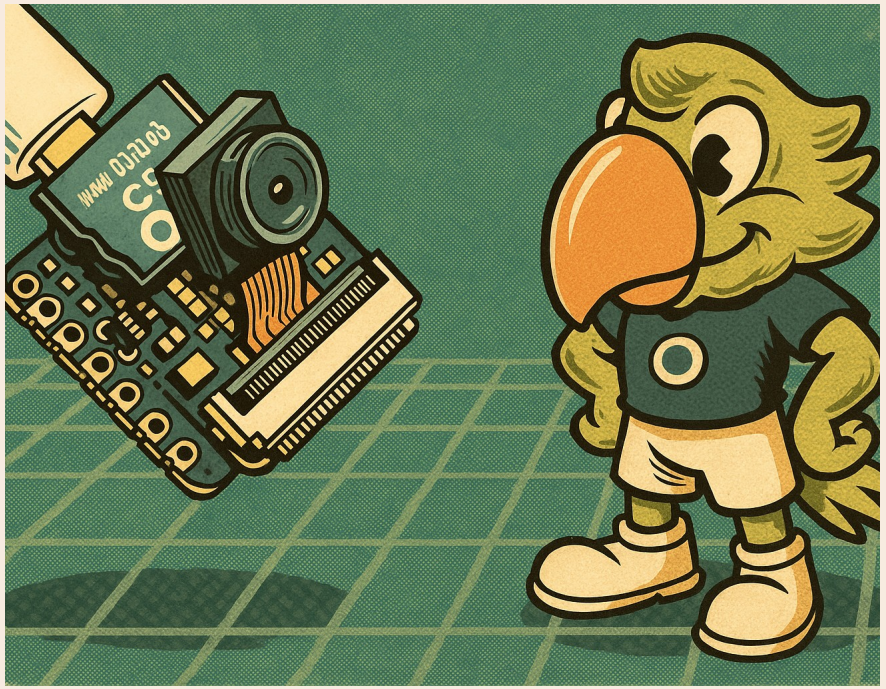
CHAPTER 2 · TINYML ON SEEED

Image Classification

Teaching a board to see



SENSECRAFT AI

EDGE IMPULSE

MOBILENET V2

52 slides · 4 labs · XIAO ESP32S3 Sense

AIEng4D

01

WHERE WE ARE · PART I

One pipeline, four labs

The sensor changes. The network changes. The five steps never do.

THE WHOLE BUDGET

600 milliseconds per frame. That is 1.7 frames per second, and 830 milliwatts.

Measured on the XIAO ESP32S3 Sense with the pre-trained Person Classification model.

LEARNING OBJECTIVES

What you will be able to do

01

Deploy a pre-trained model

SenseCraft AI, no code, a few minutes from plugging in to the first inference.

02

Collect your own dataset

Three classes, about fifty images each, from the board's own camera.

03

Train with transfer learning

MobileNet V2 at 96 by 96, quantized to int8, and know why.

04

Ship it twice

As a TFLite model on SenseCraft, then as an Arduino library you can edit.

DEFINITION

One image in, one label out. Everything else in computer vision is built on this.

Image classification assigns one label to the whole frame. It does not say where.

THREE DIFFERENT QUESTIONS

Classification, detection, segmentation

Classification

What is in this image? One label for the whole frame. Cheapest to run.

Detection

What is where? A box and a label per object. FOMO does this in the next module.

Segmentation

Which pixels belong to what? A mask, class by class. Usually out of reach on a microcontroller.

WHY IT FITS ON A MICROCONTROLLER

From AlexNet to MobileNet

2012 · AlexNet

About sixty million parameters. It showed that deep learning works for vision.

2014–2015 · VGG, ResNet

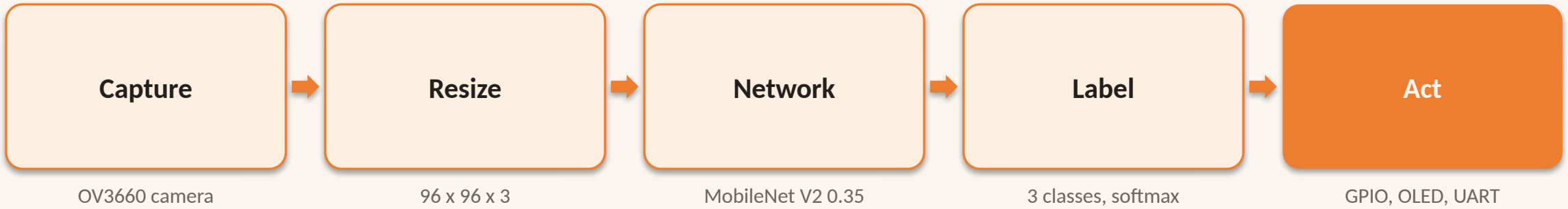
Deeper and deeper. Accurate, and far too heavy for a board.

2017 · MobileNet

Depthwise separable convolutions, and a width multiplier alpha. This is what we use.

THE PIPELINE

Every lab in this course uses the same five steps



The only thing that changes between labs is the sensor and the network.

02

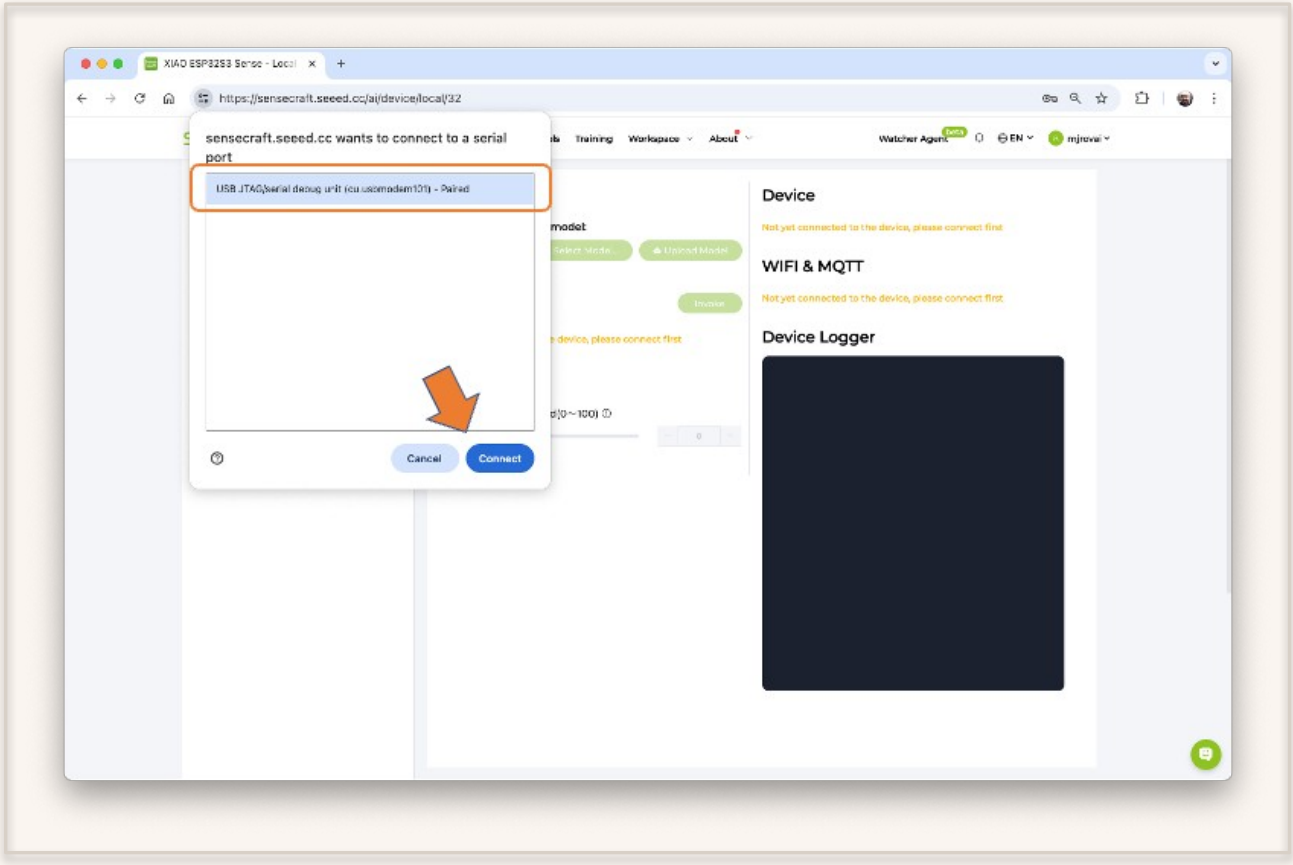
PART 1 OF 4 · NO CODE

Pre-trained, in five minutes

Plug the board into the browser and watch it classify.

SENSECRAFT AI · WORKSPACE

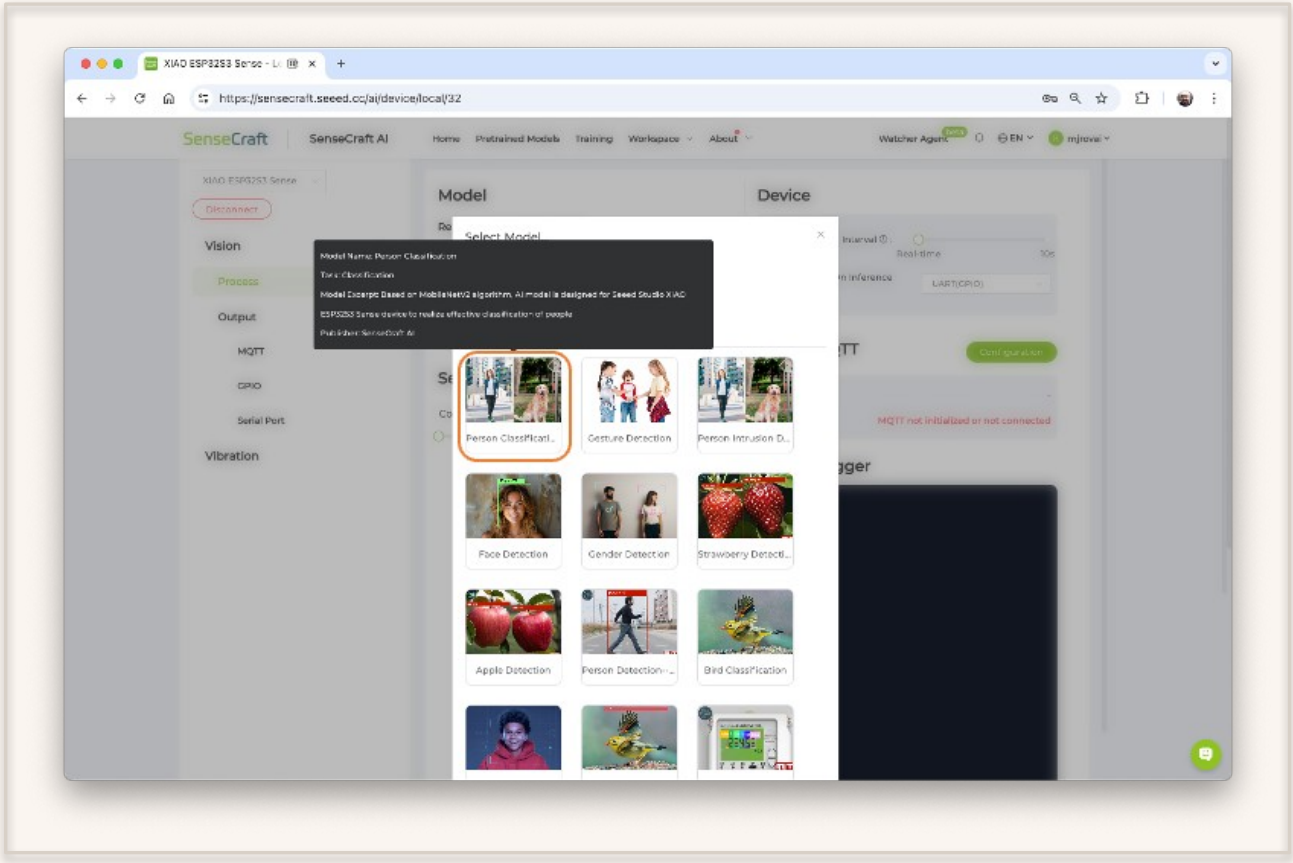
Connect the board, open the workspace



1. XIAO ESP32S3 Sense over USB-C, connected to the browser with Web Serial
2. Pick the serial port in the pop-up and press Connect
3. No toolchain to install — the browser flashes the firmware for you

MODEL LIBRARY

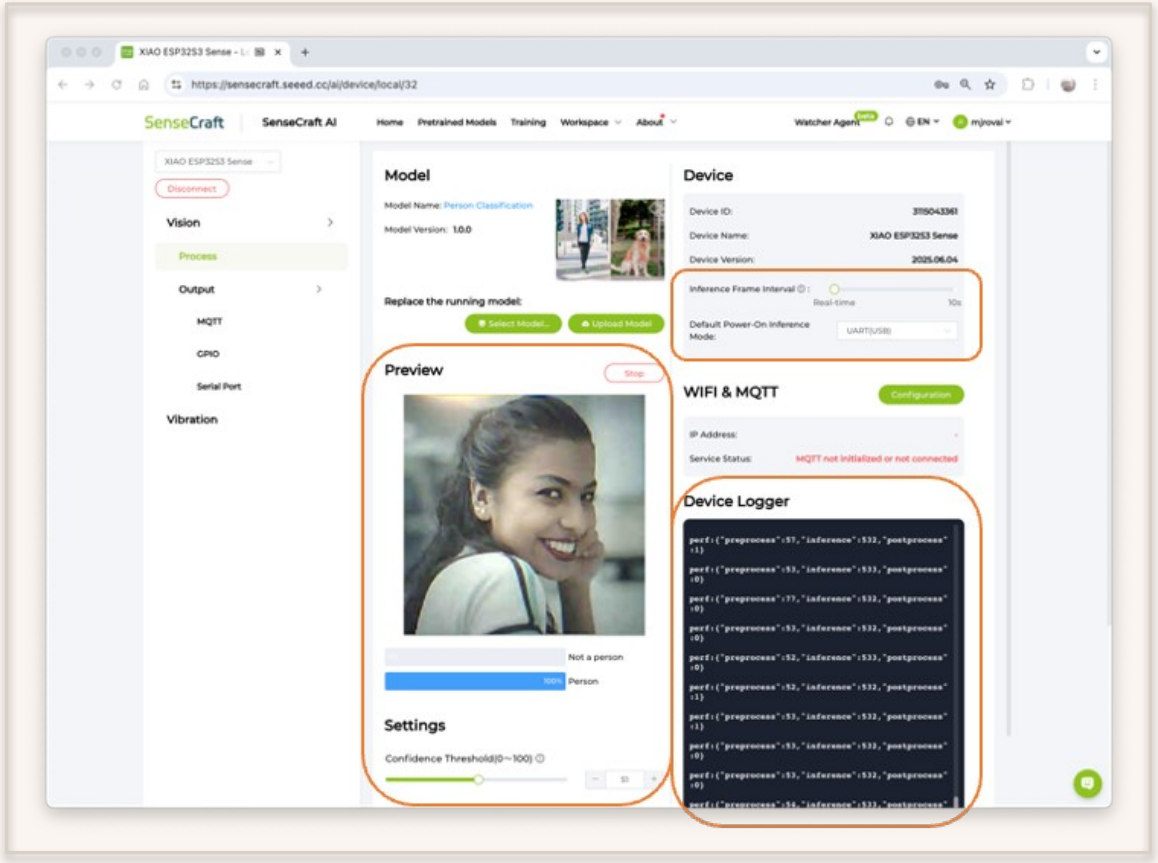
Search "Person Classification" and read the card



1. Select Model, then search the library by task
2. Hover a model: its main characteristics appear in a pop-up
3. Pick the Person Classification model trained on MobileNet V2

INFERENCE

Live feed, result, and the device logger



1. Preview: the live camera stream from the XIAO
2. Result: Person or Not a Person, with a confidence
3. Device logger: pre-processing and inference time, per frame

WHAT THE LOGGER SAYS

The cost of one frame, measured

QUANTITY	VALUE	NOTES
Pre-processing	52-78 ms	camera frame to model input
Inference	~532 ms	MobileNet V2 0.35, on the ESP32S3
Total per frame	~600 ms	about 1.7 frames per second
Peak current	160 mA	at 5.23 V
Power	830 mW	measured at the USB input

Pre-processing is not free — here it already takes about a tenth of every frame.

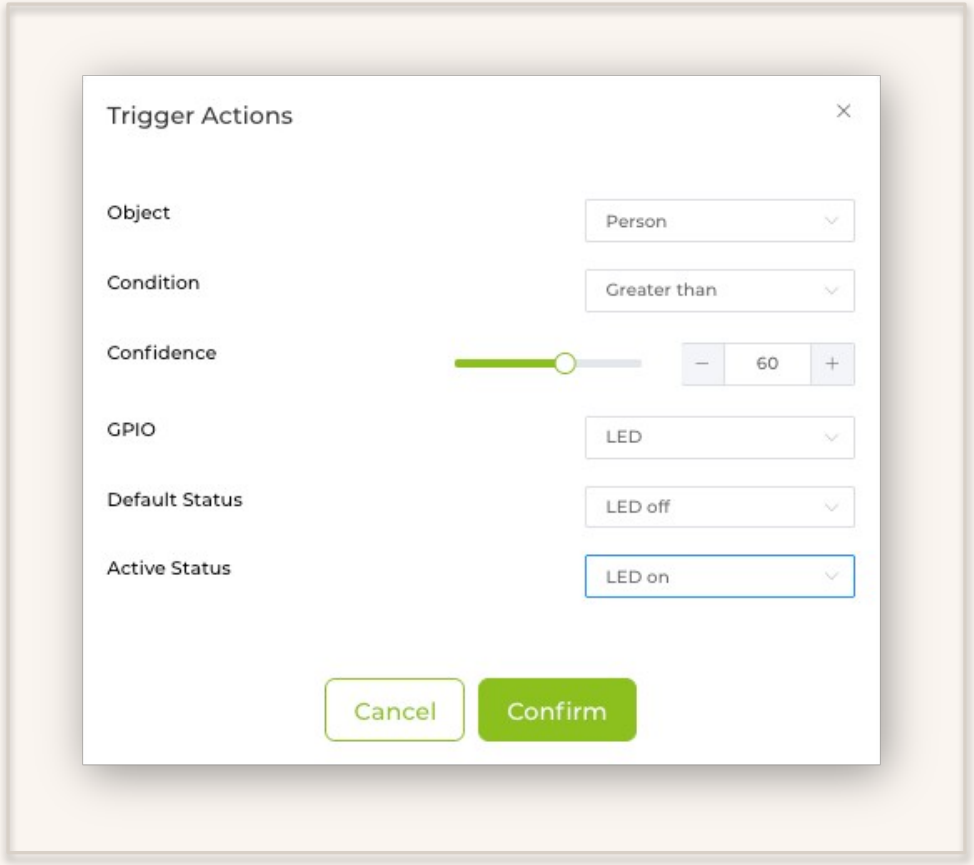
POST-PROCESSING

An inference is worth nothing until something in the world changes.

Turn a light on, move a servo, send a message — otherwise you only have a log line.

TRIGGER ACTIONS

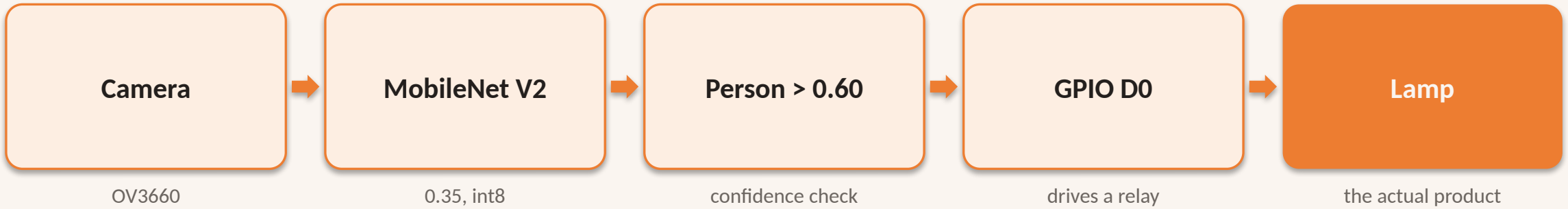
If the class is person, with confidence above 60 %, drive a GPIO



1. Condition: object Person, confidence greater than 60
2. Action: the internal LED for a first test, or D0, D1, D2, D11, D12 for a relay
3. Confirm, then press Send to upload the rule to the XIAO

THE WHOLE LOOP

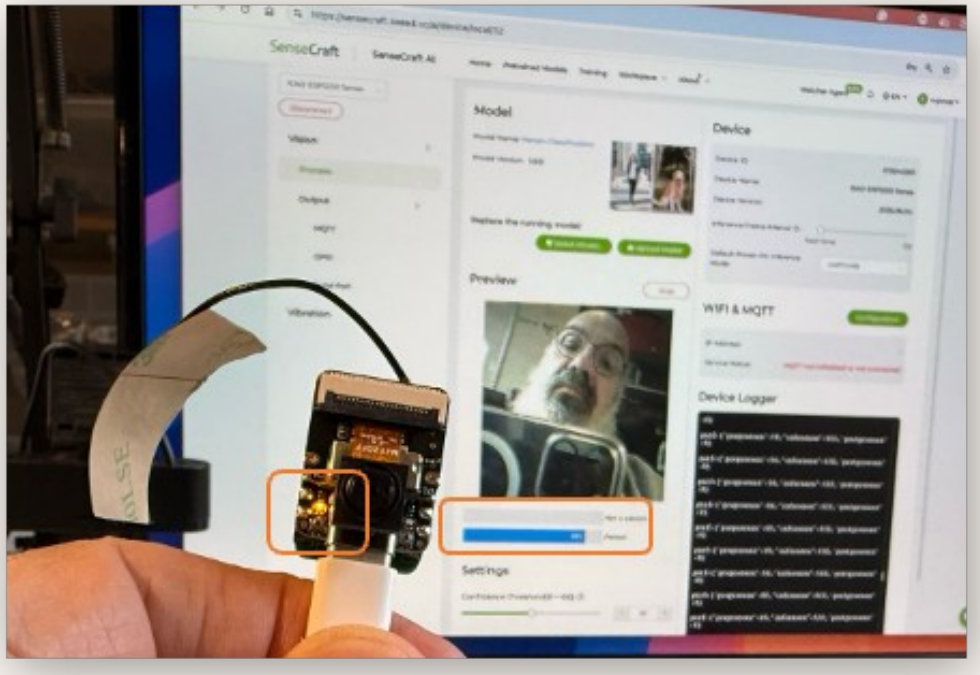
Person in frame, lamp on. Nobody in frame, lamp off.



Replace the lamp with a valve, a conveyor, or a door and you have an industrial product.

RESULT

The internal LED comes on while a person is detected



Same board, same model — the only difference is that the result now has a consequence.

HONEST SCOREBOARD

What you did, and what you did not

You did

Deployed a pre-trained model, read its latency, and wired the result to a GPIO.

You did not

Train anything. The dataset, the classes and the accuracy belong to someone else.

What is missing

Your own object, your own classes — and the memory budget to run them.

PART 2 OF 4 · NO CODE

03

Your own model, no code

Three classes, fifty images each, one training run.

THE GOAL COMES FIRST

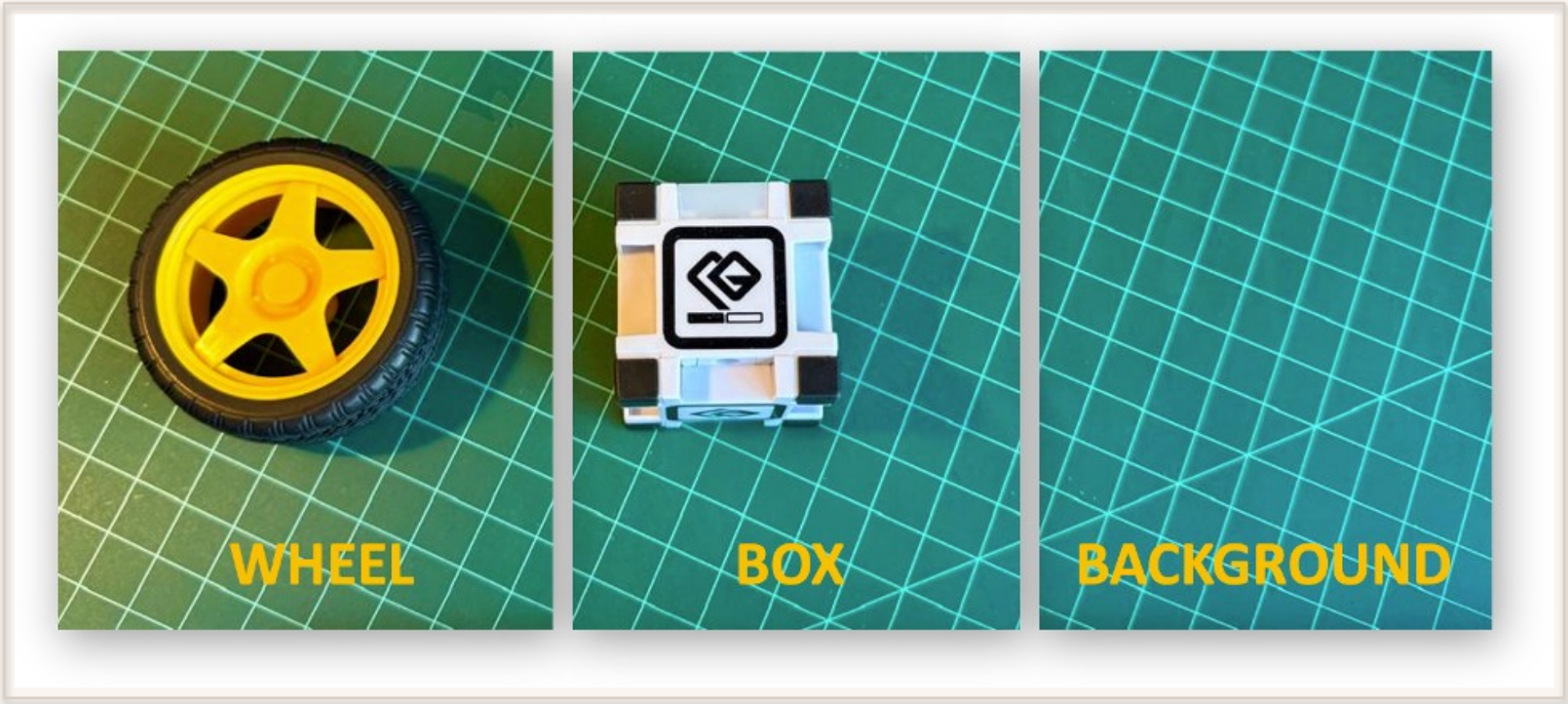
An installation that sorts wheels and boxes



Before any data, write down the decision. Here it is box, wheel, or nothing at all.

THREE CLASSES, NOT TWO

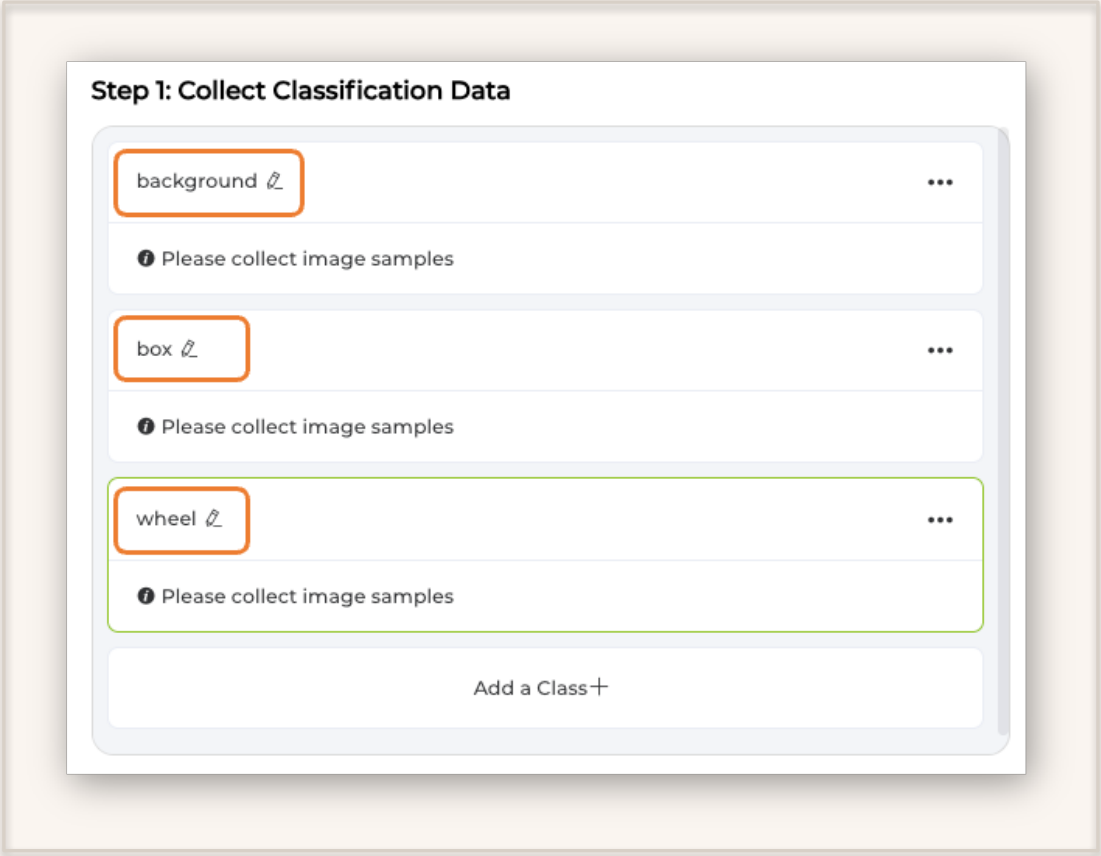
You must teach the model what nothing looks like



Without a background class, every empty frame is forced into box or wheel.

CLASSES

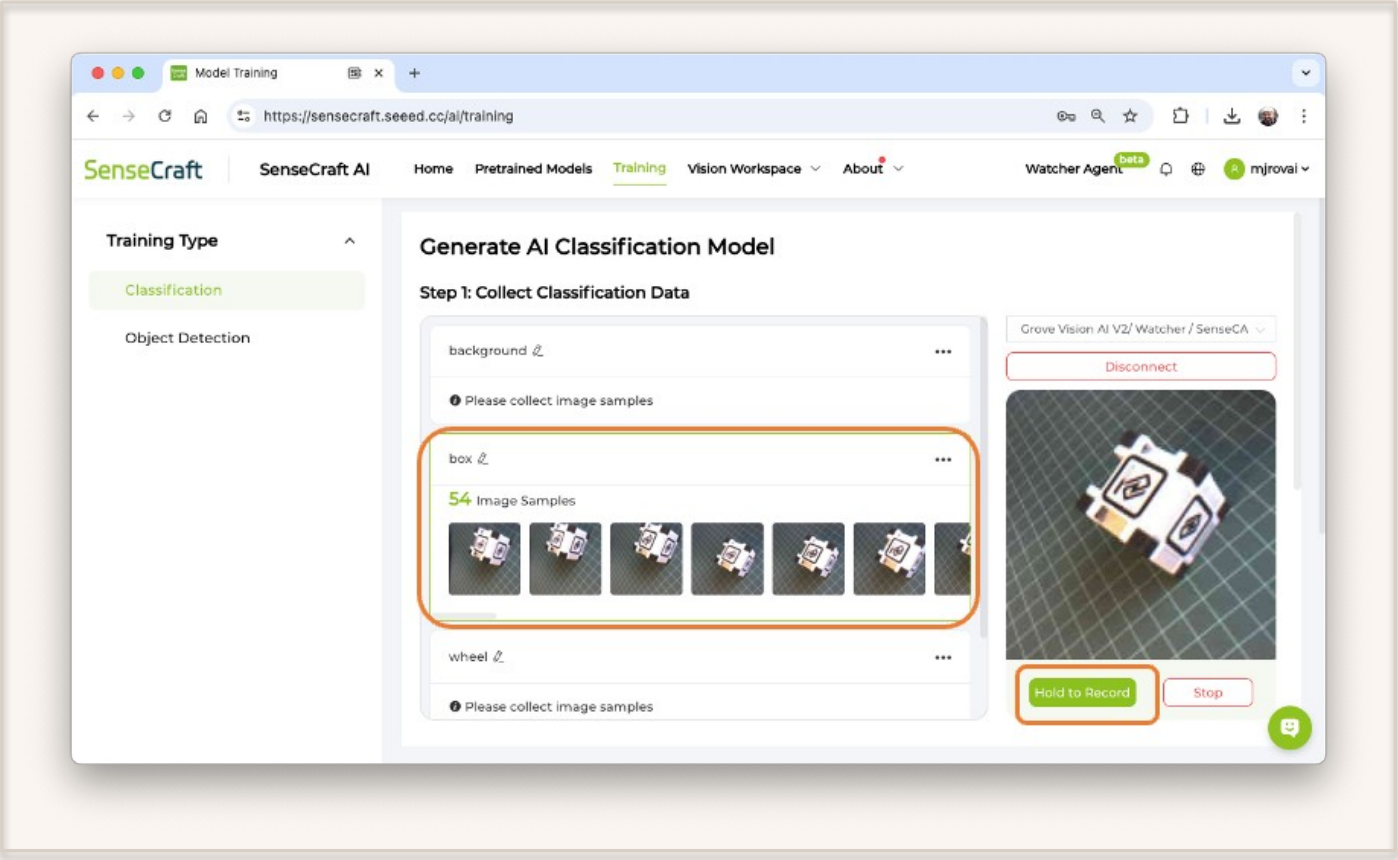
Create them in alphabetical order and keep them there



Order matters later — the model exports labels by index, and a swapped label is a silent bug.

DATA COLLECTION

Hold to record, and move the camera slowly



1. Select a class and hold the green button to record frames into it
2. The samples, and their count, appear under the class
3. Release, rearrange the object, hold again — you control the variety

CAPTURE RULES

Three rules that decide your accuracy

01

About fifty per class

More is better, but variety beats quantity — fifty varied frames beat two hundred identical ones.

02

Move slowly, review after

Blurred frames teach blur. Review the samples and delete the bad ones before training.

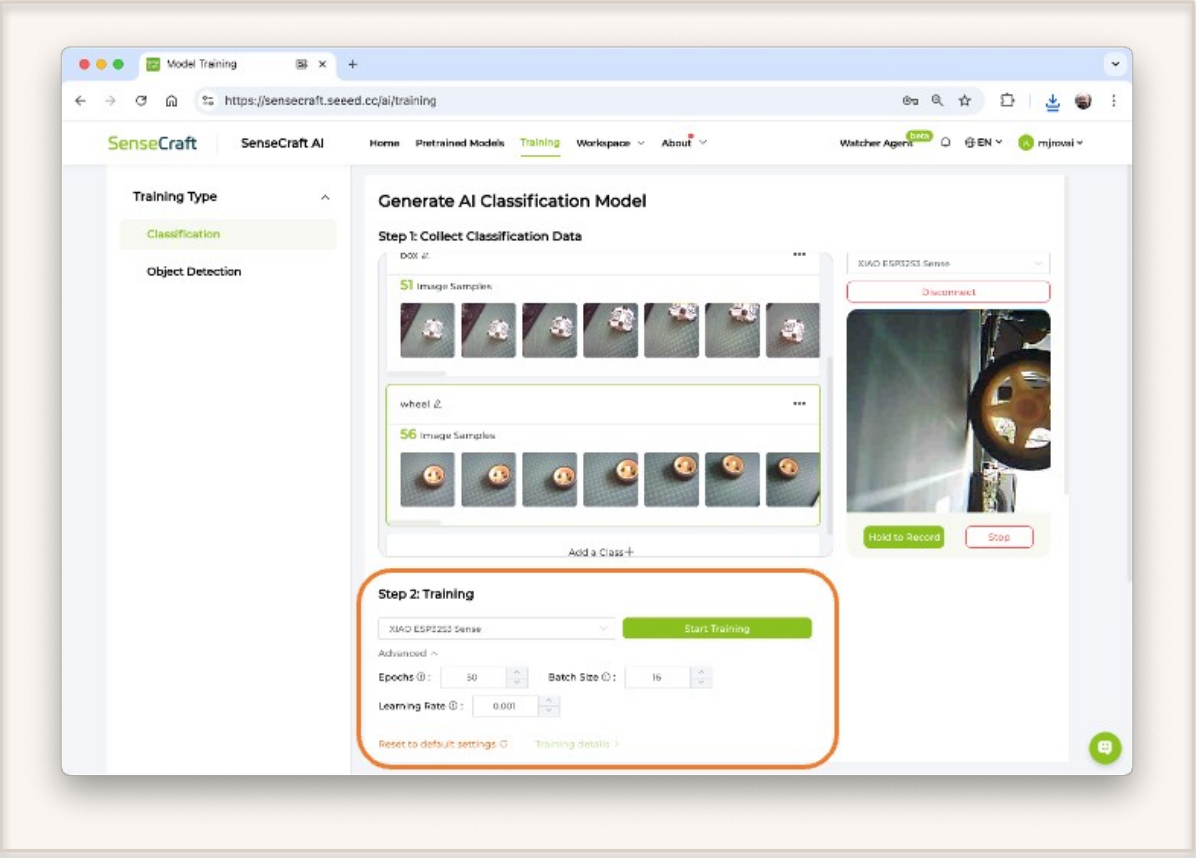
03

Film the background on purpose

The empty table, hands, shadows — in production, that is the class the model sees most.

TRAINING

Confirm the device, then train



The result is an int8 MobileNet V2 with a 224 by 224 input, about 320 KB.

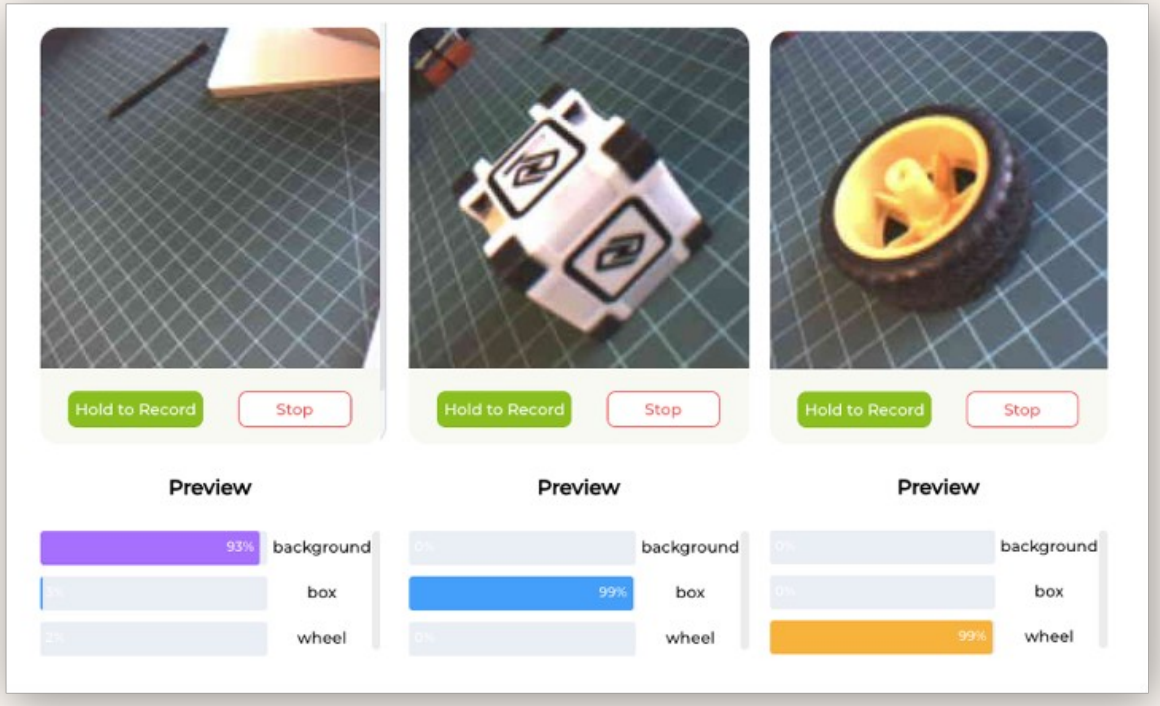
READ THE FINE PRINT

That preview is not running on your board.

The XIAO only captures the frames. The freshly trained model runs in the studio.

TEST, THEN DEPLOY

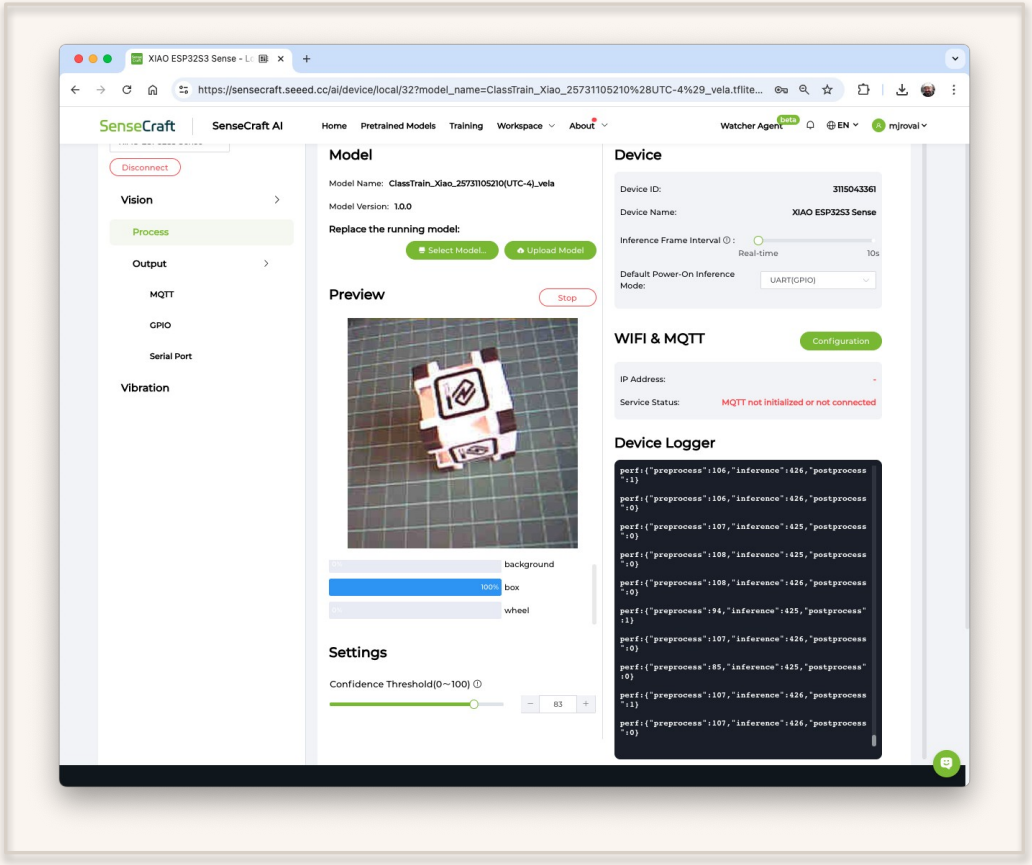
From preview to firmware



The preview proves the model. Deploy to device puts it on the board.

NUMBERS ON THE DEVICE

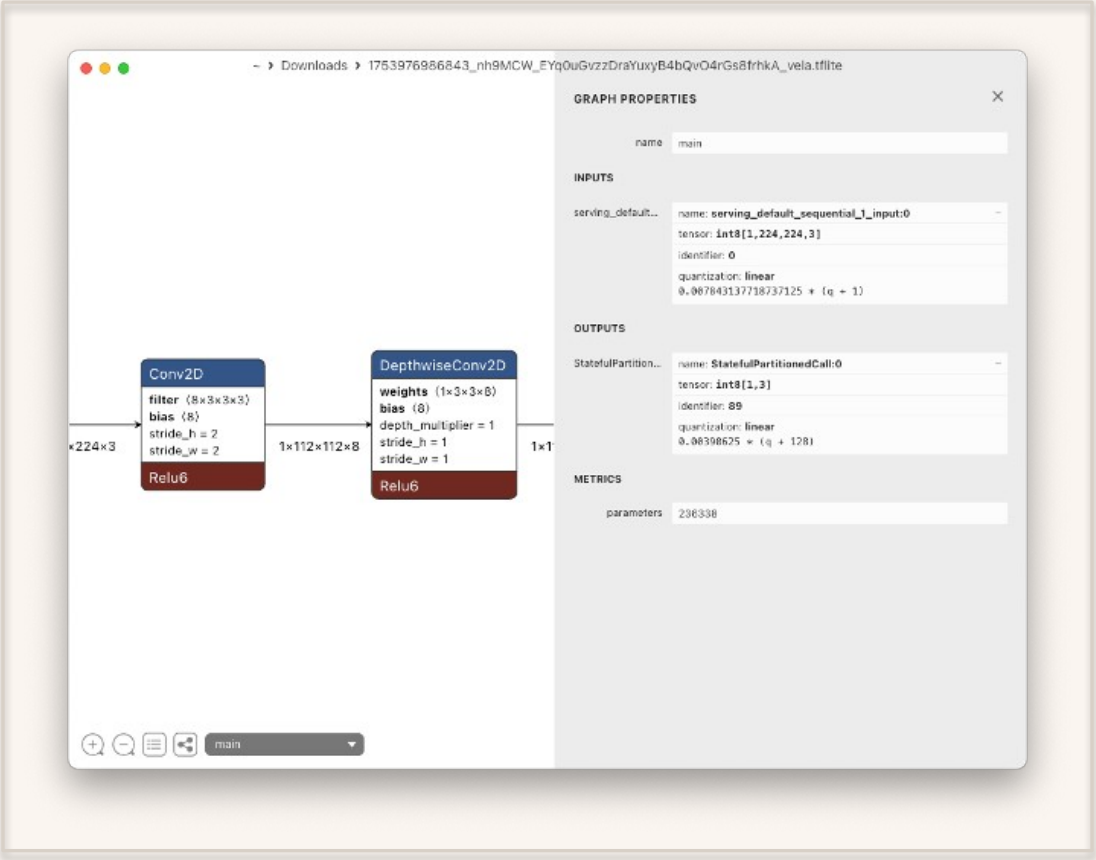
110 ms of pre-processing, 426 ms of inference



1. Model deployed: the int8 MobileNet V2, 224 by 224
2. Pre-processing about 110 ms, inference about 426 ms
3. About 1.8 frames per second, at roughly 730 mW

WHAT YOU ACTUALLY BUILT

An int8 MobileNet V2, 320 KB, input 224 by 224 by 3



1. Input tensor: 224 by 224 by 3 — that is 150,528 values per frame
2. Depthwise and pointwise convolutions, the MobileNet signature
3. Weights quantized to int8: one byte each instead of four in float32

PART 3 OF 4 · EDGE IMPULSE

04

Full control with Edge Impulse

Your dataset, your input size, your profiler, your library.

WHAT CHANGES

Five things the studio gives you

A real dataset view

Split, balance, relabel, and see exactly what the network will see.

Transfer learning

Borrow an ImageNet backbone and retrain only the head — this is why fifty images work.

Your input size

96 instead of 224. About five times fewer pixels per frame.

A profiler

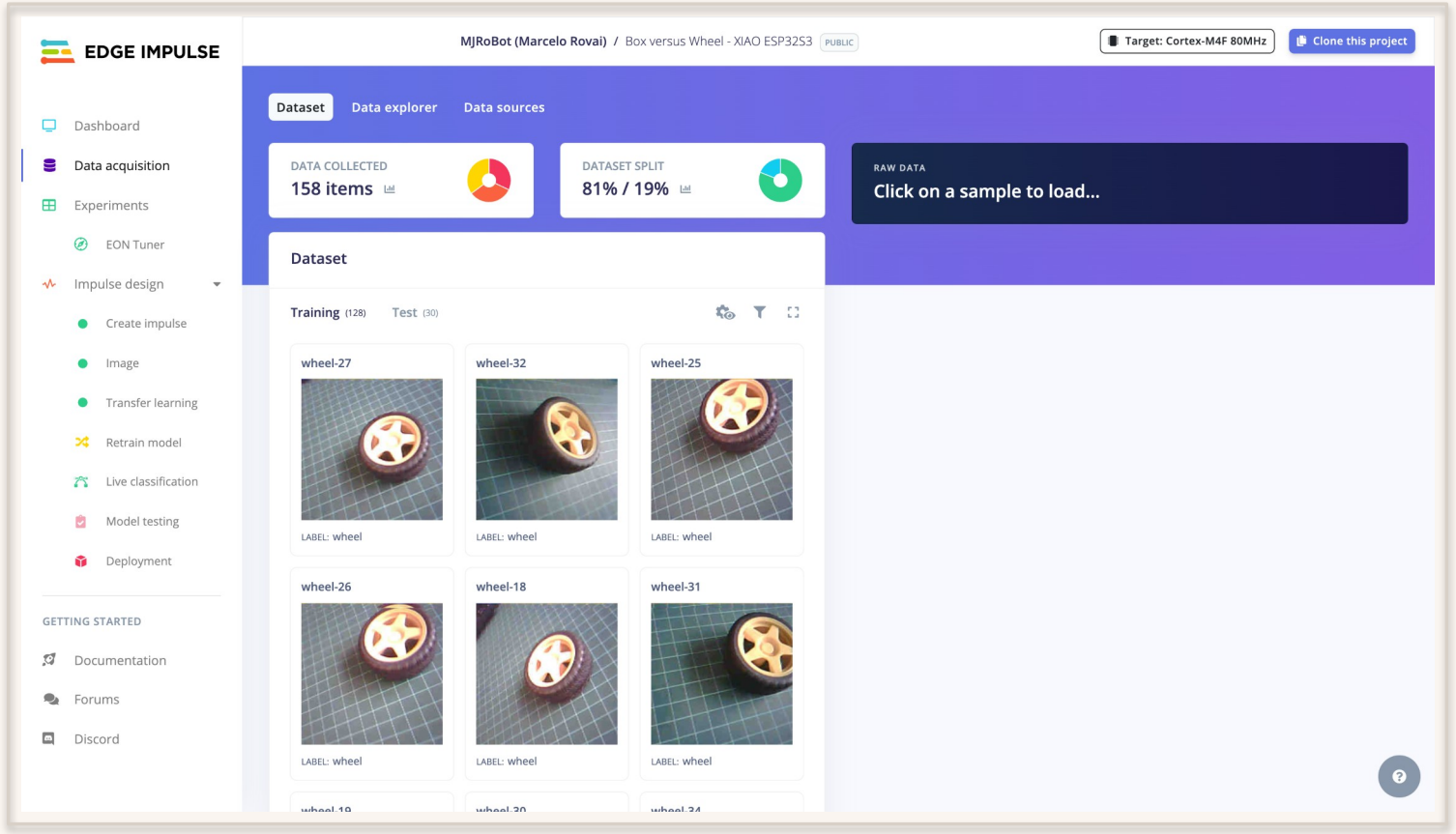
RAM, flash and latency before you flash anything.

A library

A C++ library you call from your own sketch, with your own post-processing.

DATA ACQUISITION

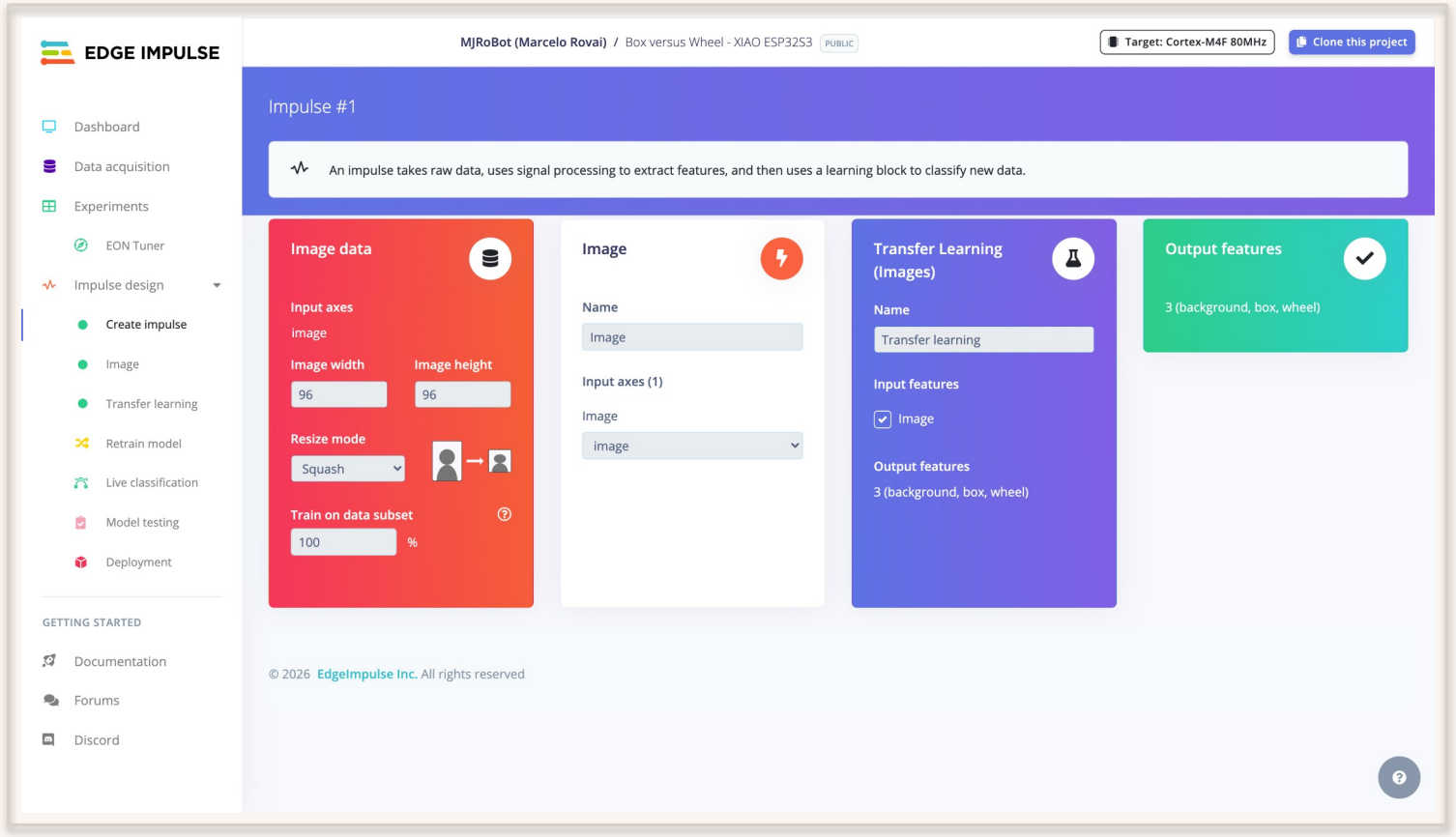
158 items, three labels, about a fifth held back



1. 158 items across background, box and wheel
2. 81 percent training, 19 percent testing — the split is automatic
3. Click a sample to confirm the label before you train on it

IMPULSE DESIGN

An impulse is the pipeline, drawn



1. Image data: every image resized to 96 by 96
2. Image block: RGB color, 27,648 features per sample
3. Transfer learning block: three output classes

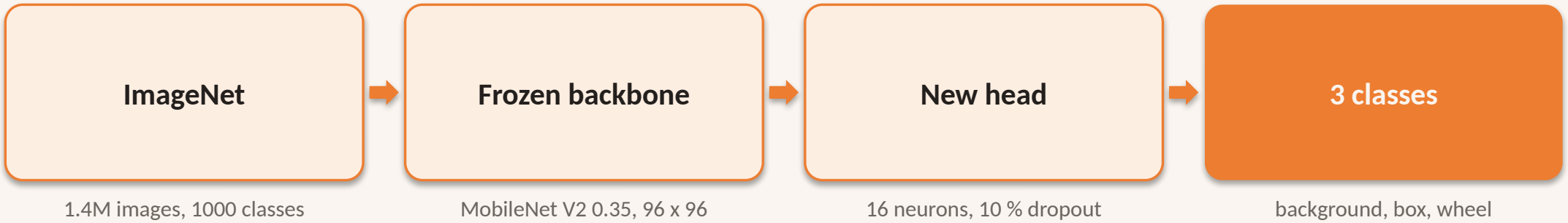
WHY FIFTY IMAGES ARE ENOUGH

You don't train a vision network from scratch. You borrow one and retrain the head.

The ImageNet backbone already knows edges, corners and textures. Your fifty images only teach the new head.

TRANSFER LEARNING

Frozen backbone, new head

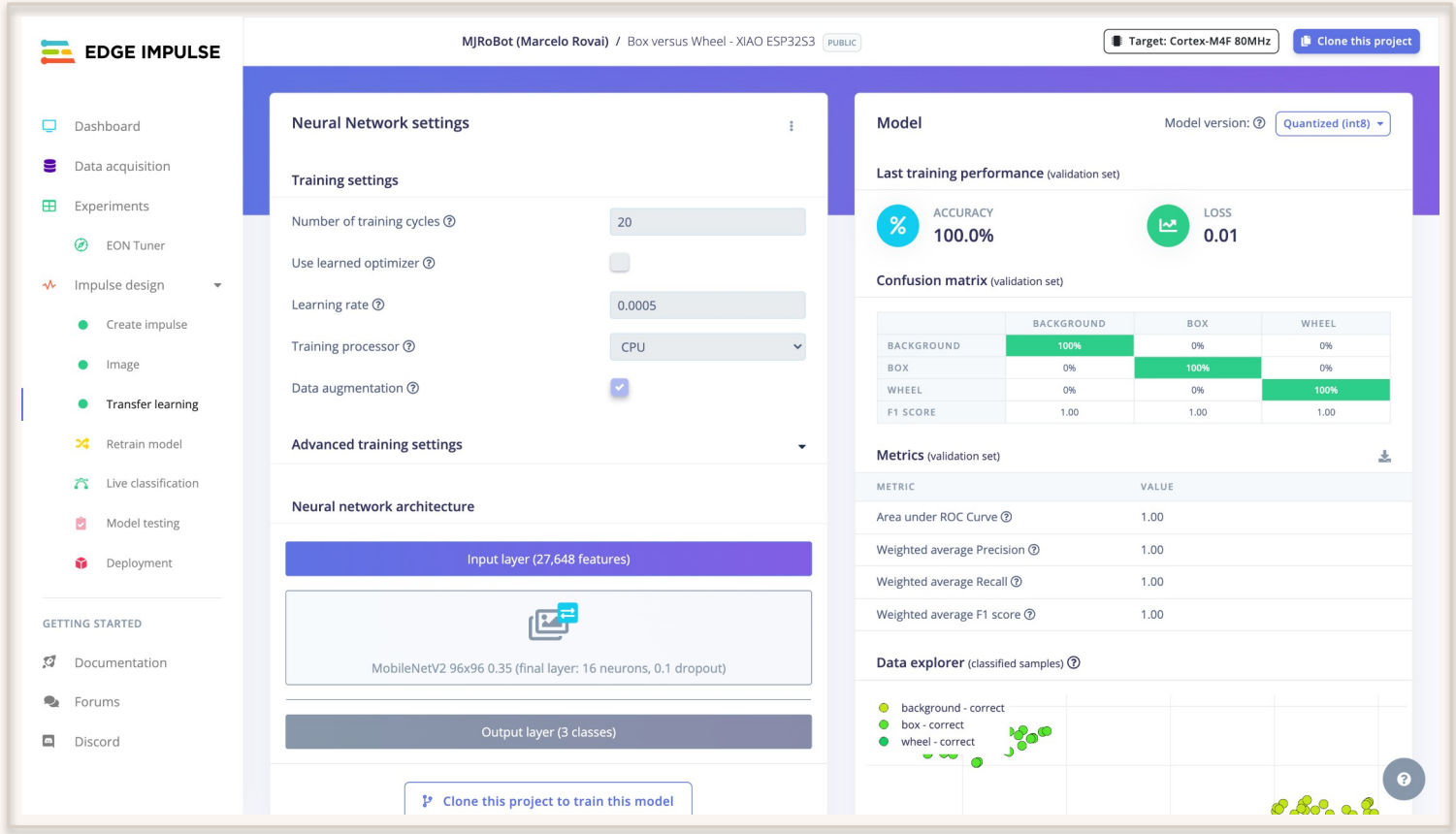


The backbone stays in the model — you pay for its weights in flash, and you get its knowledge.

This is also why the model needs about 546 KB of flash. Knowledge has a size.

THE RUN YOU ARE LOOKING AT

20 epochs, batch 32, and 100 % on validation



MobileNetV2 96x96 0.35 — 27,648 input features, 3 output classes. Validation loss 0.01, F1 1.00.

1. Training cycles 20, batch 32, learning rate 0.0005, augmentation on
2. Accuracy 100 percent, loss 0.01 — on the validation set
3. Confusion matrix is diagonal: no class is confused with another

THE COST OF RESOLUTION

MobileNet's alpha knob, and what it costs

CONFIGURATION	INPUT	RAM	FLASH
MobileNet V1 · alpha 0.10	96x96	53 KB	101 KB
MobileNet V2 · alpha 0.35	96x96	233 KB	546 KB
MobileNet V2 · alpha 1.00	160x160	1.3 MB	2.6 MB

Our choice sits in the middle. All three fit in 8 MB of PSRAM. What grows with them is latency.

WHY INT8

Quantization, with numbers from the project

MODEL	LATENCY	RAM	FLASH
Quantized int8	1,400 ms	353 KB	684 KB
Float32	1,564 ms	1.1 MB	1.7 MB

About 10 % faster, three times less RAM, two and a half times less flash. That is what int8 buys.

AUGMENTATION

Small random changes, so each image counts more

Edge Impulse augmentation policy (excerpt)

```
def augment_image(image, label):
    # flips the image randomly
    image = tf.image.random_flip_left_right(image)
    # zooms in, then randomly crops back to the input size
    resize_factor = random.uniform(1, 1.2)
    new_height = math.floor(resize_factor * INPUT_SHAPE[0])
    new_width  = math.floor(resize_factor * INPUT_SHAPE[1])
    image = tf.image.resize_with_crop_or_pad(image, new_height,
                                             new_width)
    image = tf.image.random_crop(image, size=INPUT_SHAPE)
    # varies brightness
    image = tf.image.random_brightness(image, max_delta=0.2)
    return image, label
```

— Flip, zoom, crop, brightness — small random changes on every pass.

— It runs during training only; inference sees the plain resized frame.

— The network rarely sees exactly the same image twice.

READ THE PROFILER

1,393 milliseconds. For a Cortex-M4F at 80 MHz — not for your board.

The profiler answers "would this fit somewhere", not "how fast is my device". Measure on the device.

PART 4 OF 4 · DEPLOY

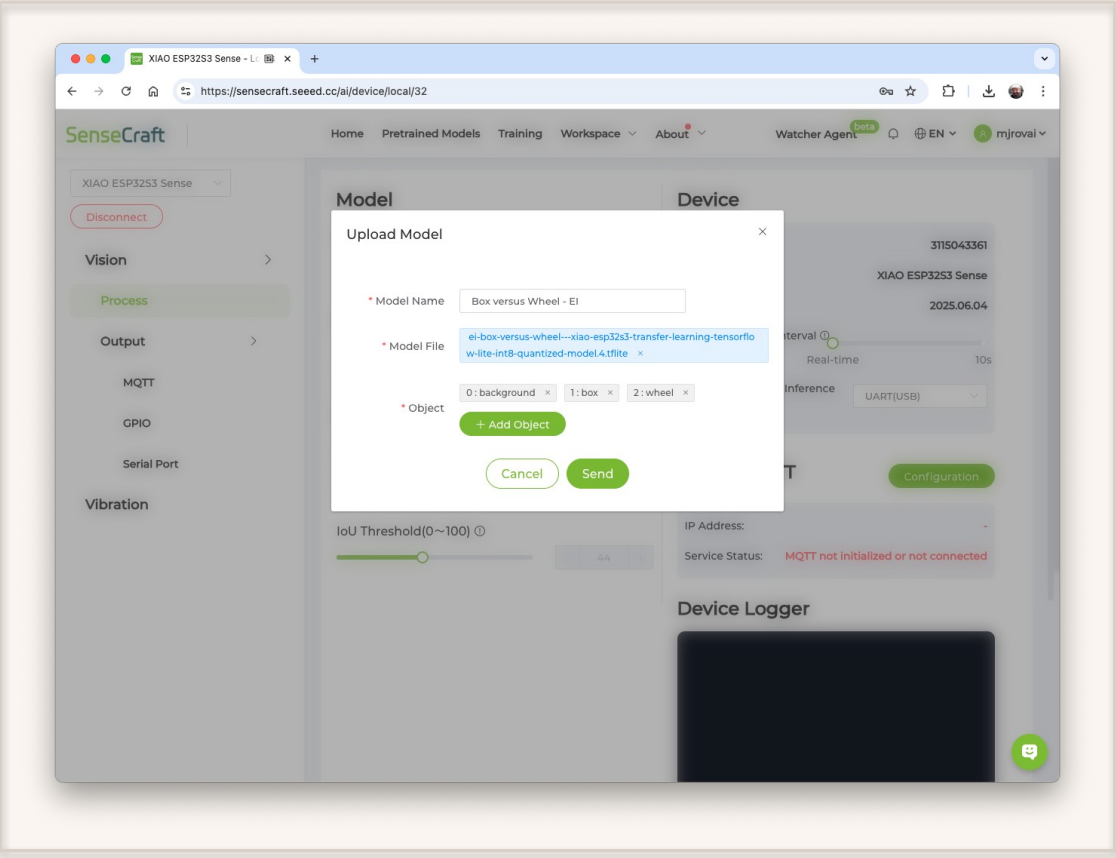
05

Deploy, run, and see it

A TFLite file, an Arduino library, and the real numbers.

BACK TO SENSECRAFT

Upload the TFLite model, labels in index order



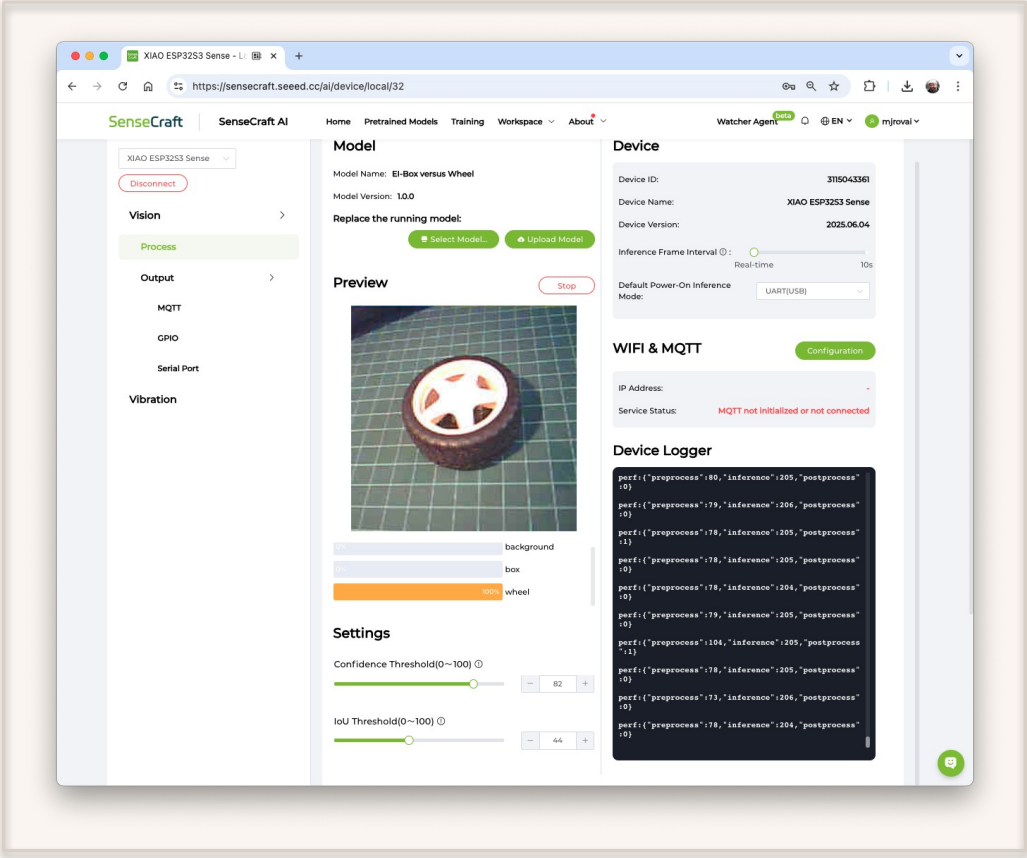
1. Download TensorFlow Lite (int8 quantized) from the Edge Impulse dashboard

2. In the workspace, Upload Model, then pick the .tflite file

3. Objects in index order: 0 background, 1 box, 2 wheel

NUMBERS ON THE DEVICE

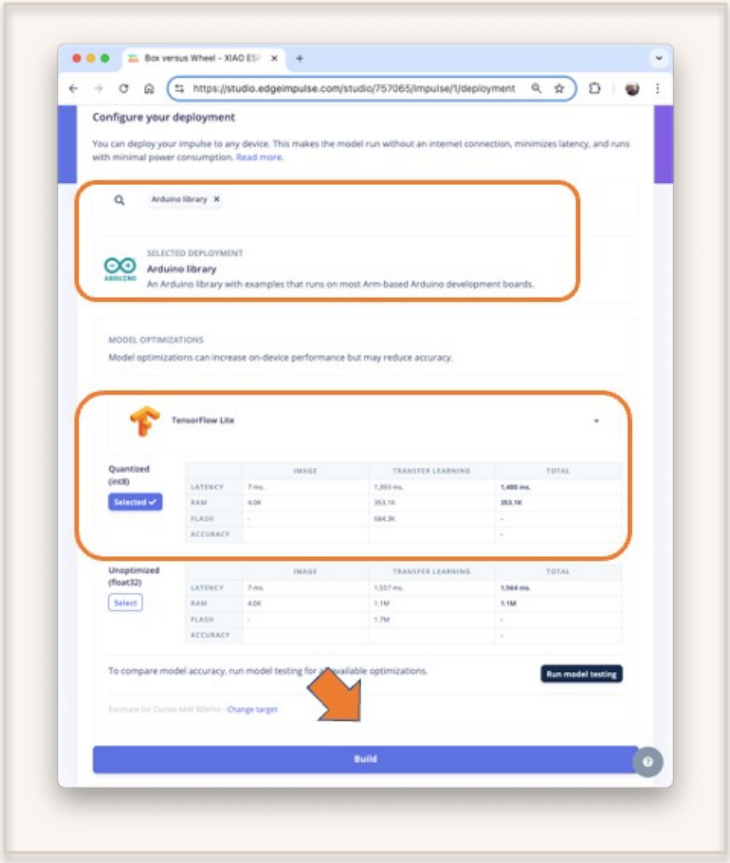
96 by 96 doubles the frame rate



1. The 96 by 96 model, running on the XIAO
2. About 81 ms of pre-processing and 205 ms of inference
3. About 3.4 frames per second, twice the 224 by 224 model

BUILD THE LIBRARY

Arduino library, TensorFlow Lite, quantized int8



1. Deployment target: Arduino library, so the model lives inside your sketch
2. Engine: TensorFlow Lite, as in the book
3. Model: quantized int8, then Build — a .zip lands in your downloads

WHAT THE SKETCH DOES

Camera to serial, in five steps



Every millisecond of that chain is on your bill — the JPEG decode is not free.

THE LOOP

Allocate, capture, classify, print

image_class_XIAOML-Kit.ino (simplified)

```
void loop() {
  // 1. buffer in PSRAM, fall back to heap
  snapshot_buf = (uint8_t*)ps_malloc(W * H * BPP);

  // 2. capture and convert to RGB888
  if (!ei_camera_capture(INPUT_W, INPUT_H, snapshot_buf)) {
    free(snapshot_buf); return;
  }

  // 3. run the impulse
  signal_t signal;
  ei_impulse_result_t result = { 0 };
  run_classifier(&signal, &result, false);

  // 4. print every class
  for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
    ei_printf("  %s: %.5f\r\n",
              ei_classifier_inferencing_categories[i],
              result.classification[i].value);
  }
}
```

— ps_malloc puts the frame buffer in the 8 MB PSRAM, not in the small internal RAM.

— run_classifier blocks. Nothing else happens during inference — plan the loop around that.

BEFORE YOU UPLOAD

Two settings, and one rescue

PSRAM must be on

Tools, PSRAM, OPI PSRAM. Without it, there is no room for the camera frame buffer.

ESP32 core 2.0.17

The sketch was tested with version 2.0.17 of the esp32 boards package. Do not update it.

If uploads fail

Hold BOOT, tap RESET, release BOOT. Then upload again — this fixes most bad flashes.

MEASURED ON THE BOARD

4 ms of DSP, 199 ms of classification

```
Predictions (DSP: 4 ms., Classification: 199 ms., Anomaly: 0 ms.):
Predictions:
  background: 0.64844
  box: 0.28516
  wheel: 0.06641
Predictions (DSP: 4 ms., Classification: 199 ms., Anomaly: 0 ms.):
Predictions:
  background: 0.38281
  box: 0.60547
  wheel: 0.01172
```

DSP

4 ms

INFERENCE

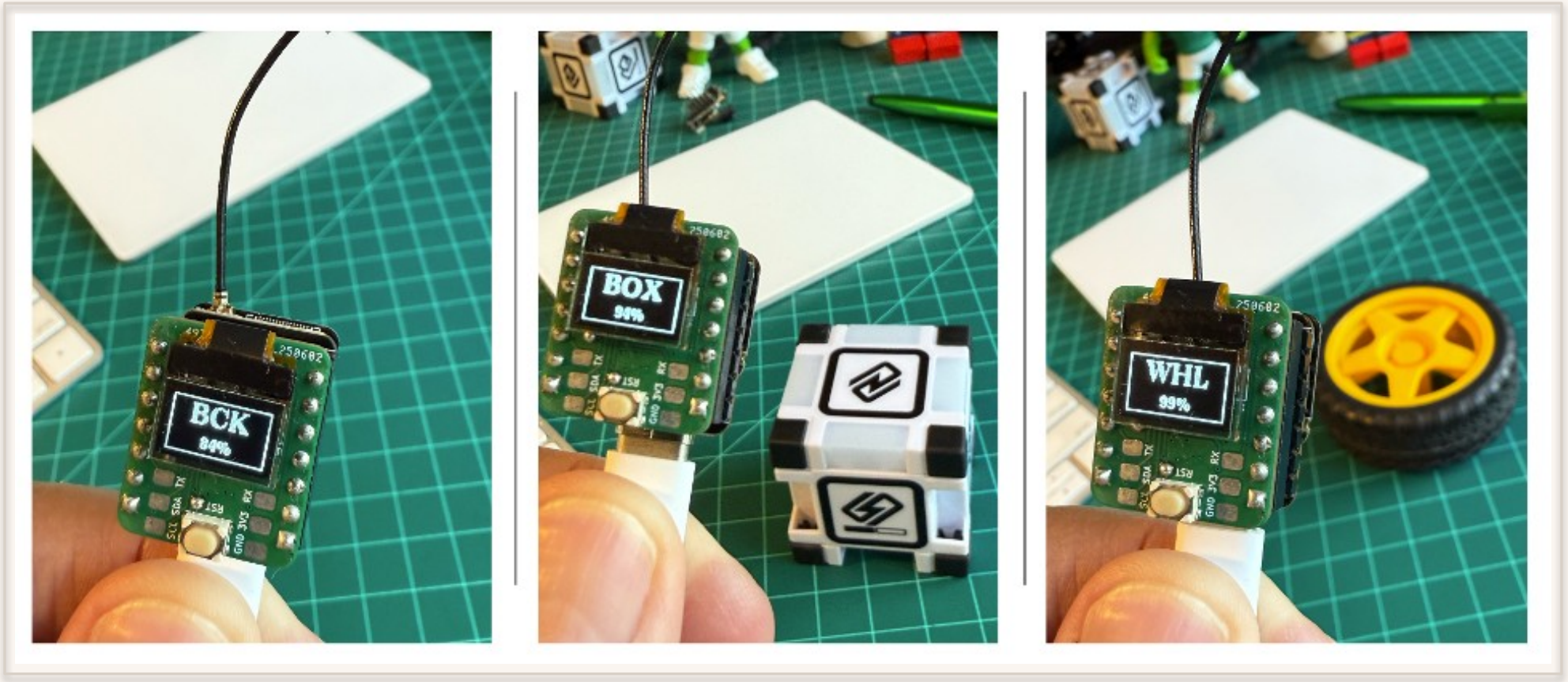
199 ms

ON THE WIRE

2.9 FPS

POST-PROCESSING

The answer on a 72 by 40 pixel OLED



Three letters and a confidence, read straight from the model's label table.

SUMMARY

Three things to remember

01 Resolution is the budget

224 by 224 buys detail; 96 by 96 buys frame rate. About five times fewer pixels, and twice the FPS.

02 Transfer learning is why fifty images work

You borrow a backbone and retrain a head. You don't train vision from scratch on a microcontroller.

03 Latency is not throughput

203 ms of model time gave 4.9 FPS on paper and 2.9 on the wire. Measure the loop, not the model.

CHECK YOURSELF

Your serial monitor reports 199 ms of classification. What do you actually get on the wire?

A About 5 frames per second

B About 2.9 frames per second — the loop adds capture, decode and printing

C Exactly 5.03 frames per second

The model is one term in the loop. Camera capture, JPEG decode, resize and the serial print all fit inside the same period.

The interactive deck gives per-answer feedback.

Questions?

Next — Object Detection, where the answer is not one label but a box per object.

