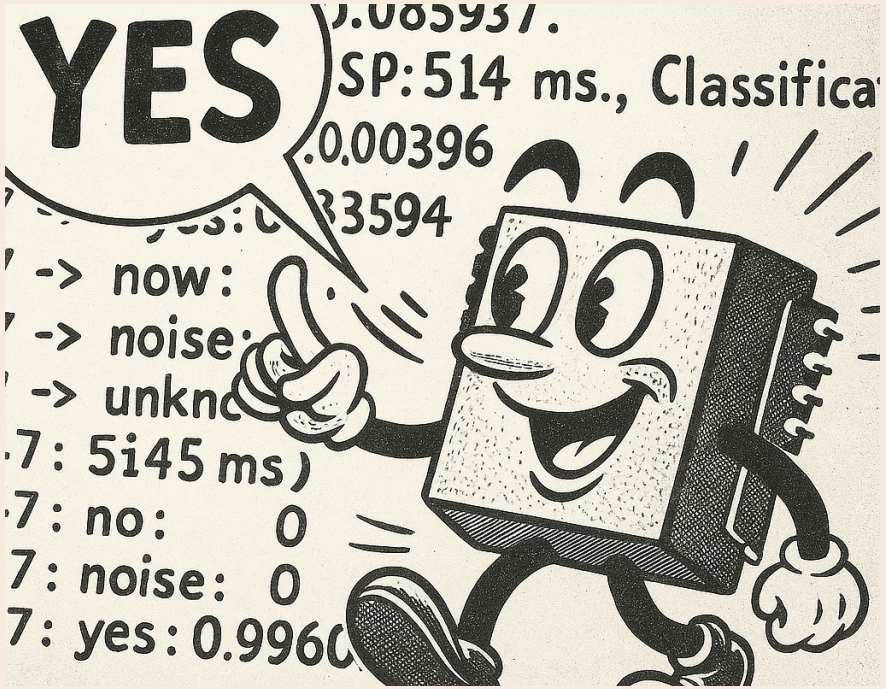
CHAPTER 4 · TINYML ON SEEED

Keyword Spotting

A microphone, one second of sound,
and a model that listens for one word

AUDIO · MFCC · EDGE IMPULSE

49 slides · 1 lab · XIAO ESP32S3 Sense



AIEng4D

WHERE WE ARE · PART I

01

From pixels to sound

The camera gave us images. The microphone gives us 16,000 numbers per second, and we will turn them into an image too.

LEARNING OBJECTIVES

What you will be able to do

01

Explain where KWS fits

A tiny model that wakes up a bigger one. Stage 1 of a cascade, always on, always local.

02

Capture audio on the board

The PDM microphone, the I2S bus, 16 kHz at 16 bits, and WAV files on the SD card.

03

Turn sound into features

One second of audio becomes a 13 × 49 MFCC image, then a small 1D CNN classifies it.

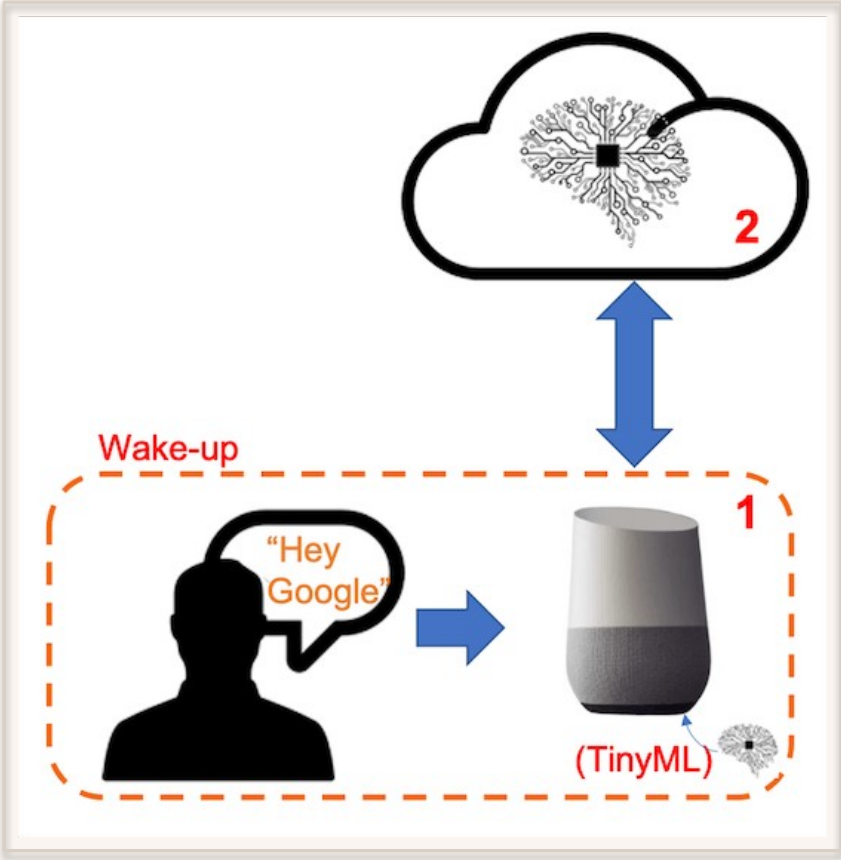
04

Deploy and act on the result

Run it on the XIAO, read the timing, and light an LED or the OLED when it hears "yes".

HOW A VOICE ASSISTANT WAKES UP

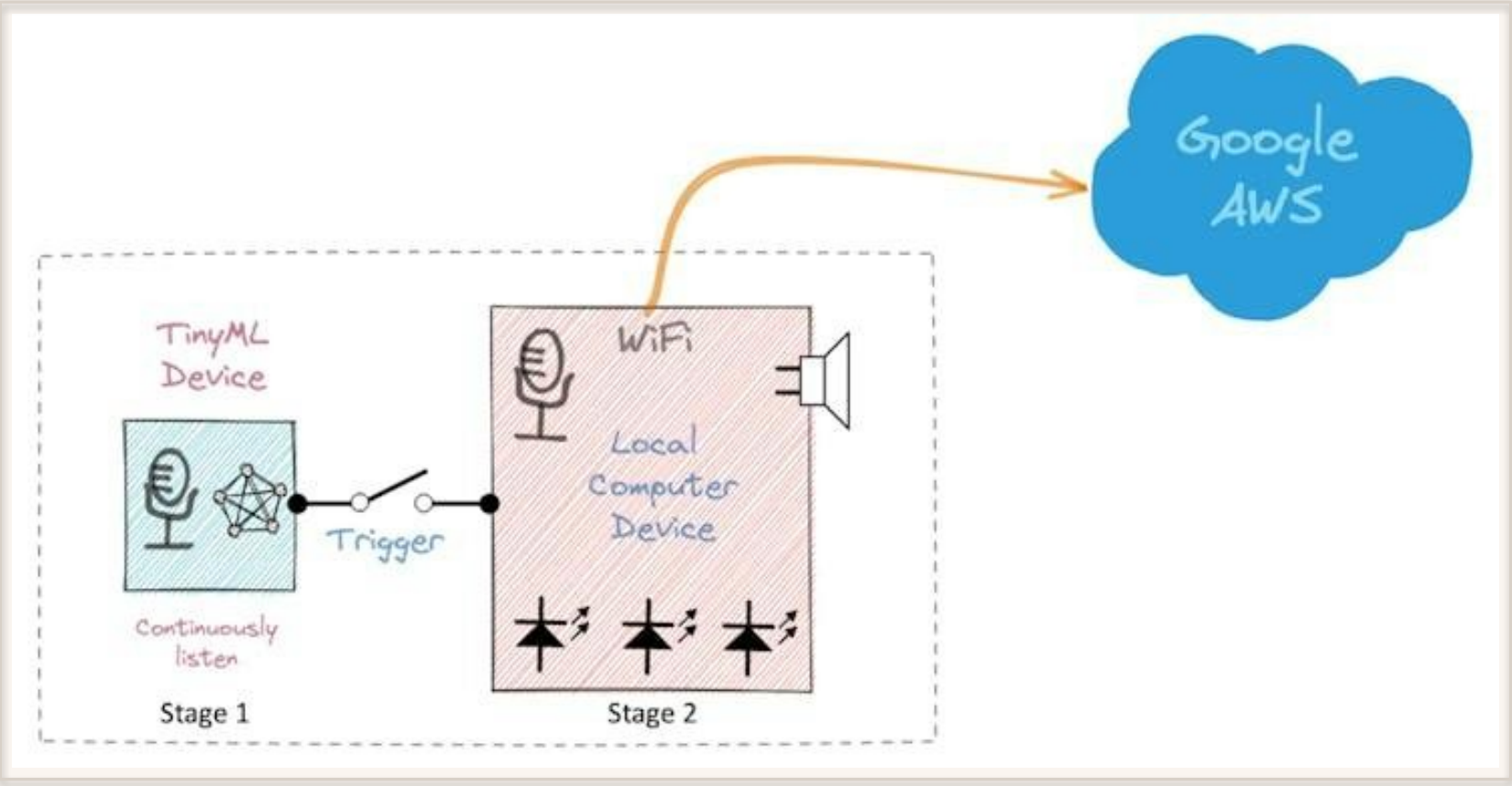
"Hey Google" is spotted on the device, not in the cloud



1. Stage 1 — a small processor inside the speaker listens all the time
2. Stage 2 — only after the wake word does audio go to a large model in the cloud
3. The wake word is the trigger between the two

CASCADE DETECTION

A tiny model that listens, a big model that understands



1. Stage 1: a TinyML device listens continuously for the keyword
2. The keyword closes the switch and triggers stage 2
3. Stage 2: a local computer or the cloud handles the full request
4. This chapter builds stage 1 on the XIAO

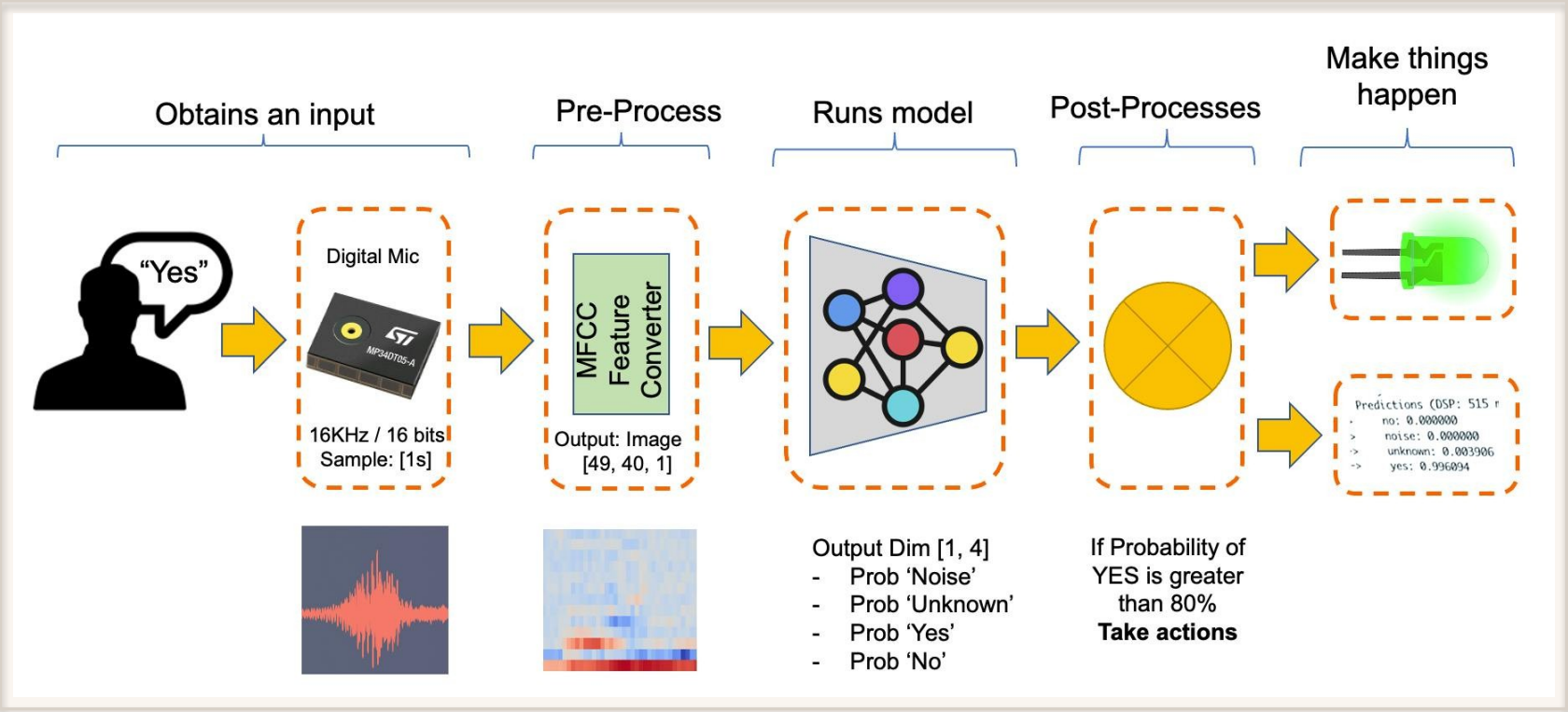
WHY STAGE 1 MUST BE TINY

It never stops listening. So it must be small, cheap in energy, and private — nothing leaves the board until the keyword is heard.

That is exactly the TinyML profile — always on, milliwatts, and local.

THE INFERENCE PIPELINE

Listen, pre-process, classify, then act



1. The digital microphone samples at 16 kHz, 16 bits, one second at a time
2. MFCC turns that second into a small image, 13 × 49
3. The model outputs four probabilities — noise, unknown, yes, no
4. If yes is above 80 %, the board takes an action

FOUR CLASSES, NOT TWO

Two keywords, and two classes for everything else

01

YES

Keyword 1 — the one that triggers the action.

02

NO

Keyword 2 — a second word the model must tell apart.

03

NOISE

No one speaking. Only the background of the room.

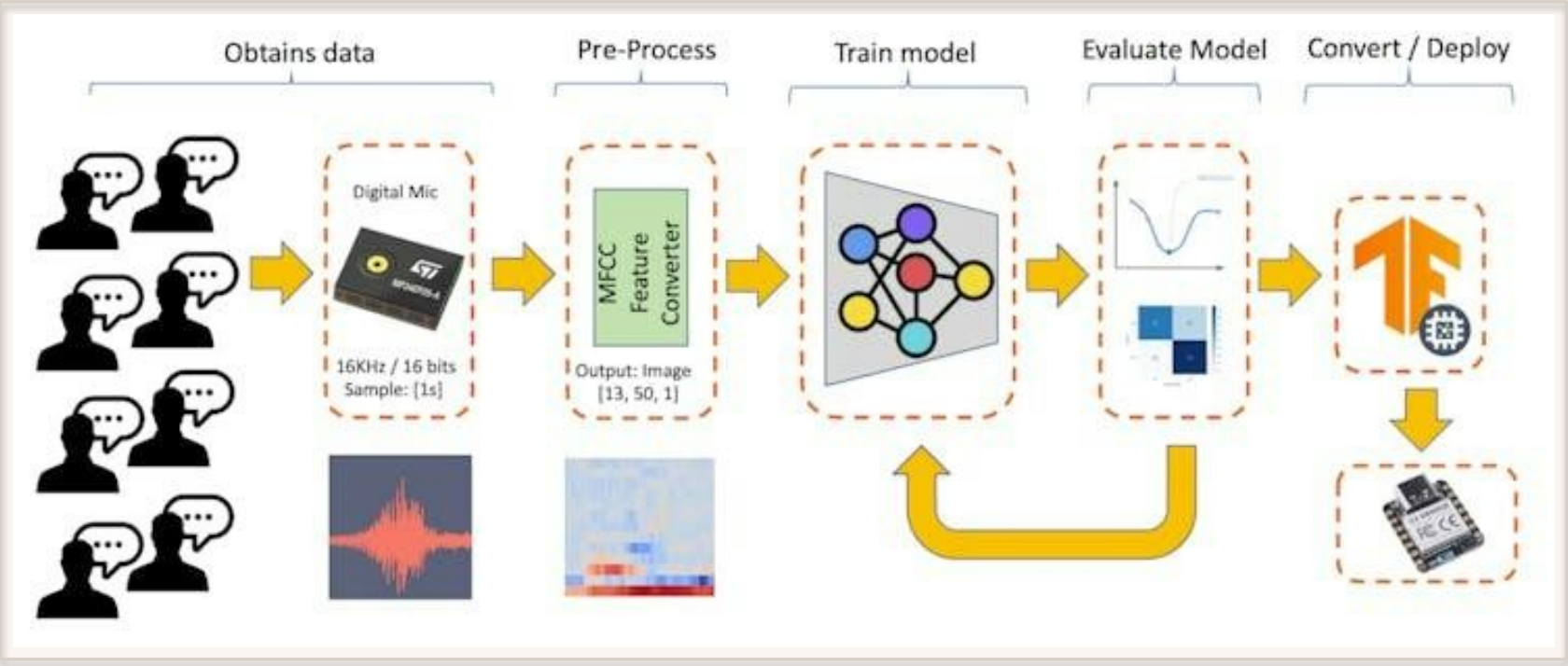
04

UNKNOWN

Other words. Without it, every word would be forced into yes or no.

THE MACHINE LEARNING WORKFLOW

Many voices in, one small model out



1. Collect many speakers saying each word
2. Pre-process every sample into MFCC features
3. Train and evaluate — and loop back when needed
4. Convert and deploy to the device

02

AUDIO DATA

Sound becomes numbers

A public dataset for the volume, and your own voice, recorded by the board, for the real conditions.

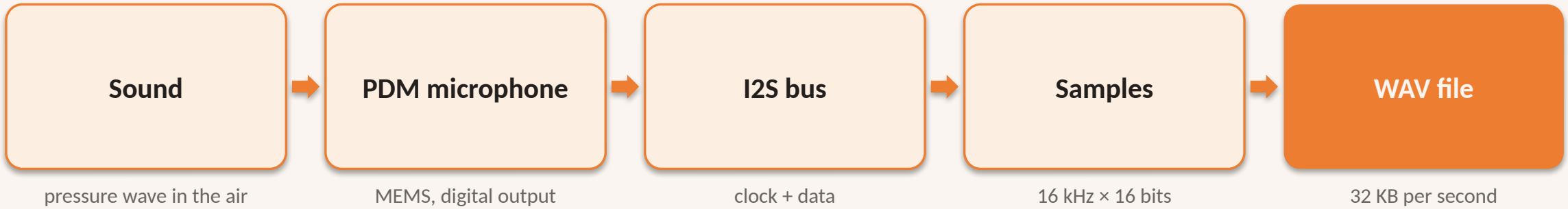
SPEECH COMMANDS · PETE WARDEN, 2018

35 words, more than 1,000 one-second samples each, recorded by thousands of different people.

Edge Impulse offers a subset with yes, no, noise, and unknown — keywords2.zip.

SOUND VERSUS AUDIO

From a pressure wave to a WAV file



Any device that records 16 kHz, 16-bit audio can feed this model — the board, a computer, or a phone.

THREE WAYS TO RECORD YOUR VOICE

Pick the one that fits your setup

01

SenseCraft AI

Record from the board in the browser and export 1-second .wav files. Quick, but the length is fixed.

02

XIAO + SD card

An Arduino sketch records 10-second WAV files, named by label, straight to the microSD card.

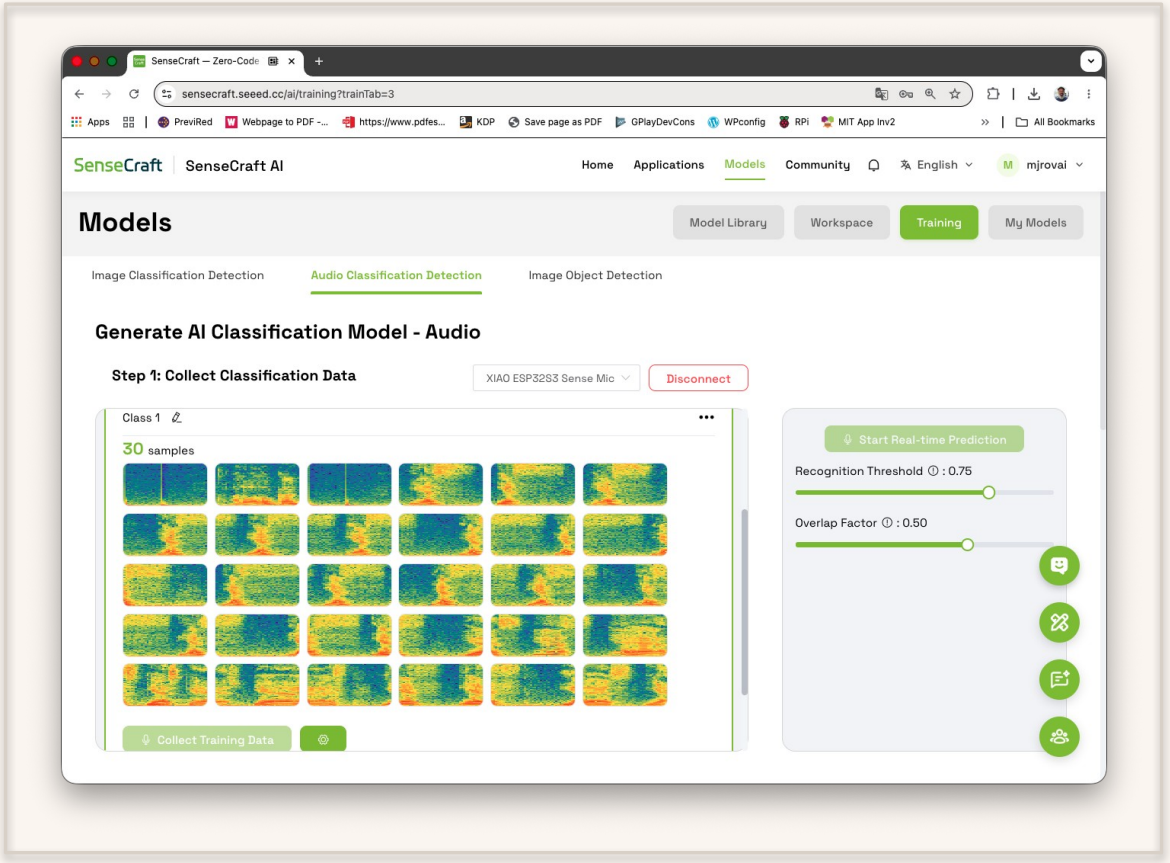
03

Phone or computer

Connect it to Edge Impulse Studio, or use a recorder app set to 16 kHz and 16 bits.

SENSECRAFT AI · TRAINING PAGE

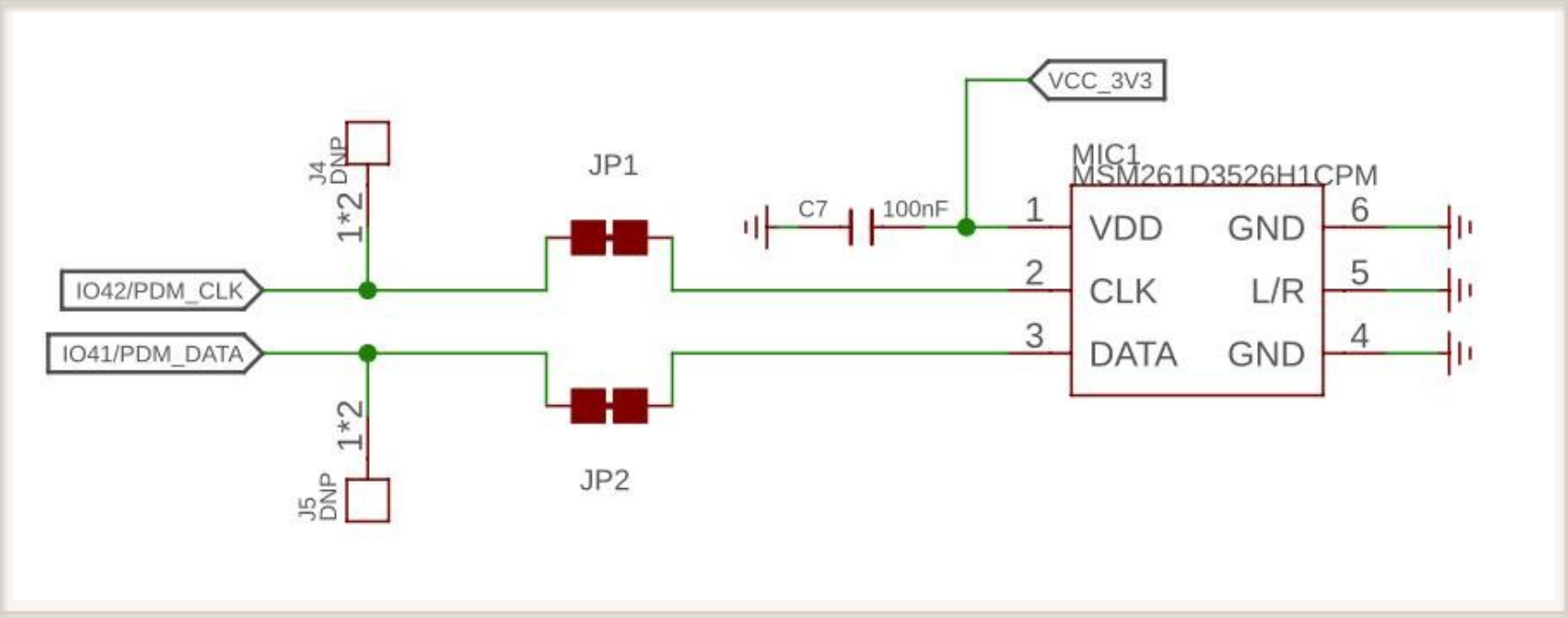
Record from the board, export one-second clips



1. Models › Training › Audio Classification Detection
2. Connect the XIAO ESP32S3 Sense Mic, then Collect Training Data
3. Recording is split into fixed 1-second samples, shown as spectrograms
4. The three-dot menu exports the clips as .wav files

THE MICROPHONE

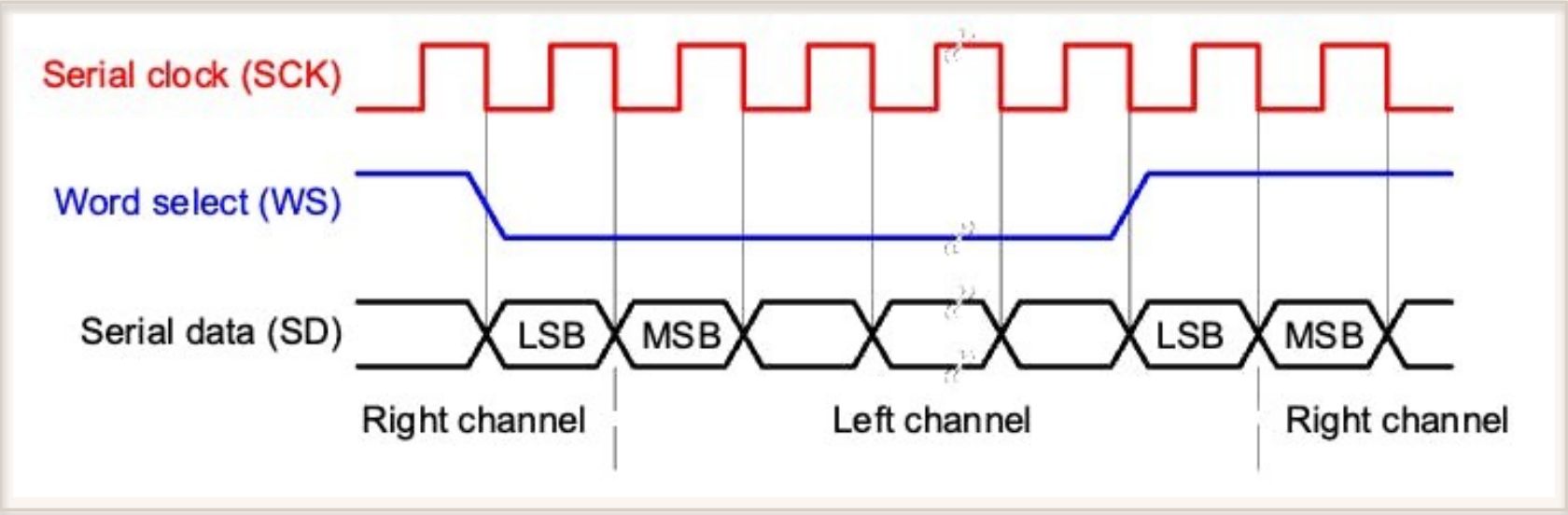
A PDM MEMS microphone on two pins



1. MSM261D3526H1CPM — a PDM, digital-output MEMS microphone
2. CLK on GPIO 42
3. DATA on GPIO 41
4. L/R tied to ground — left channel only, so mono

WHAT IS I2S?

Inter-IC Sound, a bus made for audio



1. Serial clock (SCK): one tick per bit
2. Word select (WS): left or right channel — its frequency is the sample rate
3. Serial data (SD): the audio bits, most significant first

FIRST TEST

Start I2S at 16 kHz, 16 bits, and print every sample

XiaoEsp32s3_Mic_Test.ino

```
#include <I2S.h>

void setup() {
  Serial.begin(115200);
  while (!Serial) {}
  // start I2S at 16 kHz with 16-bits per sample
  I2S.setAllPins(-1, 42, 41, -1, -1);
  if (!I2S.begin(PDM_MONO_MODE, 16000, 16)) {
    Serial.println("Failed to initialize I2S!");
    while (1);
  }
}

void loop() {
  int sample = I2S.read();
  if (sample && sample != -1 && sample != 1) {
    Serial.println(sample);
  }
}
```

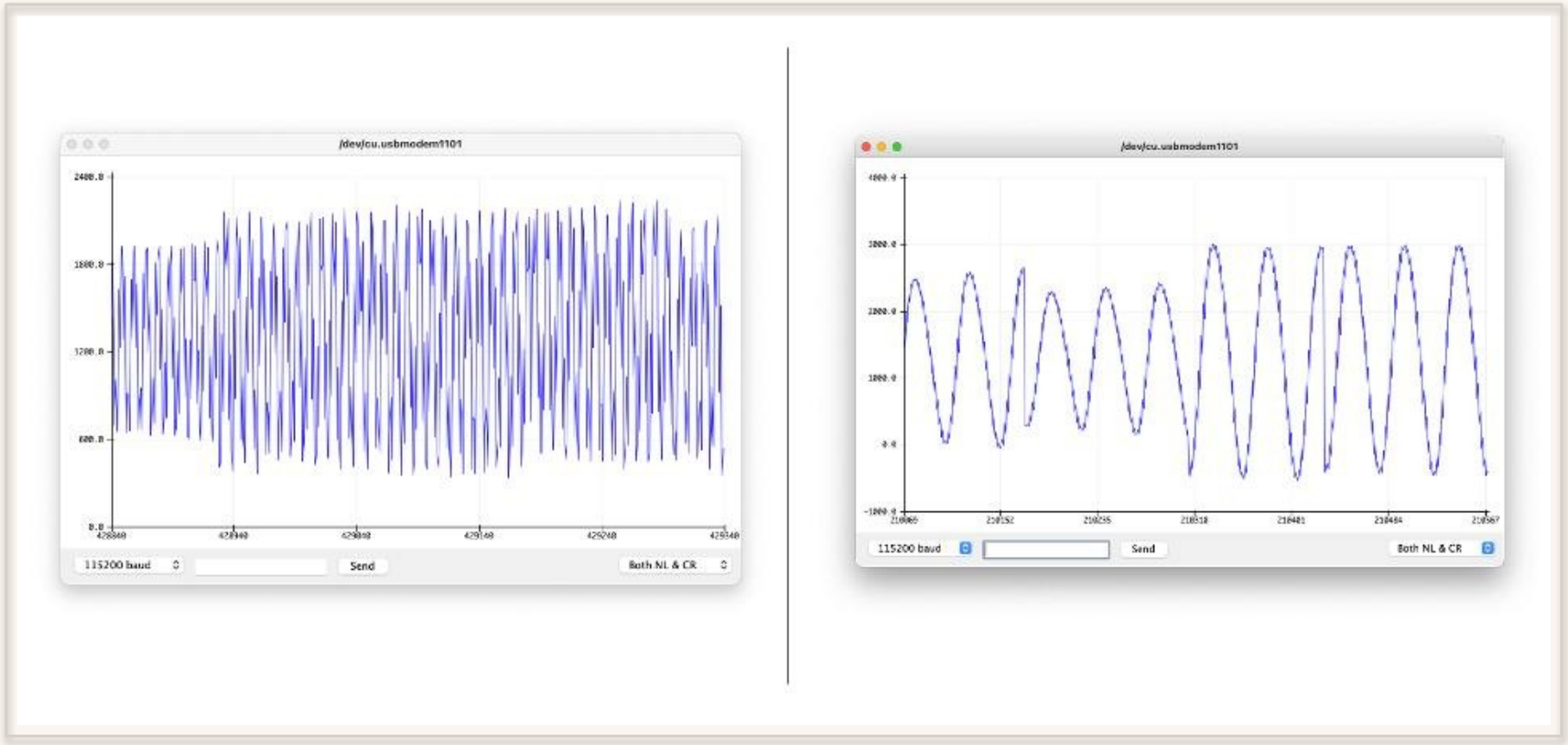
— setAllPins — clock on 42, data on 41; -1 means not used.

— PDM_MONO_MODE, 16,000 Hz, 16 bits — the format our model expects.

— Needs the esp32 core 2.0.17 and PSRAM enabled. I2S.h is gone in core 3.x.

SERIAL PLOTTER

Whispering in two tones, seen as a waveform

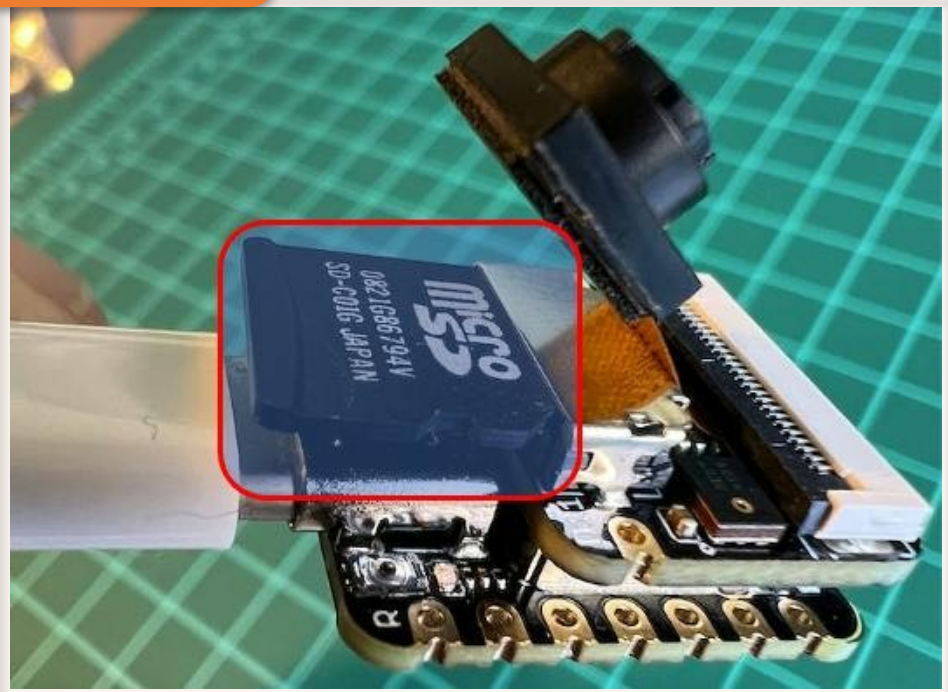


The same microphone, two whispers. A higher tone gives a faster oscillation.

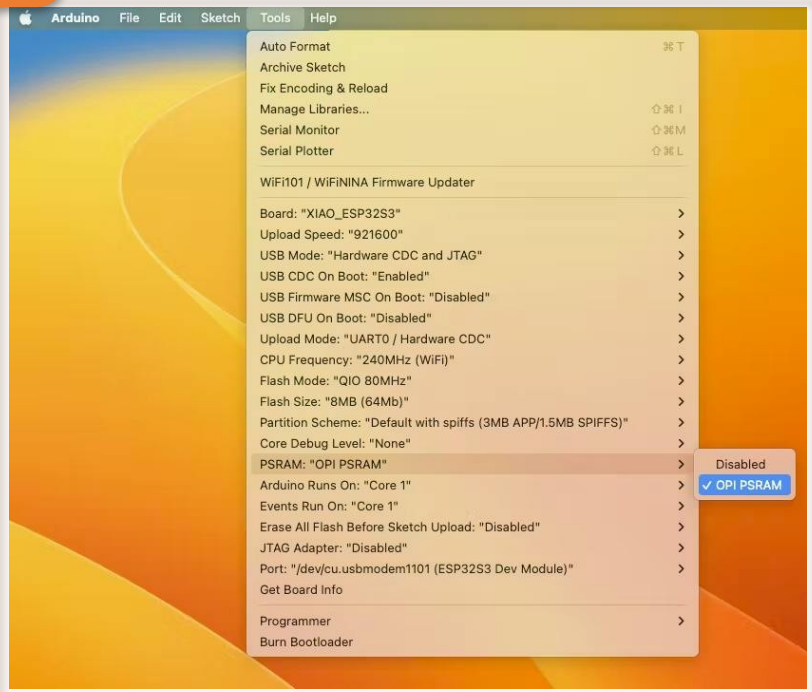
BEFORE RECORDING TO FILES

Insert a FAT32 microSD card, and enable PSRAM

microSD card



PSRAM



Tools › PSRAM › OPI PSRAM. Ten seconds of audio is 320,000 bytes — too much for internal RAM, easy for the 8 MB PSRAM.

WAV_RECORD_DATASET

Allocate in PSRAM, read, amplify, write

Wav_Record_dataset.ino — record_wav() (excerpt)

```
#define RECORD_TIME   10      // seconds
#define SAMPLE_RATE   16000U
#define SAMPLE_BITS   16
#define VOLUME_GAIN   2

rec_buffer = (uint8_t *)ps_malloc(record_size);   // PSRAM

esp_i2s::i2s_read(esp_i2s::I2S_NUM_0, rec_buffer,
                  record_size, &sample_size, portMAX_DELAY);

for (uint32_t i = 0; i < sample_size; i += SAMPLE_BITS/8) {
  (*(uint16_t *)(rec_buffer+i)) <<= VOLUME_GAIN;  // x4
}

file.write(wav_header, WAV_HEADER_SIZE);          // 44 bytes
file.write(rec_buffer, record_size);
```

— Type a label (yes) in the Serial Monitor, then rec for each sample.

— Files are saved as yes.1.wav, yes.2.wav, and so on.

— A shift left by 2 multiplies every sample by 4.

SERIAL MONITOR

Each rec gives ten seconds, and 320,000 bytes

```
Start recording ...
Buffer: 347064 bytes
Record 320000 bytes
Writing to the file ...
Recording complete:
Send rec for a new sample or enter a new label

Start recording ...
Buffer: 347064 bytes
Record 320000 bytes
Writing to the file ...
Recording complete:
```

PER FILE

10 s

SIZE

320,000 B

NAMES

no.1.wav, no.2.wav

03

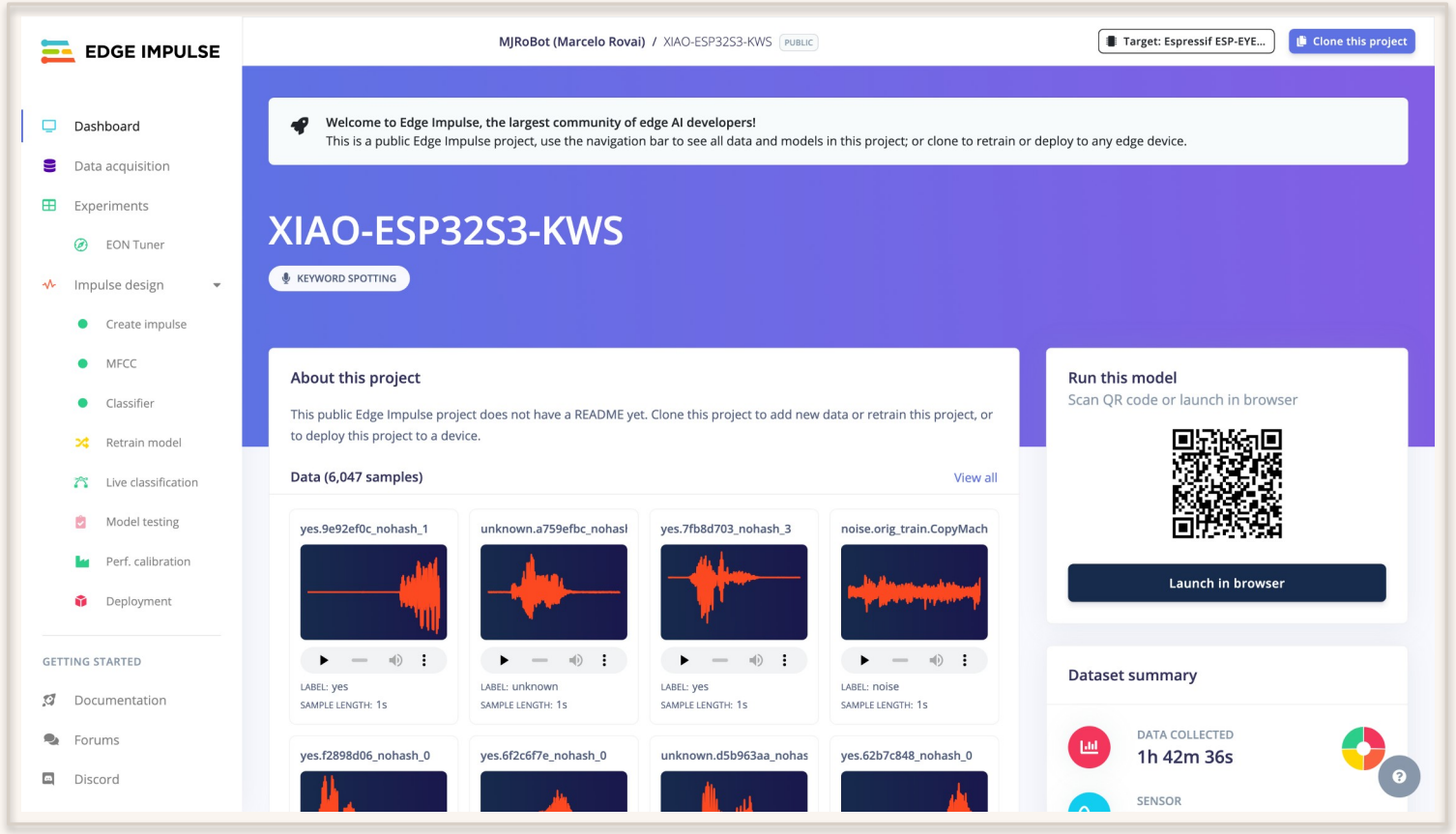
EDGE IMPULSE STUDIO

One second becomes an image

Upload, split into one-second clips, extract MFCC, and train a small 1D CNN.

THE PROJECT

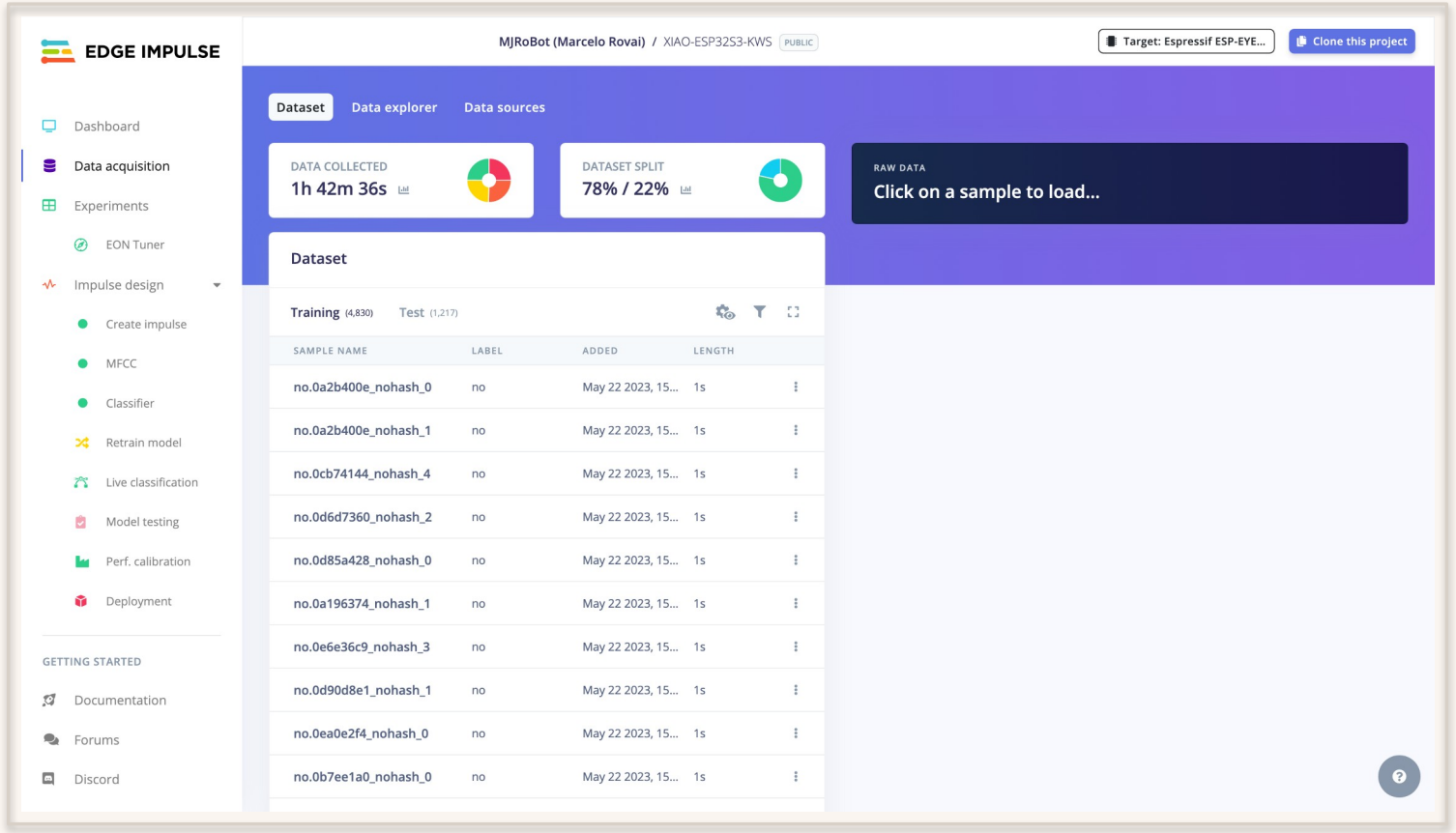
A public project you can clone, targeting the ESP-EYE



1. XIAO-ESP32S3-KWS — public project 230109
2. Target: Espressif ESP-EYE, the closest device to our board
3. 6,047 samples, each one second long
4. 1 h 42 min of audio in total

DATA ACQUISITION

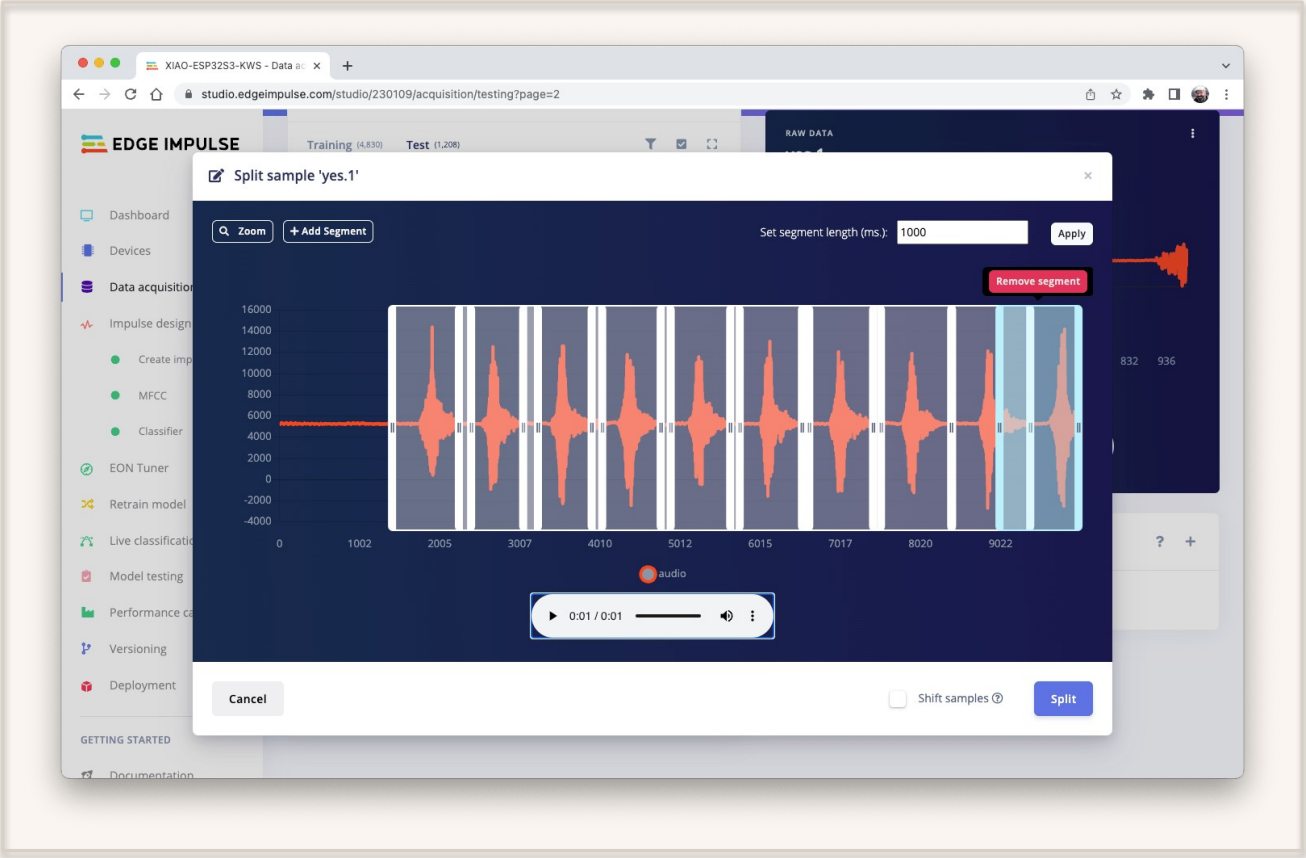
Upload the WAV files with their labels



1. Upload existing data, one class at a time, with its label
2. 4,830 training samples and 1,217 for testing
3. 78 % / 22 % split — automatic at upload, or later in the Dashboard
4. Every sample is 1 s long, the same as Speech Commands

SPLIT SAMPLE

Cut each ten-second recording into one-second clips



1. Three-dot menu on the sample › Split sample
2. Segment length 1,000 ms — the Studio finds each word
3. Add or remove segments by hand, then Split

THE IMPULSE

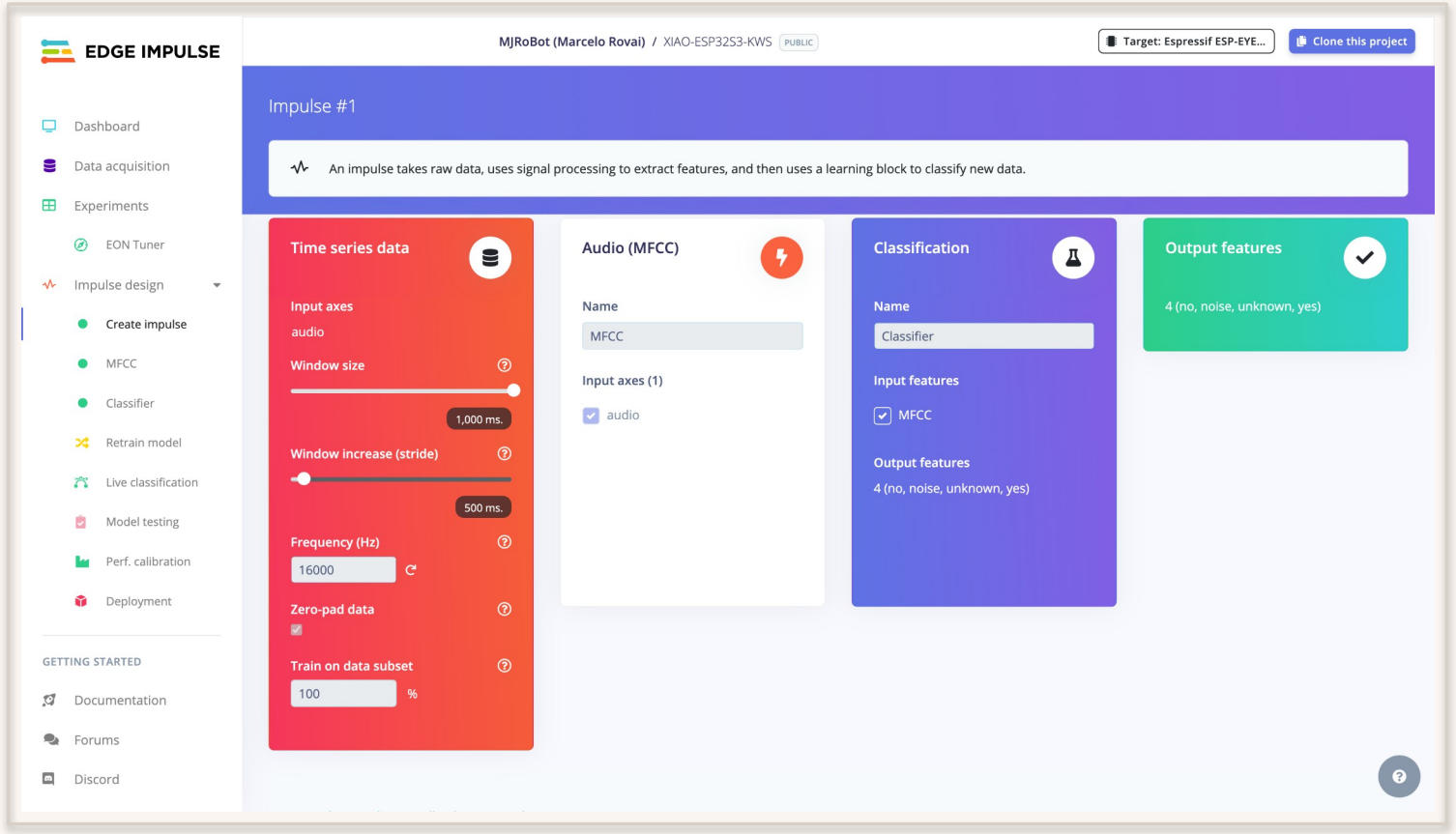
From one second of audio to four probabilities



The window slides every 500 ms over each sample — a cheap form of data augmentation.

CREATE IMPULSE

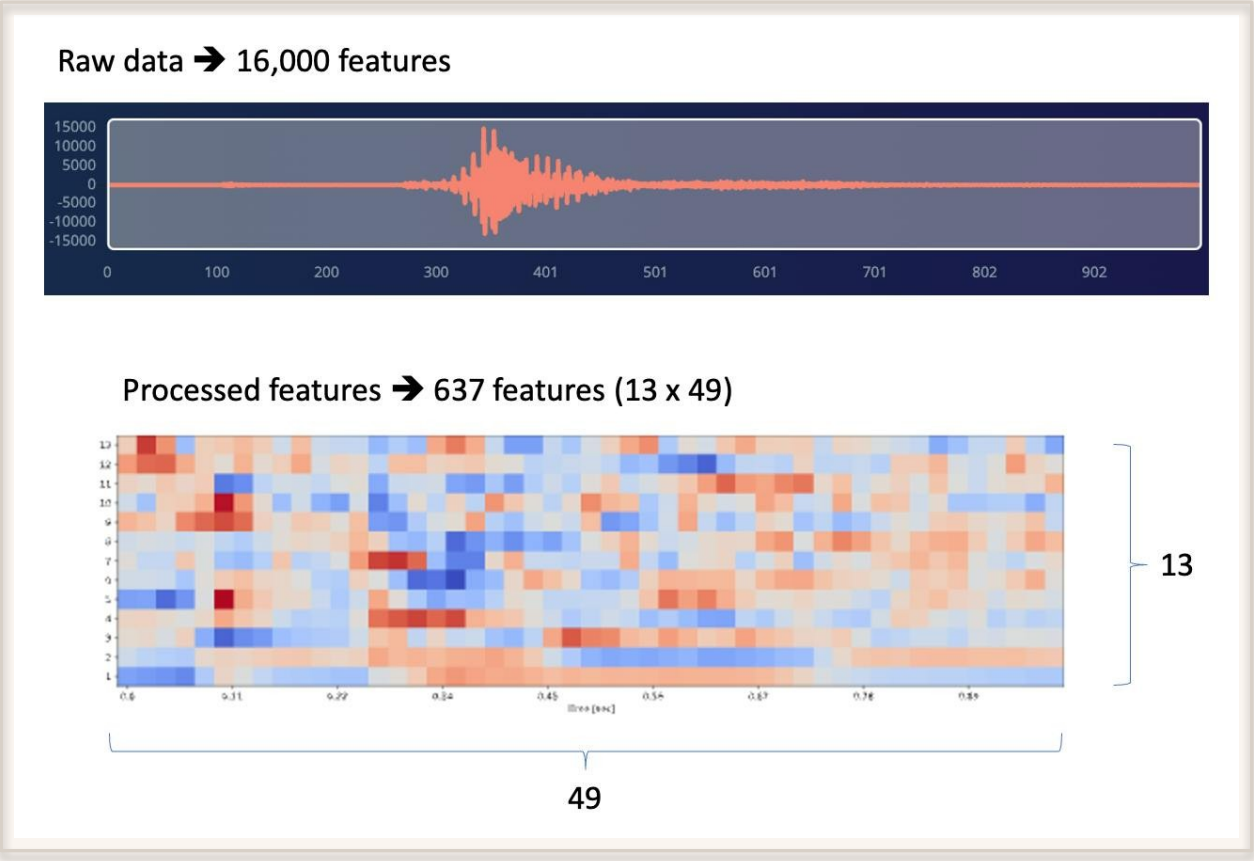
Time series data, Audio (MFCC), and a Classification block



1. Window 1,000 ms, stride 500 ms, 16,000 Hz
2. Zero-pad data on — short samples are filled with zeros
3. Audio (MFCC) as the processing block
4. Classification block — 4 outputs: no, noise, unknown, yes

WHAT MFCC DOES

16,000 numbers in, 637 numbers out



1. Raw data: one second of audio, 16,000 samples
2. Processed features: 13 coefficients × 49 time frames
3. Time runs left to right — it is a small image

INSIDE THE MFCC BLOCK

Four steps from a waveform to 13 × 49

01

Cut into frames

25 ms frames, a new one every 20 ms. One second gives 49 frames.

02

FFT per frame

A 512-point FFT shows which frequencies are present in each frame.

03

Mel filter bank

32 filters spaced like human hearing — dense in the low tones, sparse in the high.

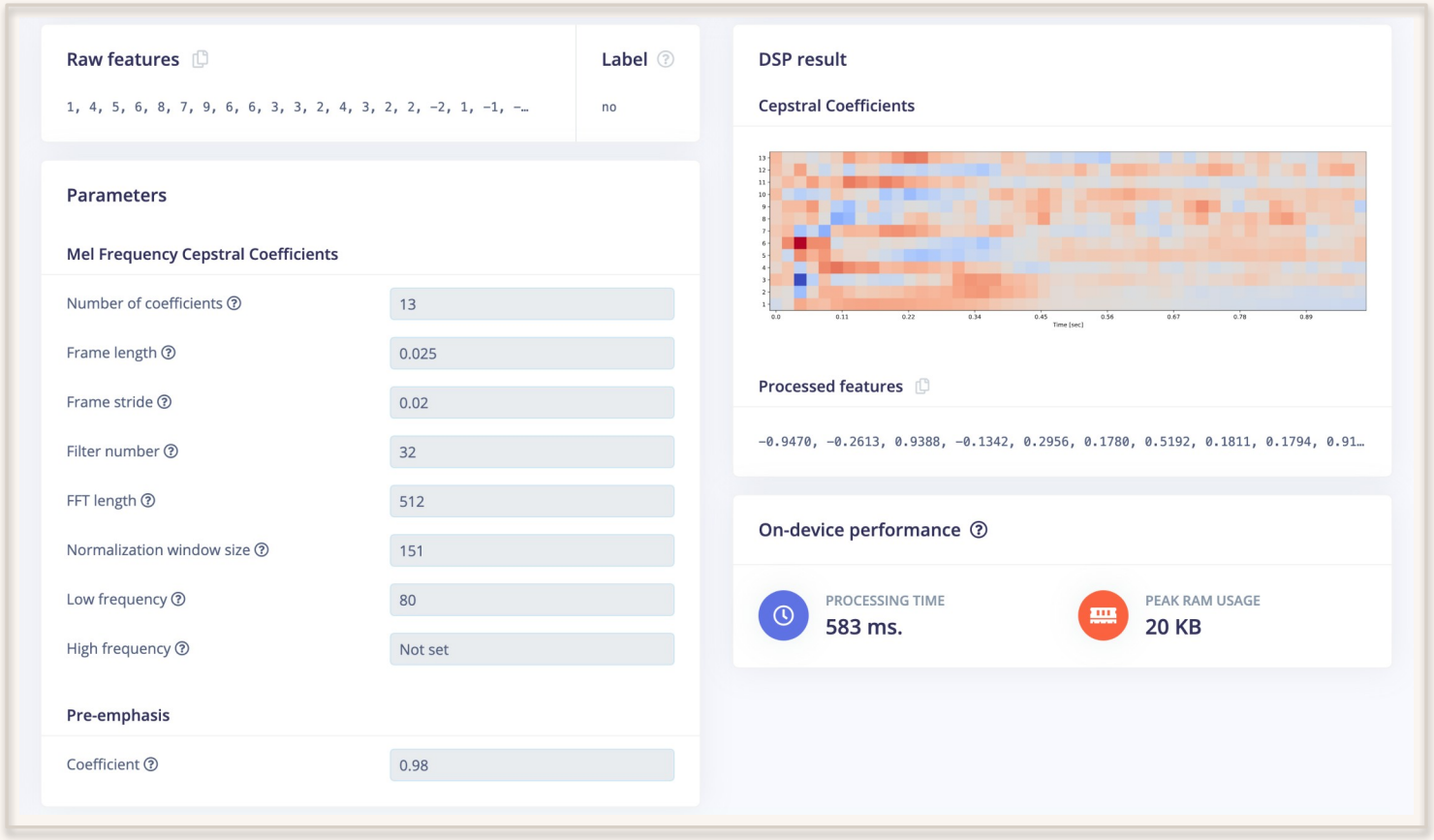
04

Log and DCT

Compress the energies and keep the first 13 coefficients — the shape of the voice.

MFCC PARAMETERS

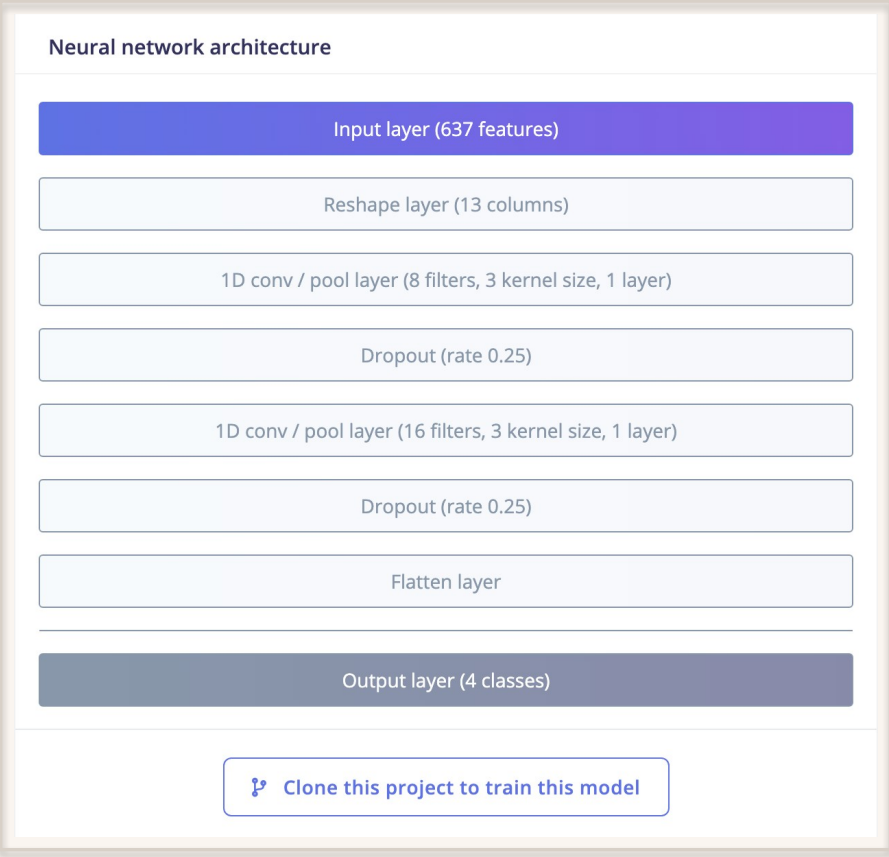
Autotune picked the parameters; the board pays for them



1. 13 coefficients, frame length 0.025 s, stride 0.02 s
2. 32 filters, FFT length 512, low frequency 80 Hz
3. The DSP result is the 13 × 49 cepstral image
4. The on-device estimate for the DSP alone is in hundreds of milliseconds

MODEL DESIGN

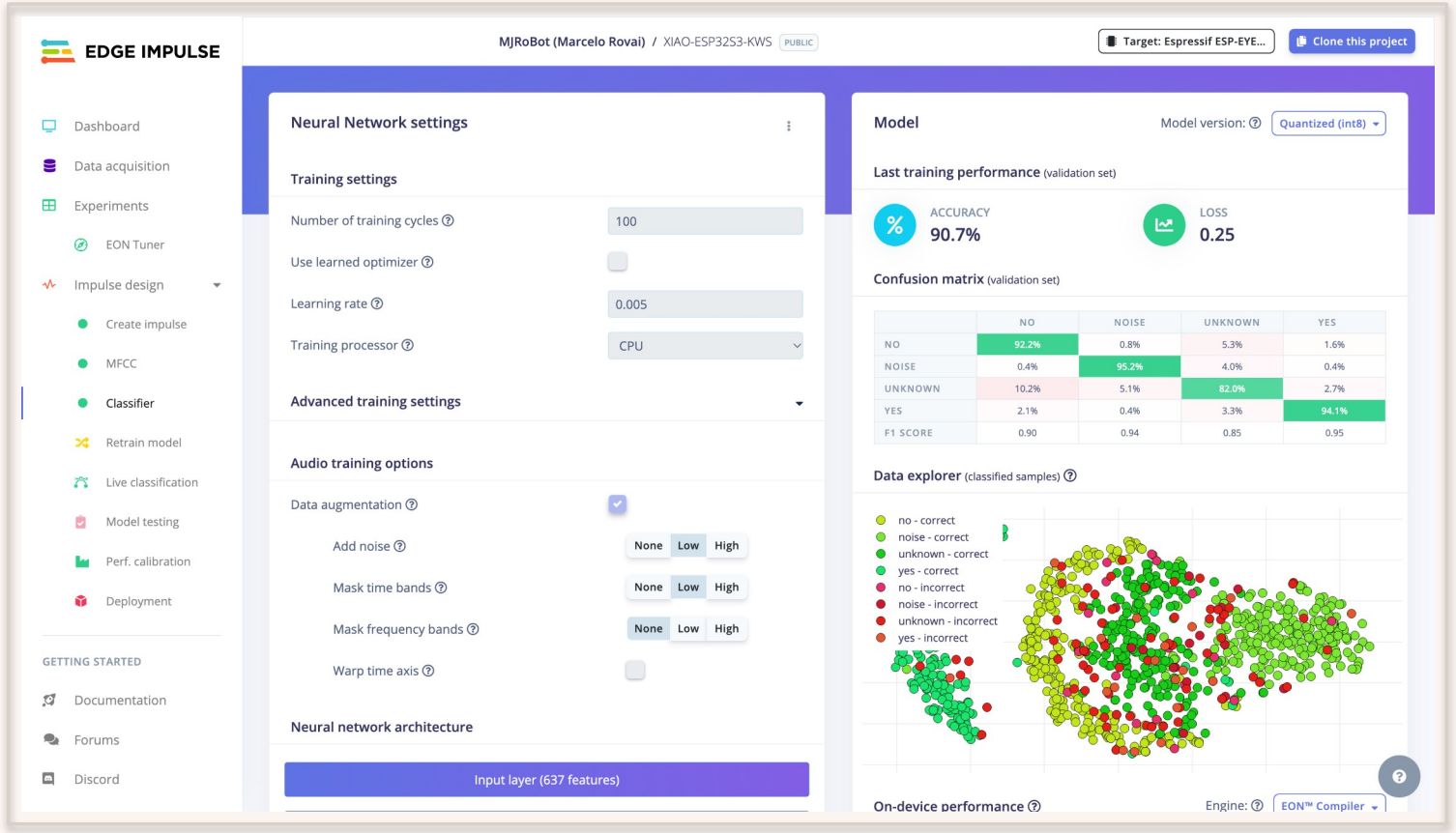
A small 1D CNN that reads the MFCC image in time



1. Input of 637 features, reshaped into 13 columns
2. 1D conv + pool, 8 filters, kernel 3, then dropout 0.25
3. 1D conv + pool, 16 filters, kernel 3, then dropout 0.25
4. Flatten, and an output layer with 4 classes

TRAINING

100 epochs, and 90.7 % on validation



1. 100 training cycles, learning rate 0.005
2. Augmentation: add noise (low) and mask time bands (low)
3. Validation accuracy 90.7 %, loss 0.25 — int8 model
4. On-device estimate: 6 ms, 3.7 KB of RAM, 27.1 KB of flash

READ THE CONFUSION MATRIX

Yes is the strongest class, and unknown the weakest

ACTUAL ↓ / PREDICTED →	NO	NOISE	UNKNOWN	YES
No	92.2 %	0.8 %	5.3 %	1.6 %
Noise	0.4 %	95.2 %	4.0 %	0.4 %
Unknown	10.2 %	5.1 %	82.0 %	2.7 %
Yes	2.1 %	0.4 %	3.3 %	94.1 %
F1 score	0.90	0.94	0.85	0.95

Validation set, int8 model. Yes, the word that triggers the action, has the best F1 — that is the one that matters.

MODEL TESTING

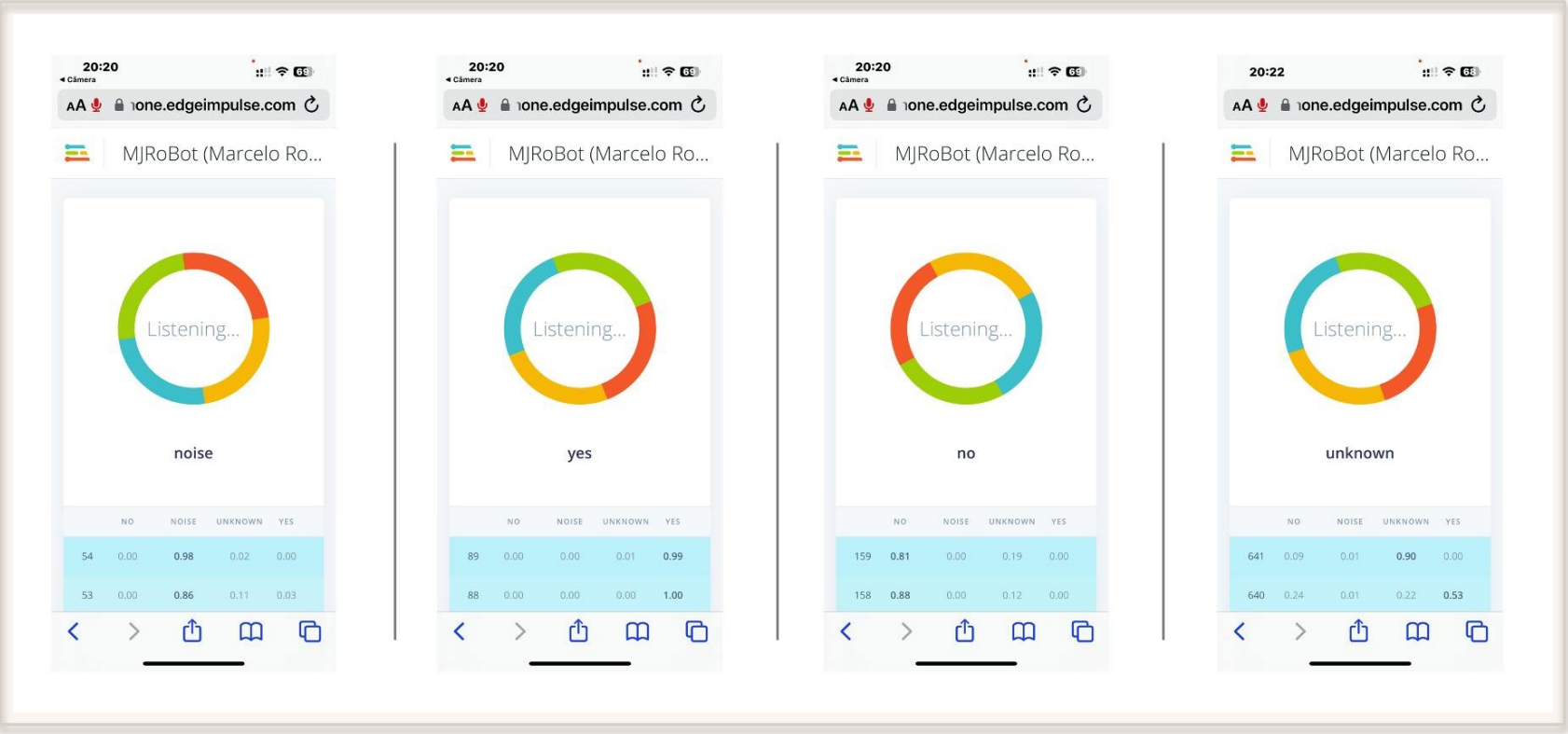
86.7 % on 1,217 samples the model never heard

ACTUAL ↓ / PREDICTED →	NO	NOISE	UNKNOWN	YES	UNCERTAIN
No	86.3 %	0.7 %	3.9 %	1.4 %	7.7 %
Noise	0 %	88.6 %	3.3 %	0.7 %	7.5 %
Unknown	4.4 %	2.7 %	78.1 %	1.7 %	13.1 %
Yes	0.3 %	0 %	0.7 %	93.9 %	5.1 %
F1 score	0.90	0.92	0.84	0.95	

Test set, as in the book. "Uncertain" means no class passed the confidence threshold — the model abstained instead of guessing.

LIVE CLASSIFICATION

Test with your phone before you flash anything



1. Live classification › Connect a development board › scan the QR code
2. Choose Classification in the phone's browser
3. Speak, and the four probabilities update every window

04

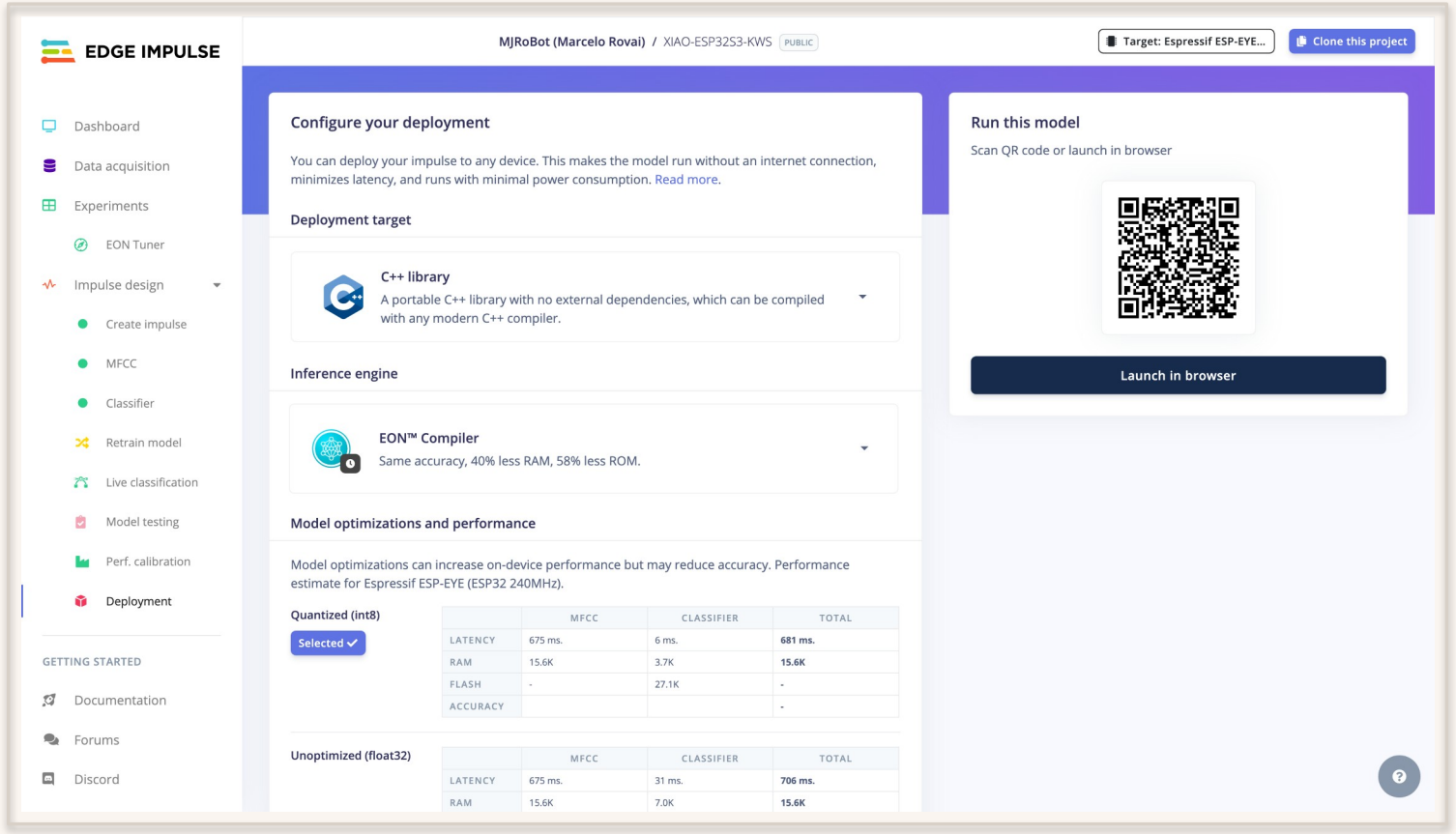
DEPLOY AND ACT

Listening on the board

An Arduino library, five edits for the XIAO microphone, and then an LED and the OLED.

DEPLOYMENT

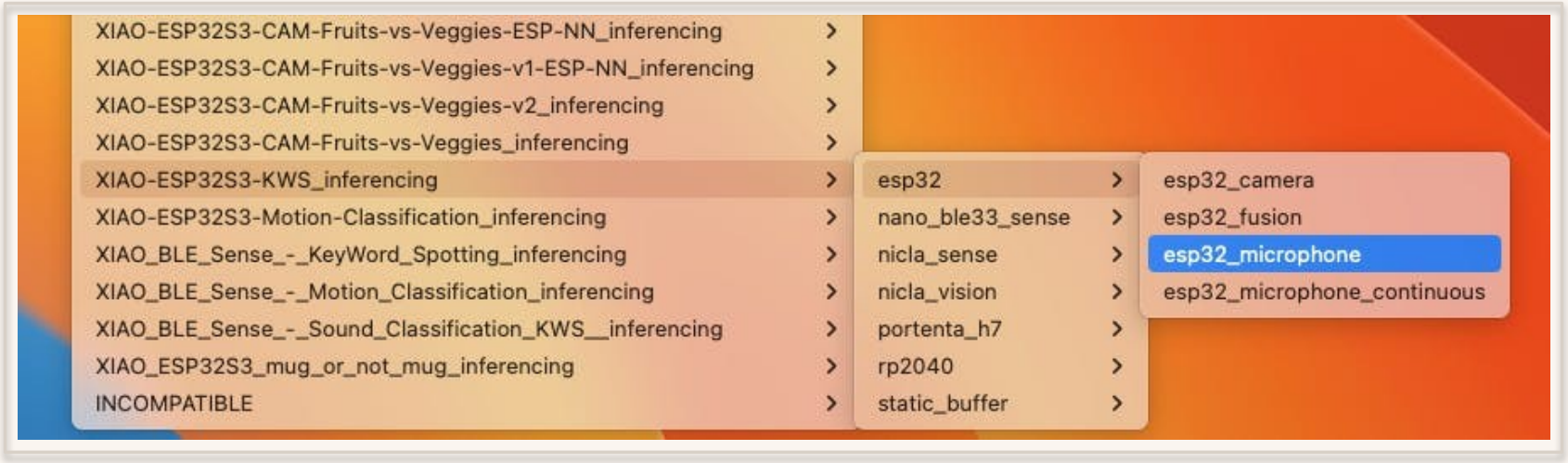
Arduino library, quantized int8, EON Compiler on



1. Search for and select Arduino library as the deployment target
2. EON Compiler — same accuracy, less RAM and flash
3. int8 estimate for the ESP-EYE: MFCC 675 ms + classifier 6 ms = 681 ms
4. float32: the classifier takes 31 ms — the MFCC is the same 675 ms

THE EXAMPLE

Start from esp32_microphone, written for the ESP-EYE



1. Sketch › Include Library › Add .ZIP Library
2. File › Examples › XIAO-ESP32S3-KWS_inferencing
3. esp32 › esp32_microphone

The ESP-EYE uses a different microphone setup, so the example needs a few changes.

FIVE EDITS

From the ESP-EYE microphone to the XIAO's

`xiao_esp32s3_microphone.ino — the changed parts`

```
// 1. includes: use the I2S library instead of driver/i2s.h
#include <I2S.h>
#define SAMPLE_RATE 16000U
#define SAMPLE_BITS 16

// 2. setup(): start the PDM microphone once
I2S.setAllPins(-1, 42, 41, -1, -1);
if (!I2S.begin(PDM_MONO_MODE, SAMPLE_RATE, SAMPLE_BITS)) { ... }

// 3. capture_samples(): read from the XIAO microphone
esp_i2s::i2s_read(esp_i2s::I2S_NUM_0, (void*)sampleBuffer,
                  i2s_bytes_to_read, &bytes_read, 100);

// 4. microphone_inference_start(): delete the i2s_init() call
// 5. microphone_inference_end():
free(sampleBuffer);
ei_free(inference.buffer);
```

— The complete sketch is on the project's GitHub.

— Same rules as before — esp32 core 2.0.17 and PSRAM on.

— The model, the MFCC code, and the sliding window come from the library unchanged.

SERIAL MONITOR

515 ms to prepare, 3 ms to classify

```
Edge Impulse Inferencing Demo
Inferencing settings:
        Interval: 0.062500 ms.
        Frame size: 16000
        Sample length: 1000 ms.
        No. of classes: 4
Starting continious inference in 2 seconds...
Recording...
Predictions (DSP: 515 ms., Classification: 3 ms., Anomaly: 0 ms.):
    no: 0.007813
    noise: 0.964844
    unknown: 0.023437
    yes: 0.000000
```

DSP (MFCC)

515 ms

MODEL

3 ms

WINDOW

16,000 samples

THE LESSON OF THIS CHAPTER

The network takes 3 ms. Turning sound into features takes 515 ms. In audio, the features are the bill.

To make it faster, tune the MFCC first — FFT length, number of filters, frame stride — not the network.

POST-PROCESSING · LED

Find the top class, and light the LED on "yes"

xiao_esp32s3_microphone_led.ino — inside loop()

```
#define LED_BUILT_IN 21          // active LOW

int pred_index = 0;
float pred_value = 0;
for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
  if (result.classification[ix].value > pred_value) {
    pred_index = ix;
    pred_value = result.classification[ix].value;
  }
}

if (pred_index == 3) {           // 0 no, 1 noise, 2 unknown, 3 yes
  digitalWrite(LED_BUILT_IN, LOW);   // turn on
} else {
  digitalWrite(LED_BUILT_IN, HIGH);  // turn off
}
```

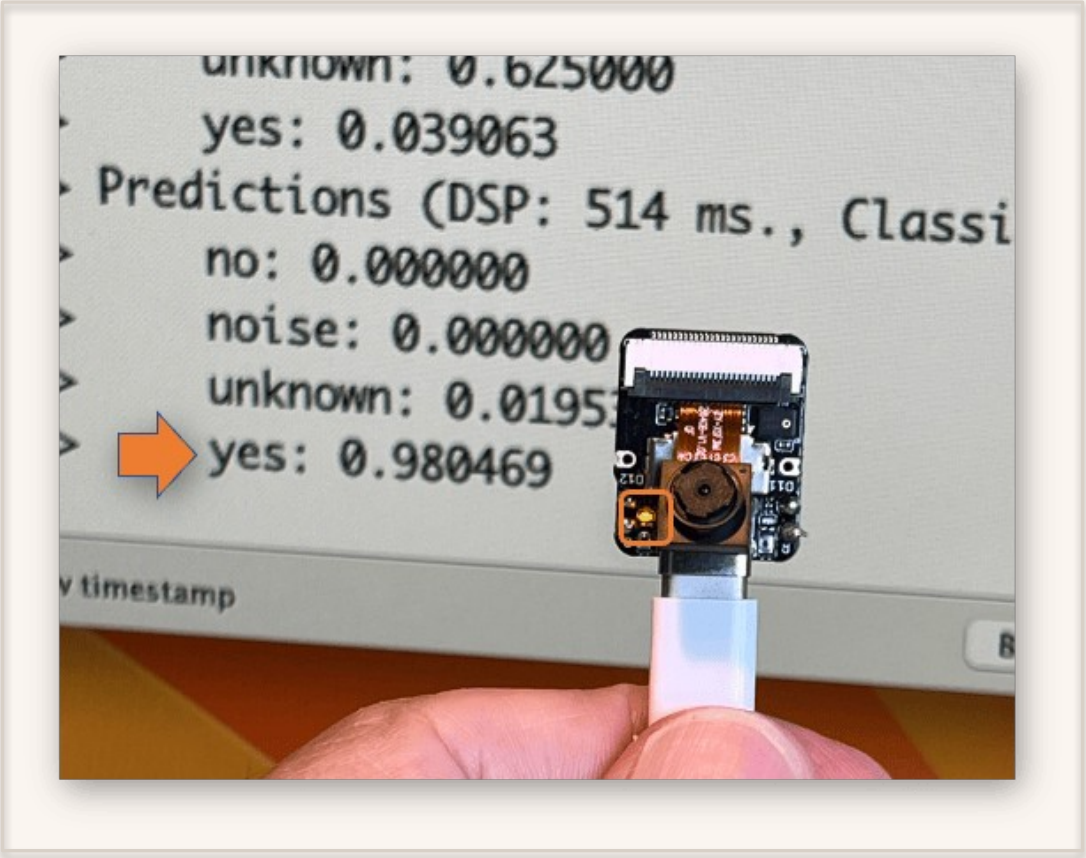
— The loop keeps the index of the highest probability.

— Index 3 is yes, because the classes are in alphabetical order.

— The LED on GPIO 21 lights when the pin is LOW.

IT WORKS

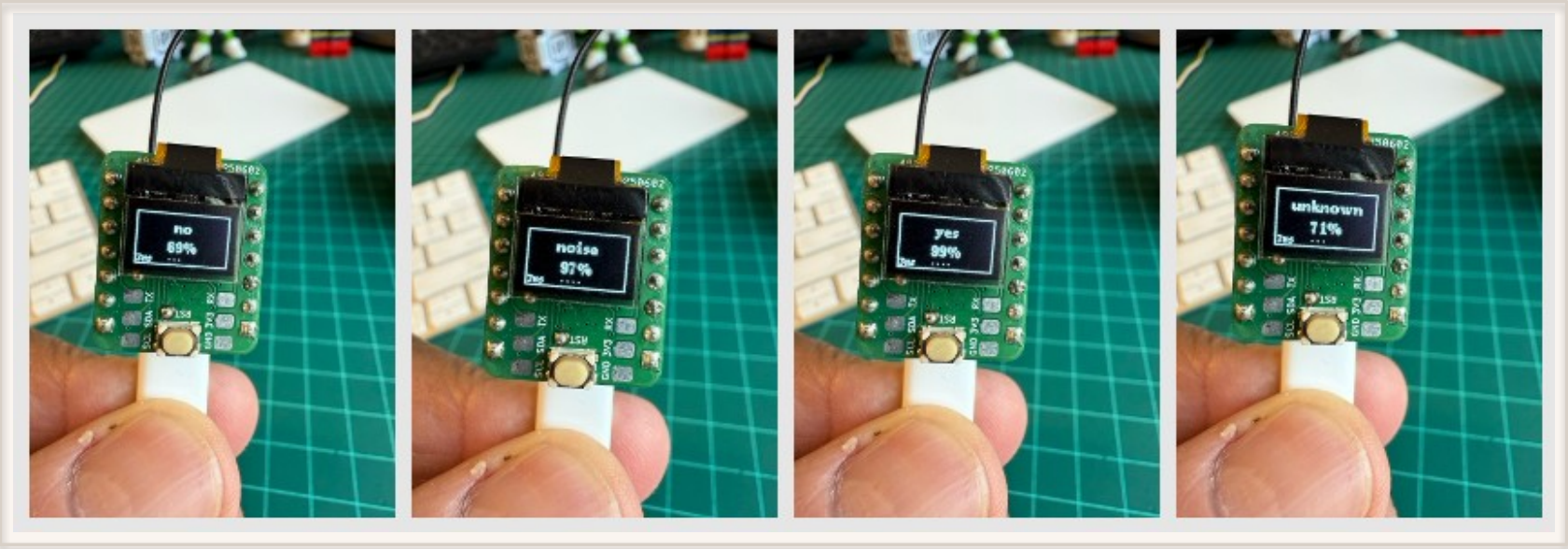
Say "yes", and the LED turns on



1. yes at 0.98 — the top class, so pred_index is 3
2. The built-in LED lights up
3. The same line could trigger a relay, or wake up stage 2

POST-PROCESSING · XIAOML KIT OLED

The class and its confidence, right on the board



1. The 0.42" OLED, 72 × 40 pixels, shows the top class and its score
2. Class names and count are read from the model — no hard-coded labels
3. No computer needed — a standalone sensor

THE BUDGET

In vision the model is the bill; in audio, the features are

CONFIGURATION	INPUT	PRE-PROCESS	MODEL	OUTPUT
KWS, Studio estimate (ESP-EYE, int8)	1 s audio	675 ms	6 ms	4 scores
KWS, XIAO, Arduino (book)	1 s audio	515 ms	3 ms	4 scores
Ch. 3 FOMO, XIAO	96^2, gray	4 ms	143 ms	centroids
Ch. 2 classifier, XIAO (measured)	96^2, RGB	4 ms	199 ms	1 label

Same board in every row. The ESP32-S3's faster cores beat the ESP-EYE estimate again, but the MFCC still dominates.

BEYOND KEYWORDS

The same pipeline classifies any sound

01

Safety and security

Breaking glass, alarms, intrusion — events, not words.

02

Industry

Machine health and anomalies, heard before they are seen.

03

Health

Coughs, breathing, sleep — on a device that never sends the audio out.

04

Environment

Birds, insects, urban noise — on a battery, in the field.

SUMMARY

Three things to remember

01 Stage 1 must be tiny

It listens all the time, so it must be small, local, and cheap in energy. It only wakes the big model.

02 Sound becomes an image

16,000 samples become a 13 × 49 MFCC image, and a small CNN classifies it.

03 In audio, the features are the bill

515 ms of MFCC against 3 ms of model. To go faster, tune the pre-processing first.

CHECK YOURSELF

The board prints "DSP: 515 ms, Classification: 3 ms". You need a faster response. What do you change first?

A Remove one convolution block from the CNN

B The MFCC parameters — for example a 256-point FFT or a longer frame stride

C Deploy the float32 model instead of int8

The network is only 3 ms of a 518 ms cycle. Only a cheaper pre-processing can make a real difference; float32 would make the model slower, not faster.

The interactive deck gives per-answer feedback.

Questions?

Next — Motion Classification and Anomaly Detection, where the sensor is an IMU.

