VOICE RECOGNITION

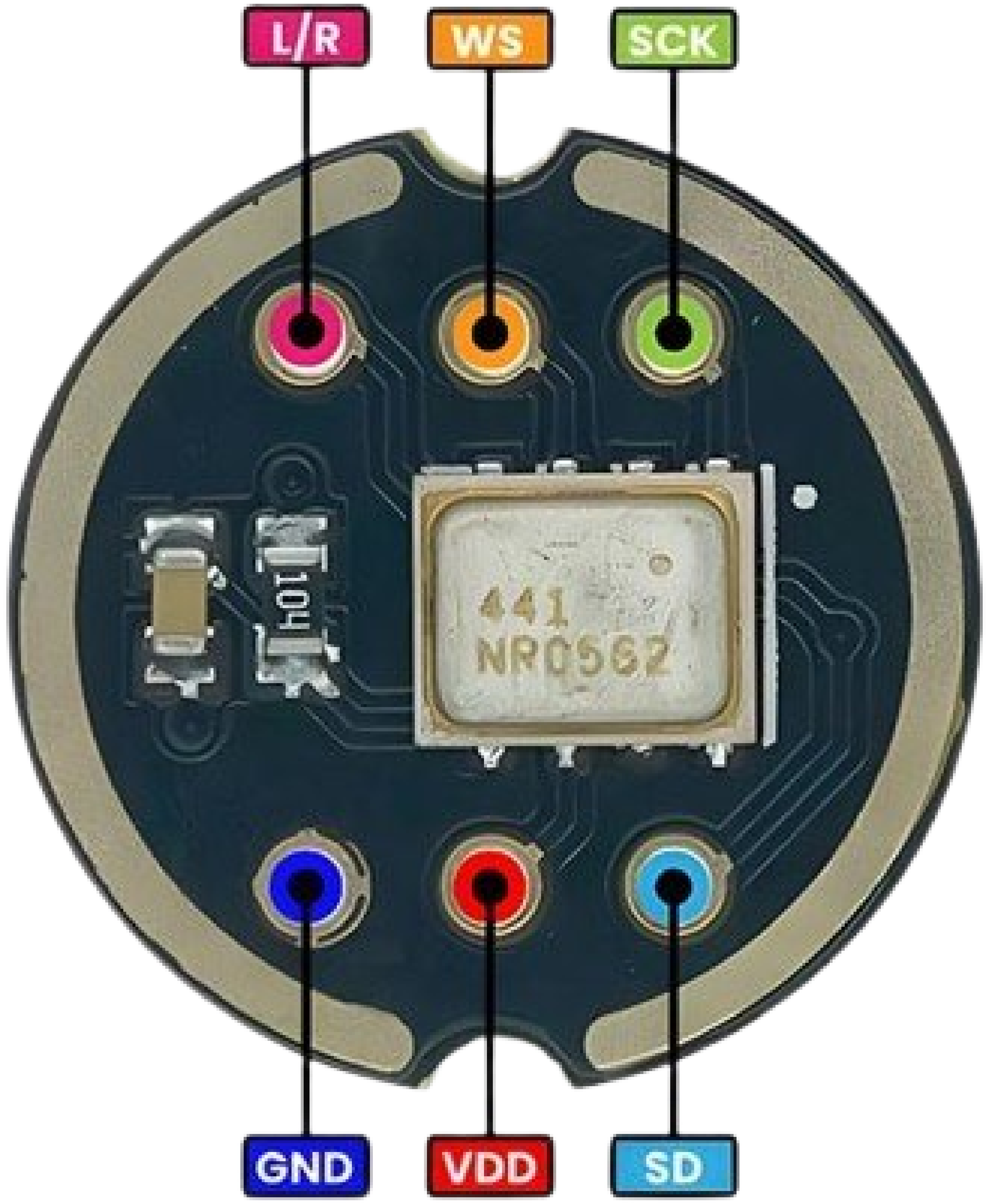
REAL WORLD APPLICATIONS

- **Virtual Assistants**: Voice recognition powers assistants like Alexa, Siri, and Google Assistant, enabling hands-free control of smart devices and access to information.
- **Security**: Voice biometrics are used for authentication in banking and secure systems, allowing user verification through unique vocal traits.
- **Customer Service**: Call centers use voice recognition to route calls and automate responses, enhancing customer support.
- **Translation**: Real-time voice translation helps overcome language barriers in travel, business, and education.
- **Accessibility**: Voice recognition aids individuals with disabilities by enabling hands-free device operation and real-time captioning.

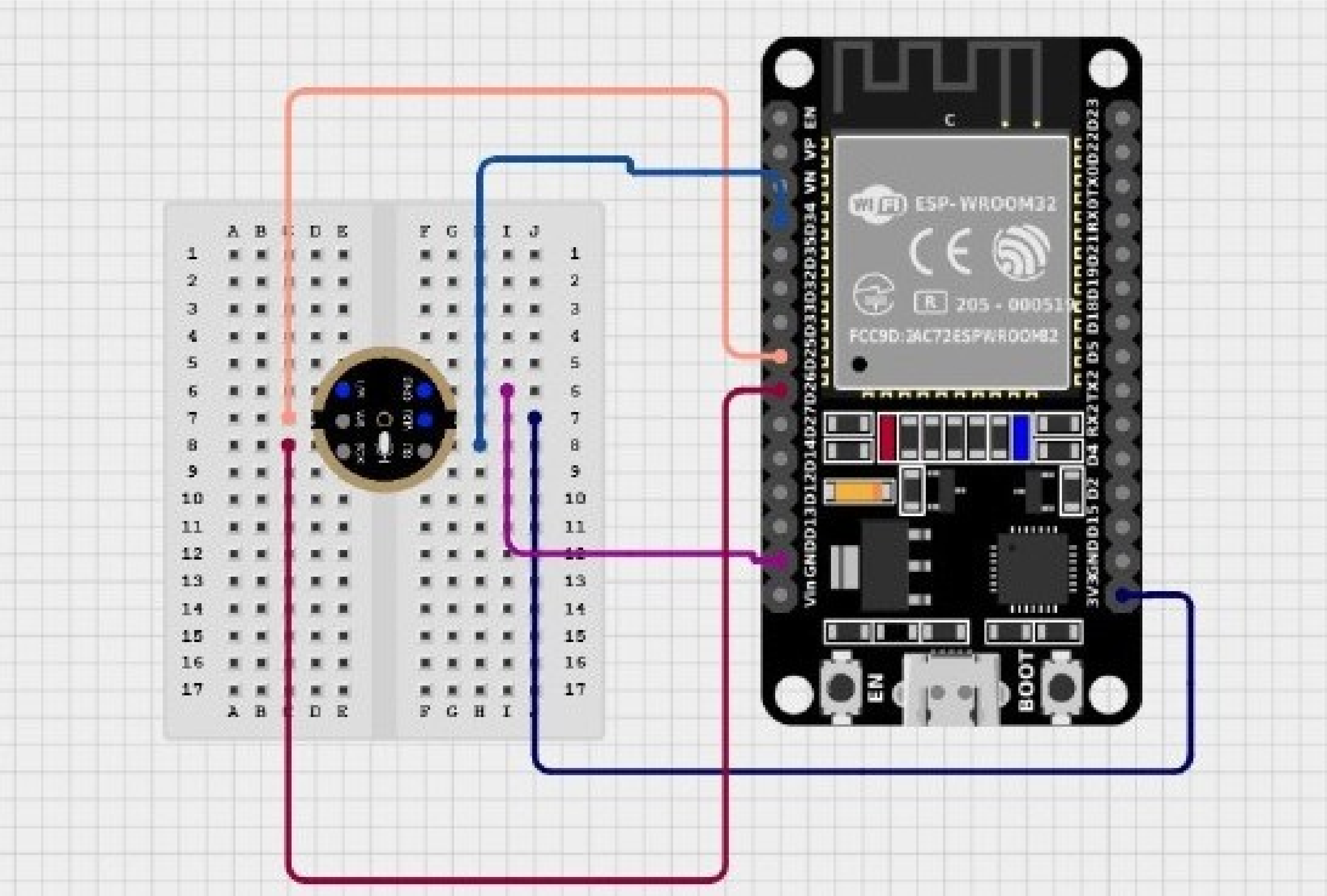
WORKING

The voice recognition project uses the INMP441 digital MEMS microphone to capture sound, converting it into digital data via I2S output. Edge Impulse processes this data with machine learning models to identify voice patterns or commands in real time. This setup enables quick, accurate voice recognition on edge devices with low power use, making it ideal for smart home controls, voice assistants, and other hands-free applications.

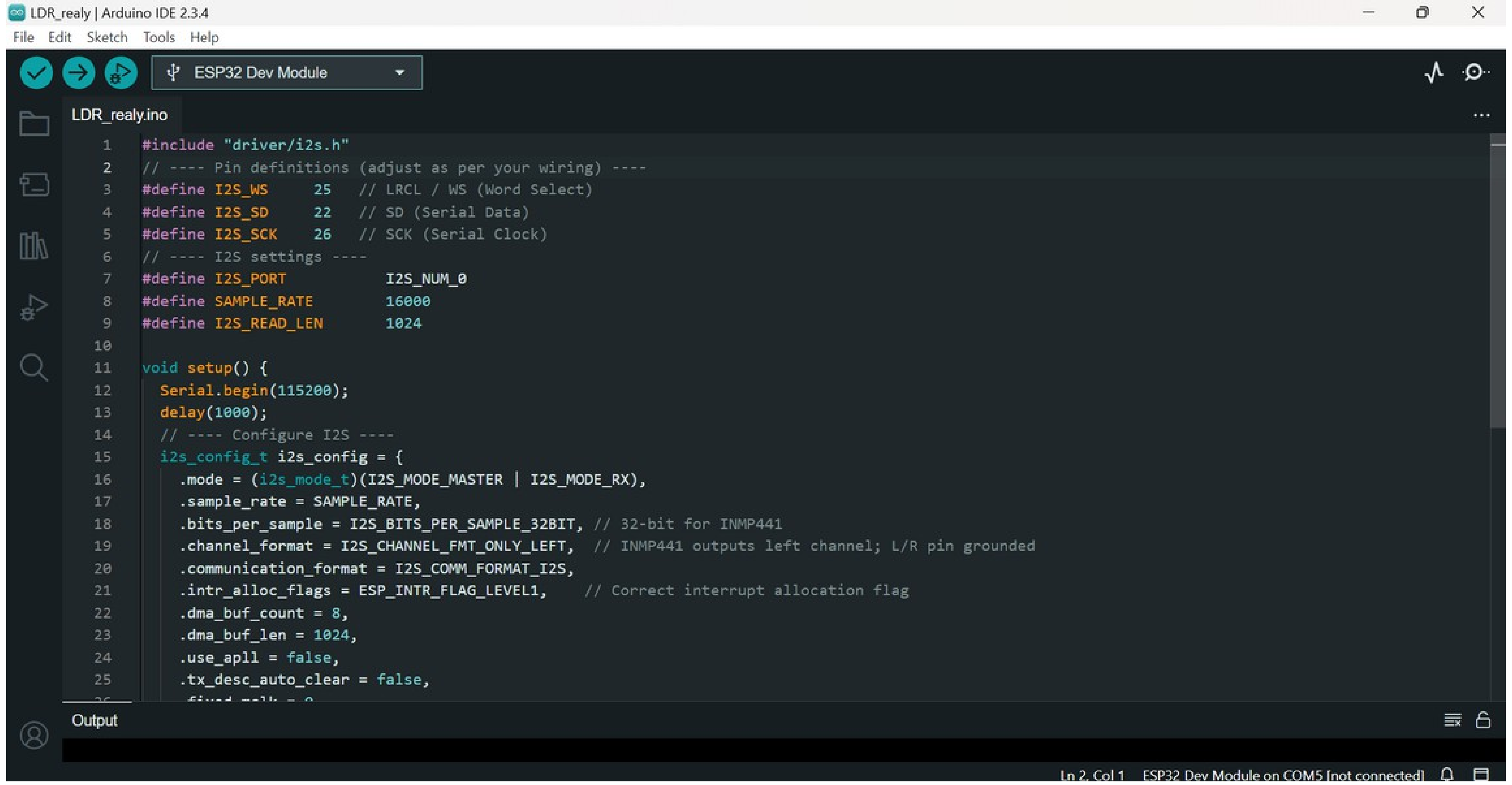
PINOUT



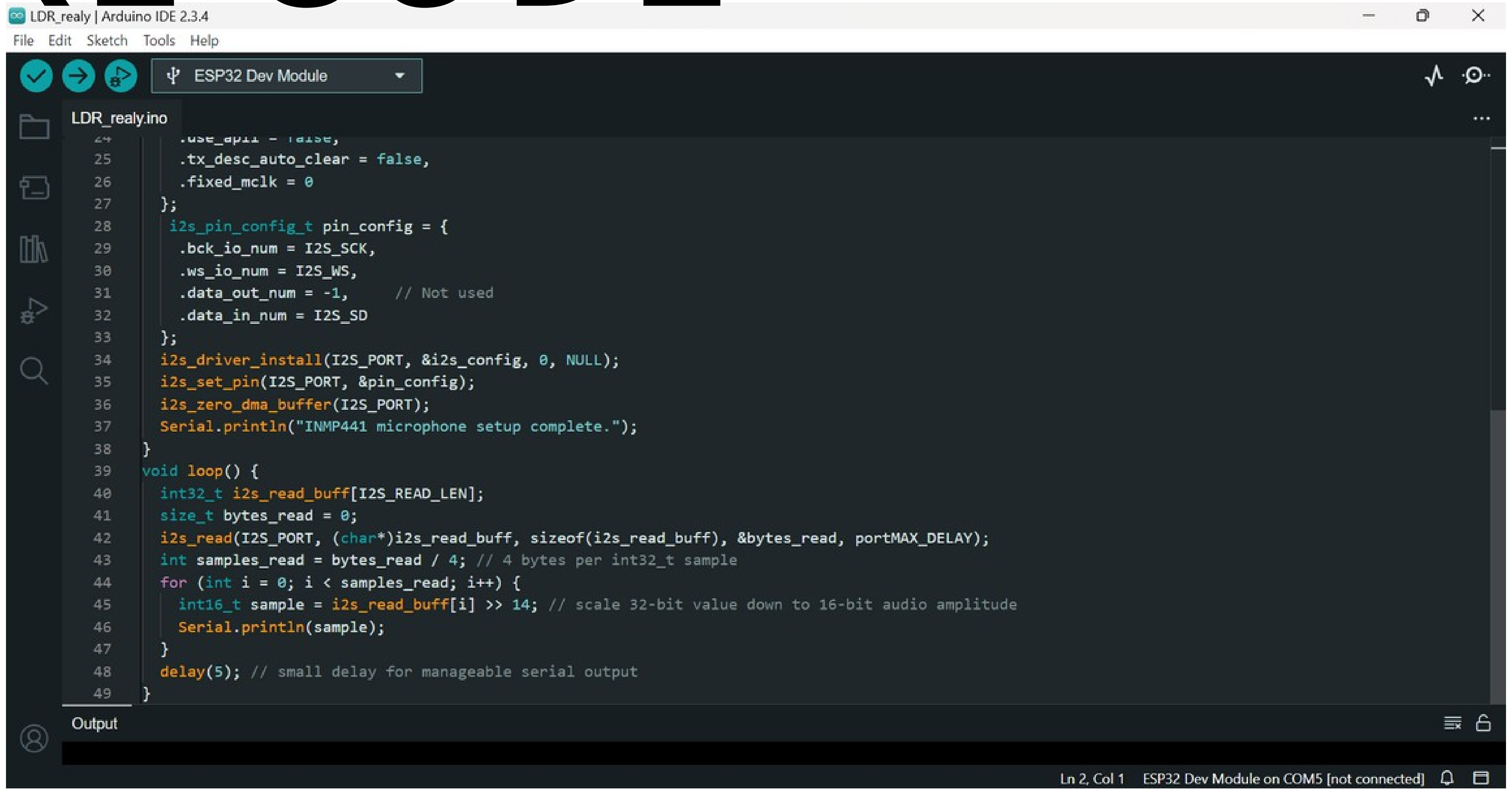
EK NAZAR IDHAR BHI



FIRMWARE CODE



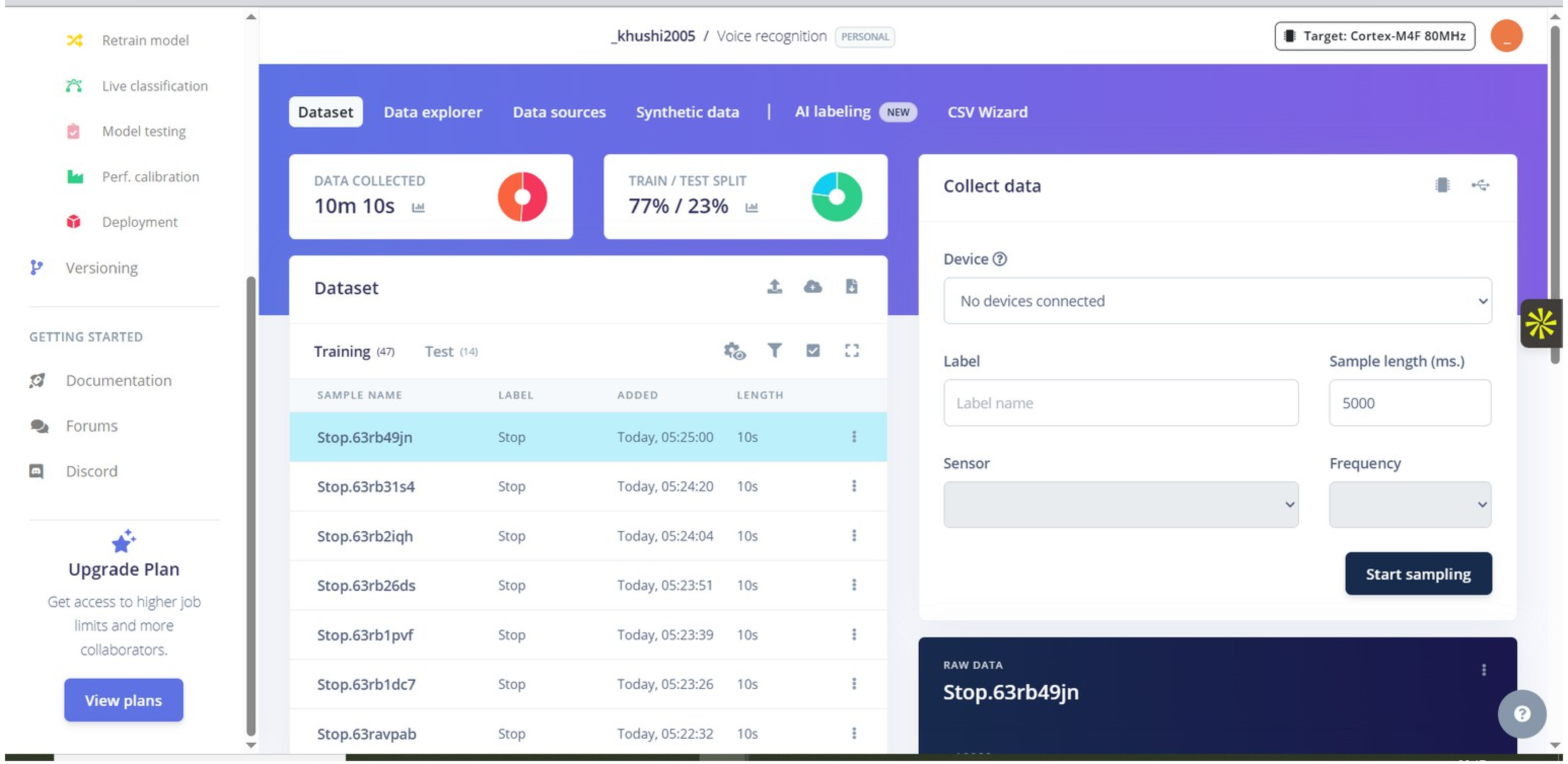

```
#include "driver/i2s.h"
// ---- Pin definitions (adjust as per your wiring) ----
#define I2S_WS      25   // LRCL / WS (Word Select)
#define I2S_SD      22   // SD (Serial Data)
#define I2S_SCK     26   // SCK (Serial Clock)
// ---- I2S settings ----
#define I2S_PORT            I2S_NUM_0
#define SAMPLE_RATE         16000
#define I2S_READ_LEN        1024

void setup() {
  Serial.begin(115200);
  delay(1000);
  // ---- Configure I2S ----
  i2s_config_t i2s_config = {
    .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = I2S_BITS_PER_SAMPLE_32BIT, // 32-bit for INMP441
    .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,  // INMP441 outputs left channel; L/R pin grounded
    .communication_format = I2S_COMM_FORMAT_I2S,
    .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,    // Correct interrupt allocation flag
    .dma_buf_count = 8,
    .dma_buf_len = 1024,
    .use_apll = false,
    .tx_desc_auto_clear = false,
```




```
    .tx_desc_auto_clear = false,
    .fixed_mclk = 0
  };
   i2s_pin_config_t pin_config = {
    .bck_io_num = I2S_SCK,
    .ws_io_num = I2S_WS,
    .data_out_num = -1,     // Not used
    .data_in_num = I2S_SD
  };
  i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);
  i2s_set_pin(I2S_PORT, &pin_config);
  i2s_zero_dma_buffer(I2S_PORT);
  Serial.println("INMP441 microphone setup complete.");
}
void loop() {
  int32_t i2s_read_buff[I2S_READ_LEN];
  size_t bytes_read = 0;
  i2s_read(I2S_PORT, (char*)i2s_read_buff, sizeof(i2s_read_buff), &bytes_read, portMAX_DELAY);
  int samples_read = bytes_read / 4; // 4 bytes per int32_t sample
  for (int i = 0; i < samples_read; i++) {
    int16_t sample = i2s_read_buff[i] >> 14; // scale 32-bit value down to 16-bit audio amplitude
    Serial.println(sample);
  }
  delay(5); // small delay for manageable serial output
}
```

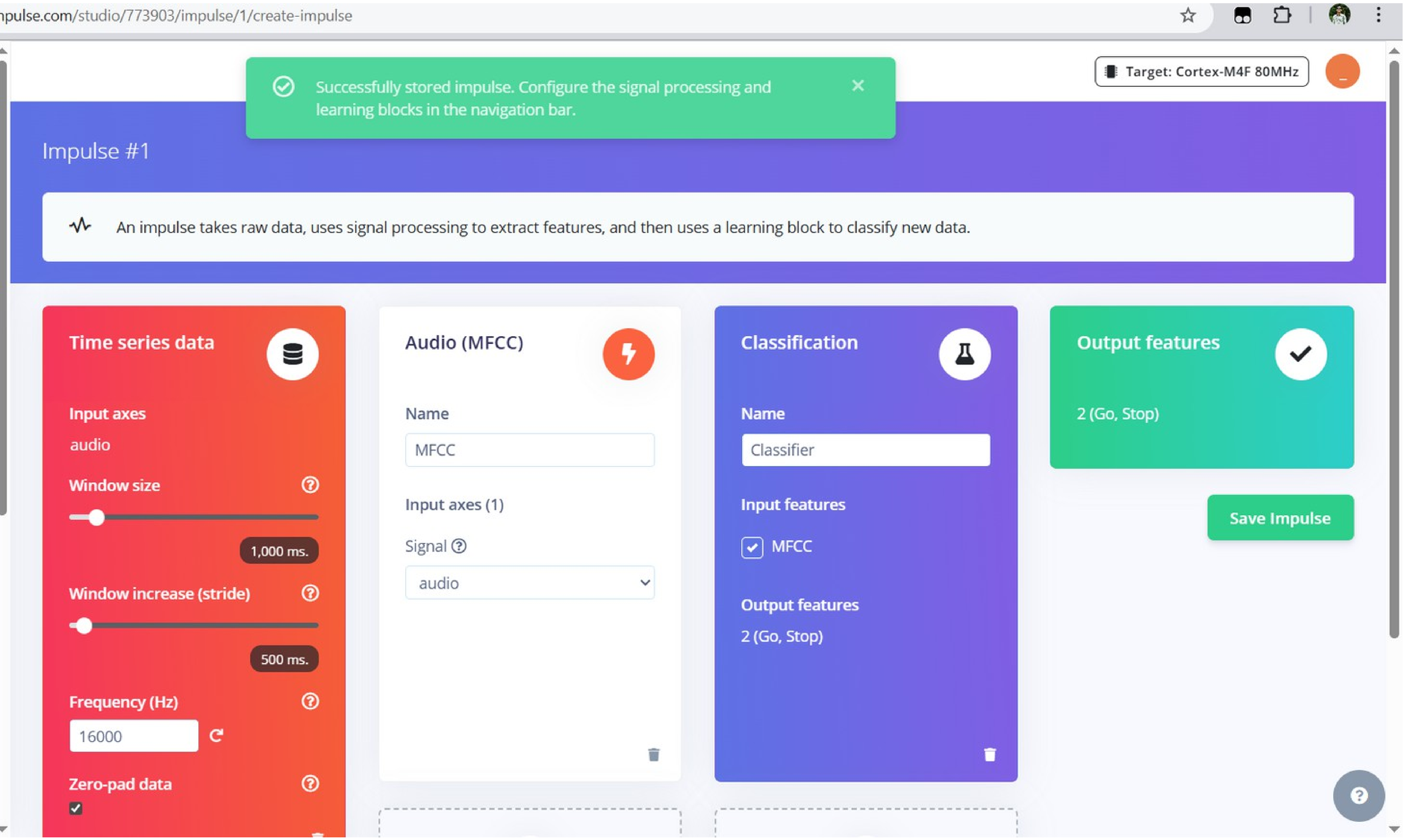
This code is used a firmware to your esp which enable it to send your data through A bridge called Edge impulse CLI. This is already uploaded in your esp..

DATA ACQUITION



We will use premade dataset for ‘Data Acquition’.

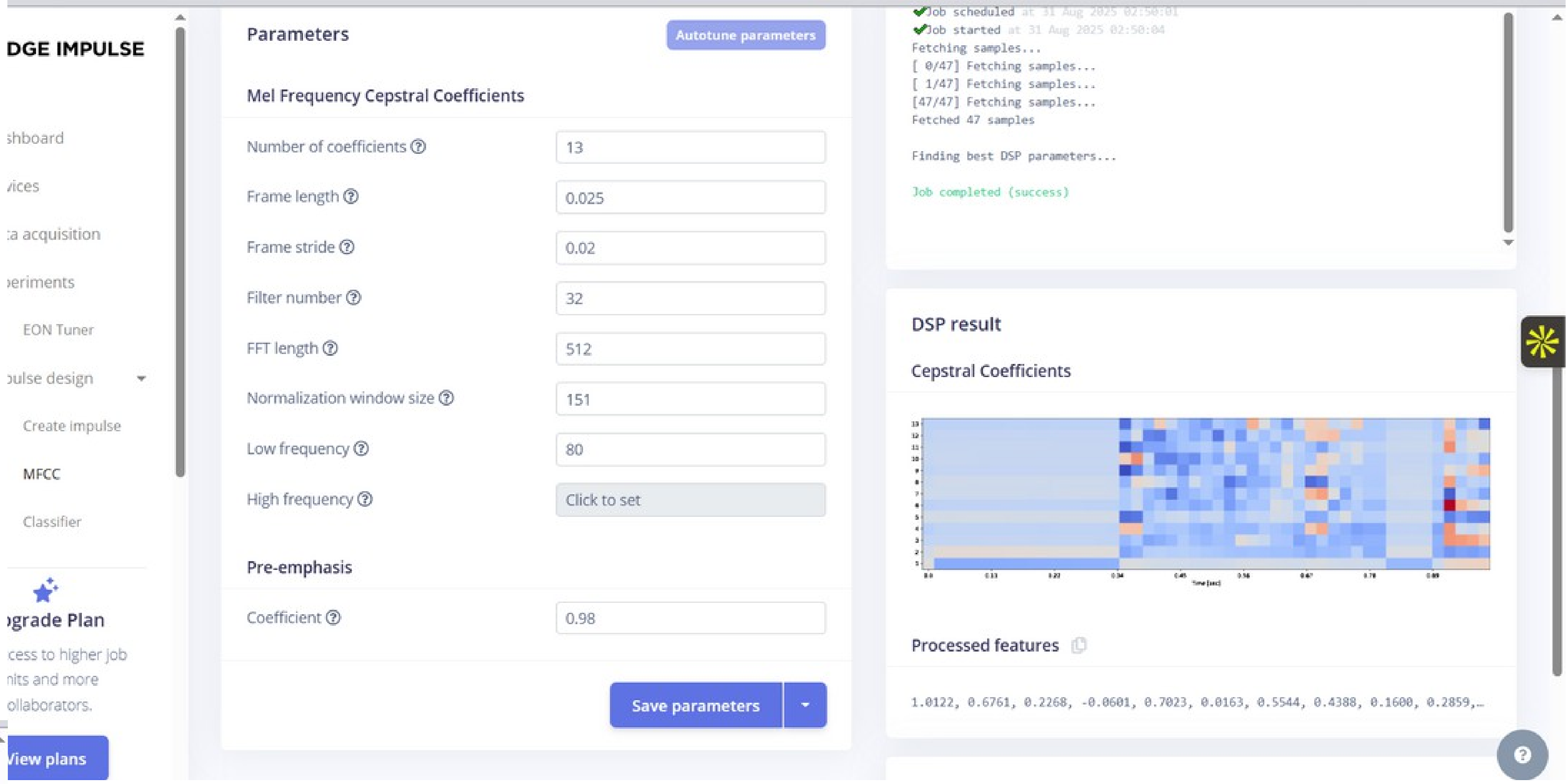
DESIGNING AN IMPULSE

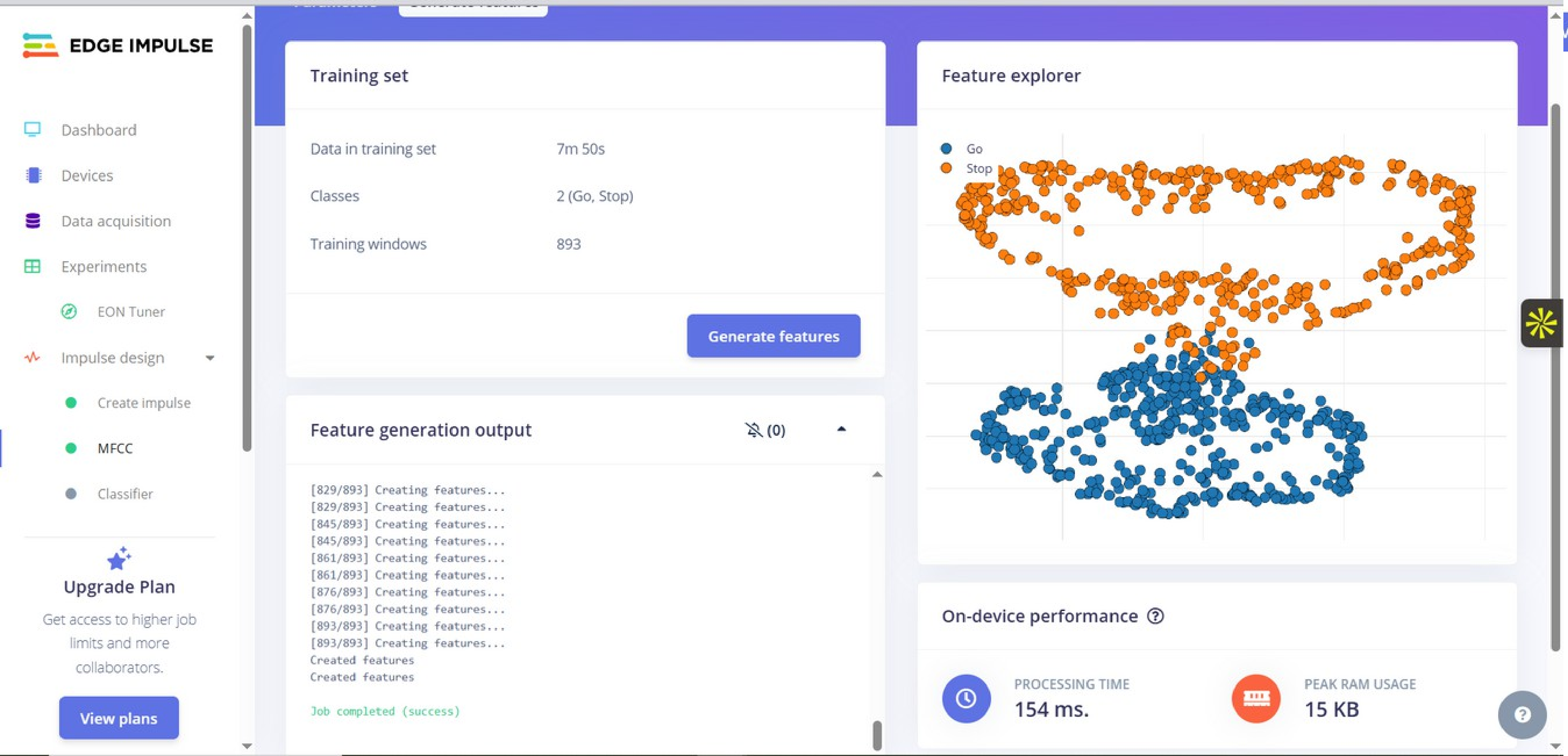


Go to the **Create impulse** tab, add a Time series data, an Audio (MFCC) and a Classification (Keras) block. Leave the window size to 1second (as that's the length of our audio samples in the dataset) and click **Save Impulse.**

CONFIGURE THE MFCC BLOCK

Now click on the **MFCC** tab in the left hand navigation menu. You just click on the autotune parameters and then save parameters.

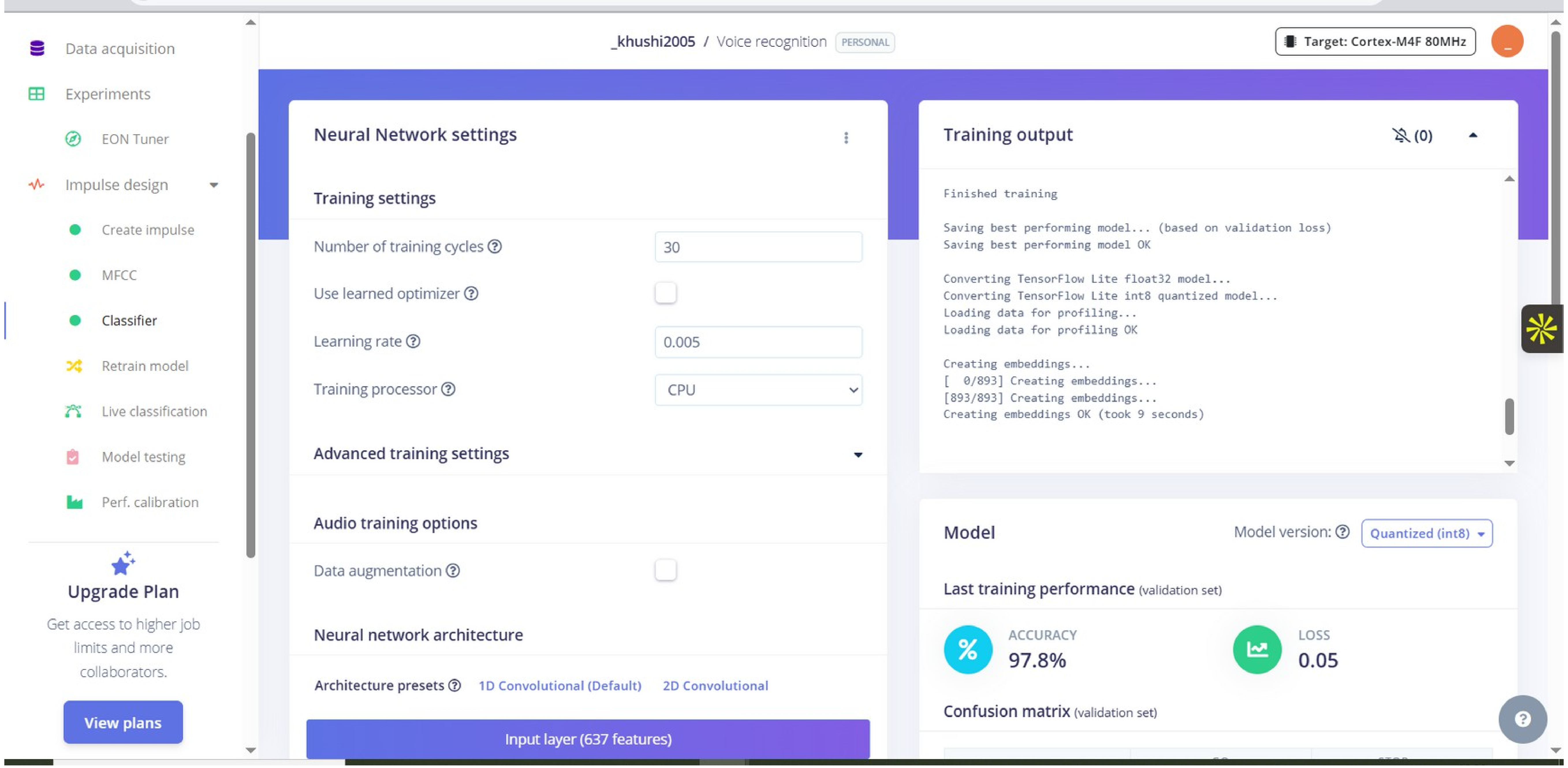




You will be redirected to feature generation window.

Click on generate features and then you can analyse the graph.

CONFIGURE THE NEURAL NETWORK

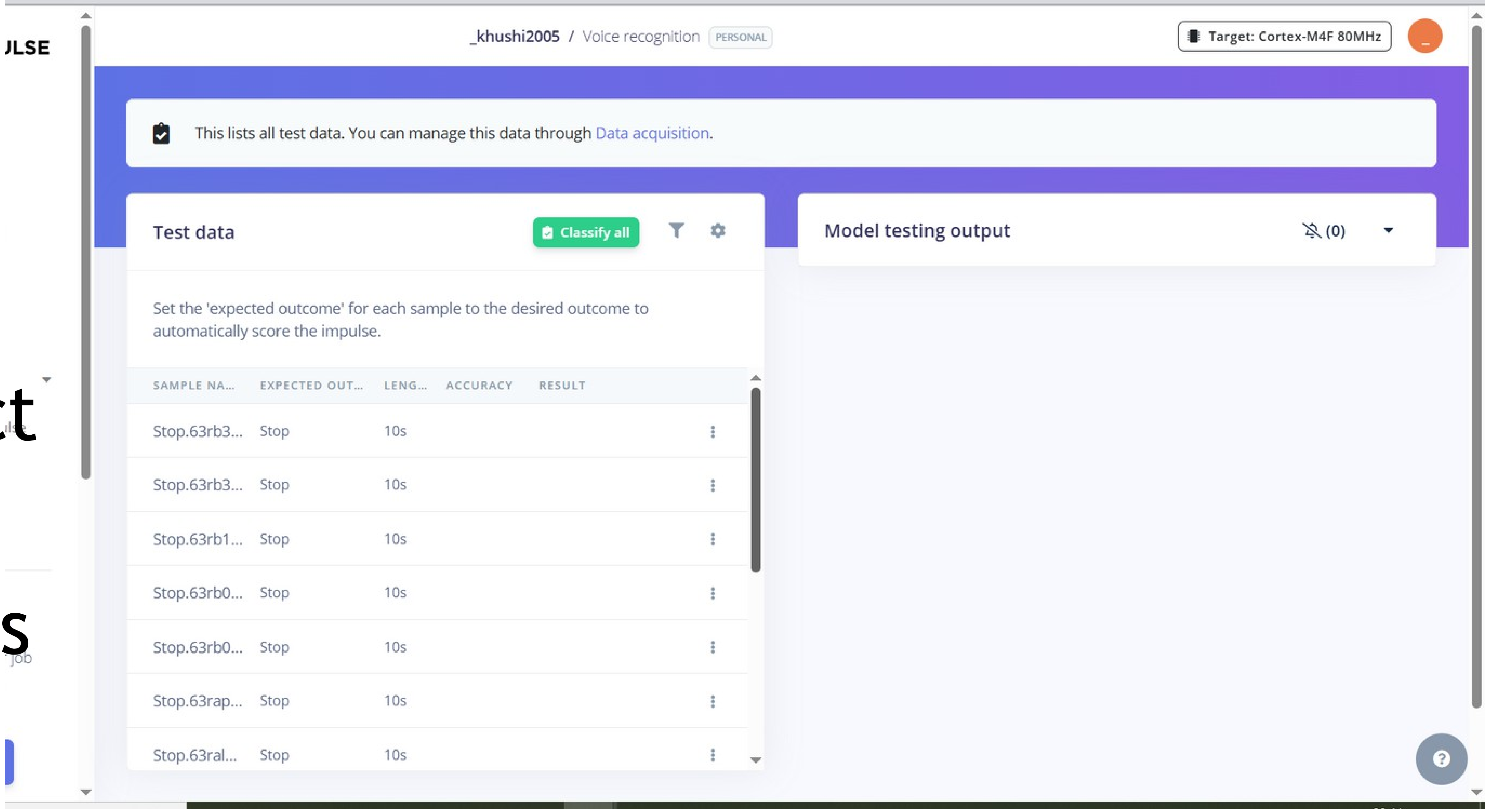


Click on **Classifier** in the left hand menu.

Change the number of training cycles to 50 and train the model.

CLASSIFYING TEST DATA

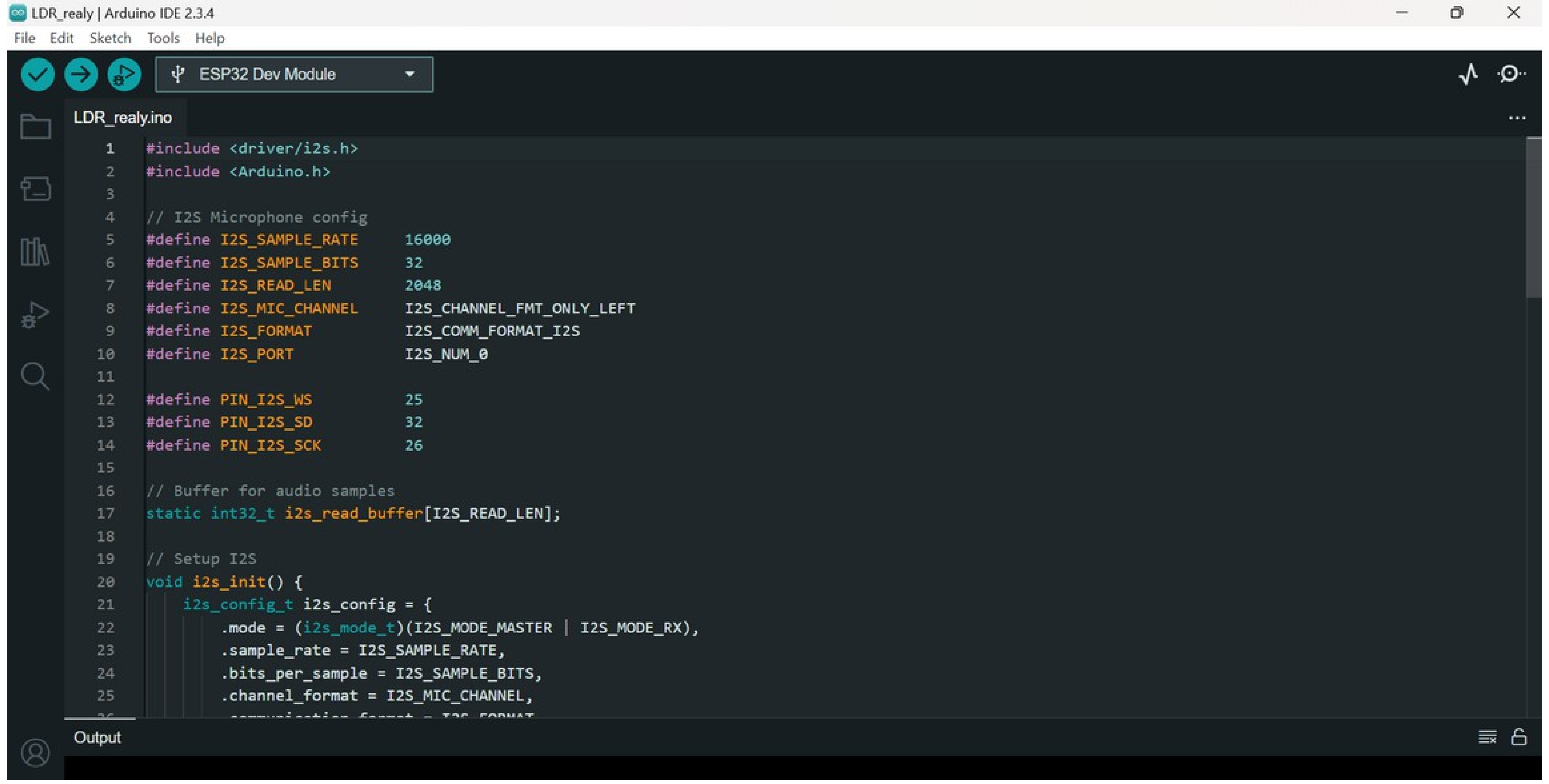
Add Test Data in Data Acquition windown if not present already. To run your model against the test set, head to **Model testing**, select all items and click **Classify selected**. Click the three dots (⋮)next to a sample and select Show classification.



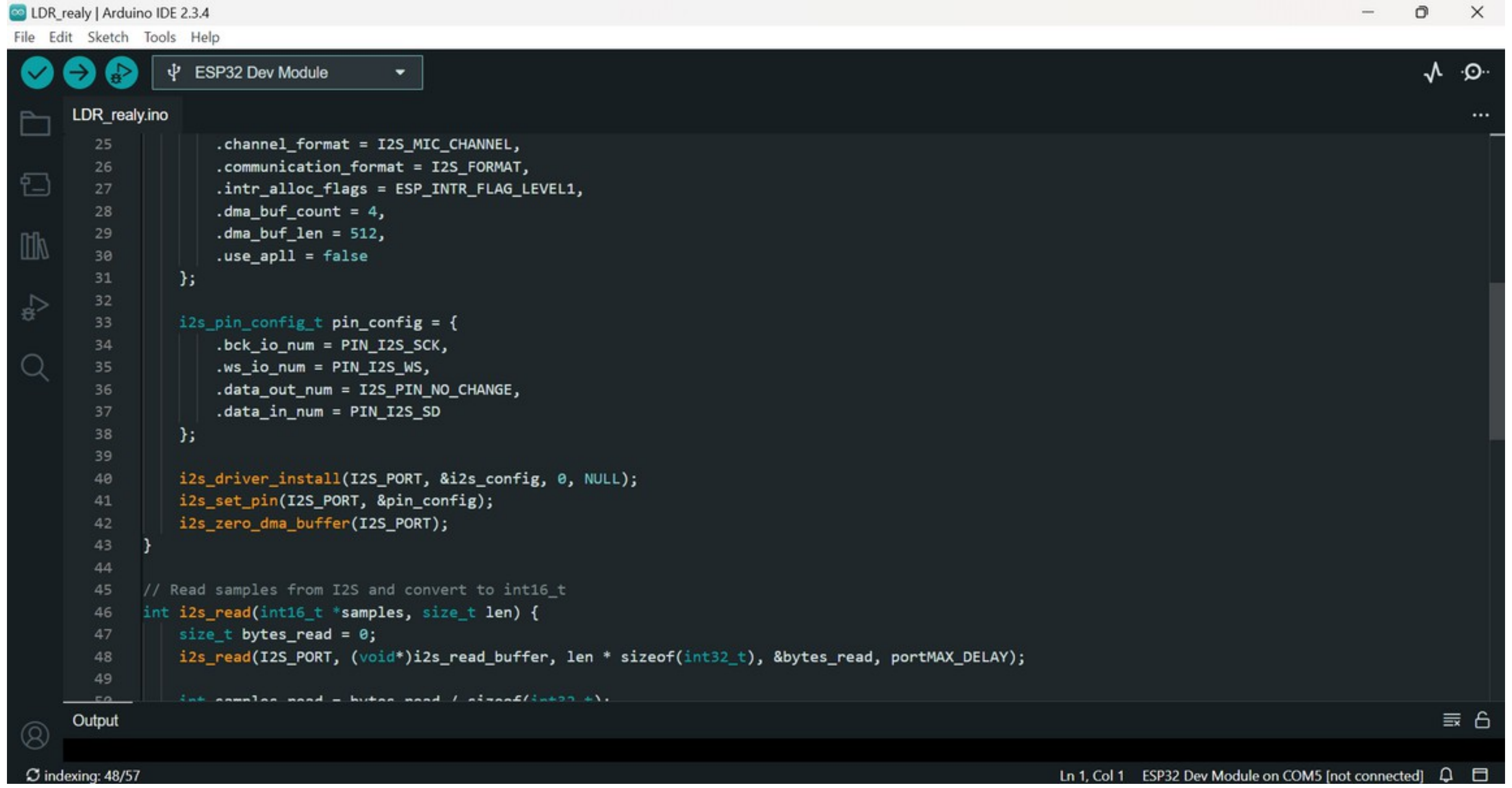
DEVELOPMENT

Add the given deployment code in your Arduino IDE and and then start running your model.As you will say something the model will predict it and will display in the serial monitor.

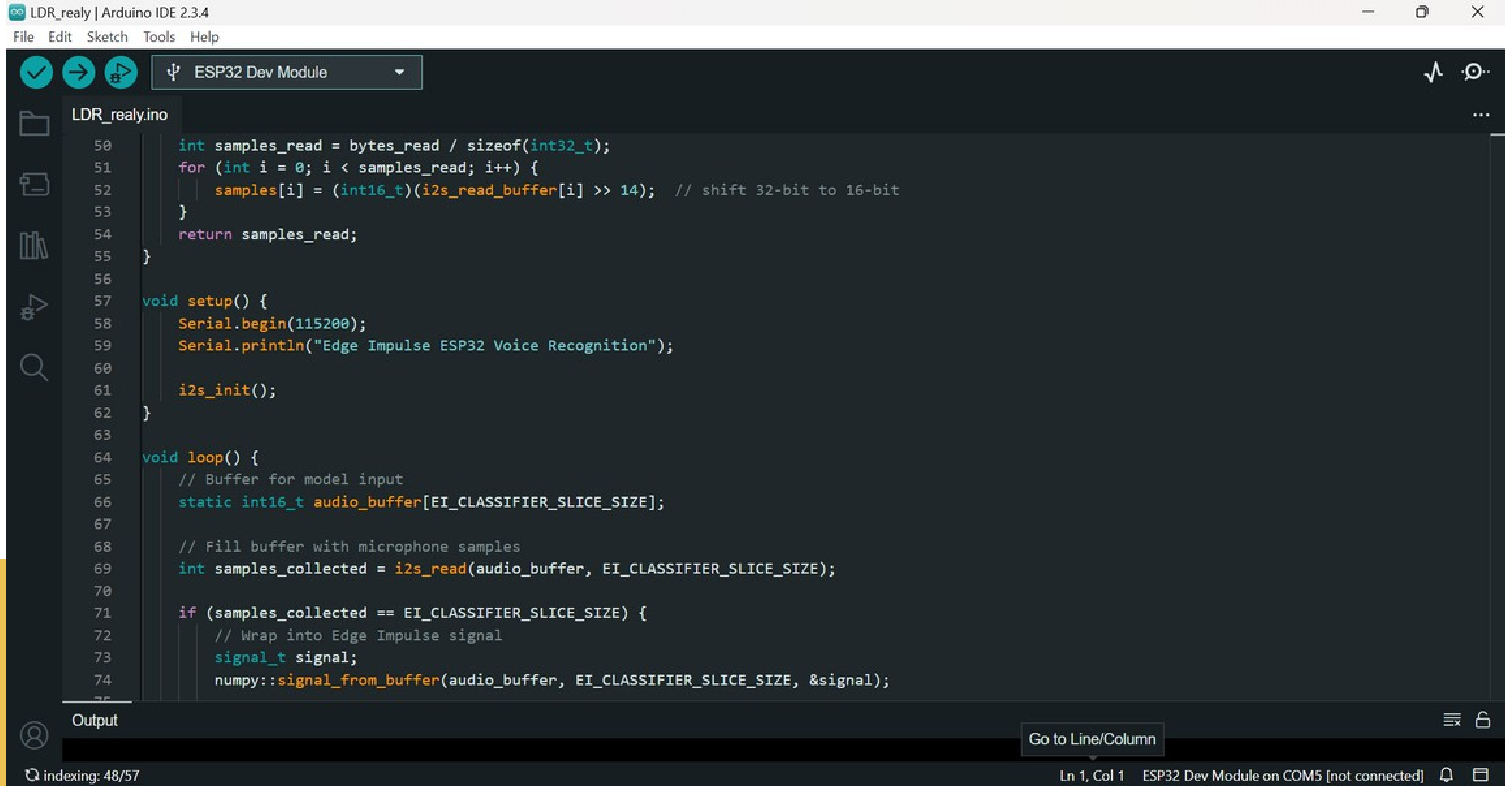
DEPLOYMENT CODE



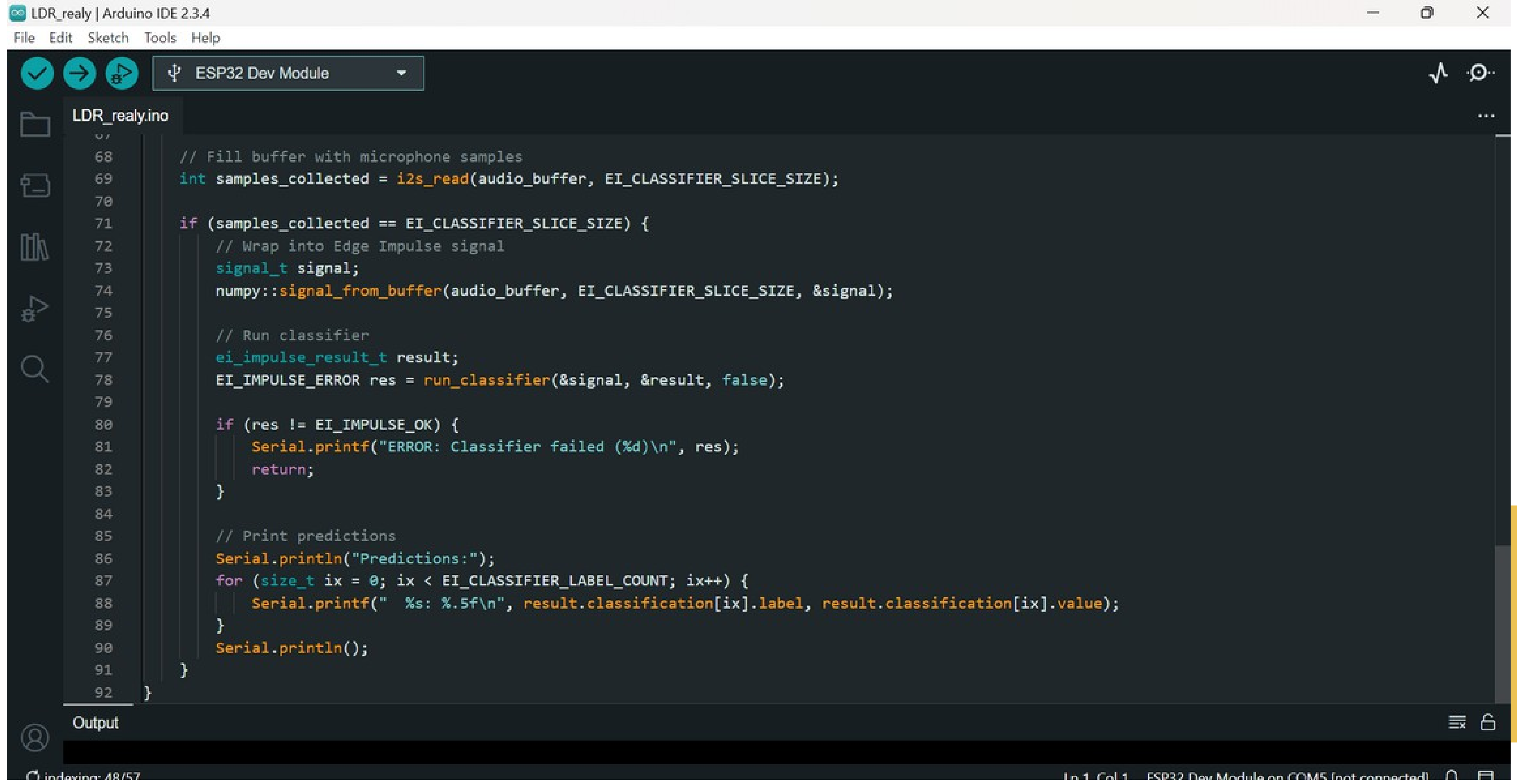
```
#include <driver/i2s.h>
#include <Arduino.h>

// I2S Microphone config
#define I2S_SAMPLE_RATE     16000
#define I2S_SAMPLE_BITS     32
#define I2S_READ_LEN        2048
#define I2S_MIC_CHANNEL     I2S_CHANNEL_FMT_ONLY_LEFT
#define I2S_FORMAT          I2S_COMM_FORMAT_I2S
#define I2S_PORT            I2S_NUM_0

#define PIN_I2S_WS          25
#define PIN_I2S_SD          32
#define PIN_I2S_SCK         26

// Buffer for audio samples
static int32_t i2s_read_buffer[I2S_READ_LEN];

// Setup I2S
void i2s_init() {
    i2s_config_t i2s_config = {
        .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
        .sample_rate = I2S_SAMPLE_RATE,
        .bits_per_sample = I2S_SAMPLE_BITS,
        .channel_format = I2S_MIC_CHANNEL,
```



```
        .channel_format = I2S_MIC_CHANNEL,
        .communication_format = I2S_FORMAT,
        .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
        .dma_buf_count = 4,
        .dma_buf_len = 512,
        .use_apll = false
    };

    i2s_pin_config_t pin_config = {
        .bck_io_num = PIN_I2S_SCK,
        .ws_io_num = PIN_I2S_WS,
        .data_out_num = I2S_PIN_NO_CHANGE,
        .data_in_num = PIN_I2S_SD
    };

    i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);
    i2s_set_pin(I2S_PORT, &pin_config);
    i2s_zero_dma_buffer(I2S_PORT);
}

// Read samples from I2S and convert to int16_t
int i2s_read(int16_t *samples, size_t len) {
    size_t bytes_read = 0;
    i2s_read(I2S_PORT, (void*)i2s_read_buffer, len * sizeof(int32_t), &bytes_read, portMAX_DELAY);

```



```
    int samples_read = bytes_read / sizeof(int32_t);
    for (int i = 0; i < samples_read; i++) {
        samples[i] = (int16_t)(i2s_read_buffer[i] >> 14);  // shift 32-bit to 16-bit
    }
    return samples_read;
}

void setup() {
    Serial.begin(115200);
    Serial.println("Edge Impulse ESP32 Voice Recognition");

    i2s_init();
}

void loop() {
    // Buffer for model input
    static int16_t audio_buffer[EI_CLASSIFIER_SLICE_SIZE];

    // Fill buffer with microphone samples
    int samples_collected = i2s_read(audio_buffer, EI_CLASSIFIER_SLICE_SIZE);

    if (samples_collected == EI_CLASSIFIER_SLICE_SIZE) {
        // Wrap into Edge Impulse signal
        signal_t signal;
        numpy::signal_from_buffer(audio_buffer, EI_CLASSIFIER_SLICE_SIZE, &signal);
```



```
    // Fill buffer with microphone samples
    int samples_collected = i2s_read(audio_buffer, EI_CLASSIFIER_SLICE_SIZE);

    if (samples_collected == EI_CLASSIFIER_SLICE_SIZE) {
        // Wrap into Edge Impulse signal
        signal_t signal;
        numpy::signal_from_buffer(audio_buffer, EI_CLASSIFIER_SLICE_SIZE, &signal);

        // Run classifier
        ei_impulse_result_t result;
        EI_IMPULSE_ERROR res = run_classifier(&signal, &result, false);

        if (res != EI_IMPULSE_OK) {
            Serial.printf("ERROR: Classifier failed (%d)\n", res);
            return;
        }

        // Print predictions
        Serial.println("Predictions:");
        for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
            Serial.printf("  %s: %.5f\n", result.classification[ix].label, result.classification[ix].value);
        }
        Serial.println();
    }
}
```

THANK YOU