IE417/EL530-Introduction to Embedded Artificial Intelligence

Lab 1

Voice Controlling LEDs with Edge Impulse

Group name : Embedded Minds

Ruturajsinh Chauhan(202101146)

Devansh Shrimali(202101492)

Raj Chauhan(202101049)

Jalp Patel(202101267)

1.Introduction

- Keyword spotting (KWS) is an essential technology used to enable hands-free interaction with devices in various daily-life applications. Popular voice assistants like "OK Google," "Alexa," "Hey Siri," and "Cortana" utilize KWS to detect specific wake-up phrases before interacting with the device.

- In this lab, we explore the application of KWS through Edge Impulse by building an application that controls a light-emitting diode (LED) based on voice commands. The commands include specifying the LED color (red, green, or blue) and the number of times it should blink (one, two, or three times). This TinyML application can be utilized in smart educational toys to teach both color and number vocabulary, offering a privacy-secure solution since it does not require internet connectivity.

2. Steps Performed

2.1 Dataset Preparation

We started by preparing the dataset, collecting audio data for the commands "blue," "red," "green," and "idle state" using a Arduino Nano BLE 33 Sense. The dataset included various utterances for each command to ensure diversity and accuracy during the training process.

2.2 Feature Extraction

The collected audio samples were processed using Mel-frequency cepstral coefficients (MFCC), a widely used feature extraction method for speech recognition. This step involved converting the raw audio data into a format that could be efficiently used by the neural network model.

2.3 Model Design and Training

We designed a neural network (NN) model and trained it using the Edge Impulse platform. During the training phase, we achieved a remarkable accuracy of 94.4%, which indicated the model's effectiveness in recognizing the specified voice commands.

2.4 Deployment

After successfully training and optimizing the model, we deployed it to the Arduino Nano BLE 33 Sense. We used the following library for deployment:

After successfully training and optimizing the model, we deployed it to the Arduino Nano BLE 33 Sense. We used the following library for deployment:

- Library Name: ei-hack110011-project1-arduino-1.0.10

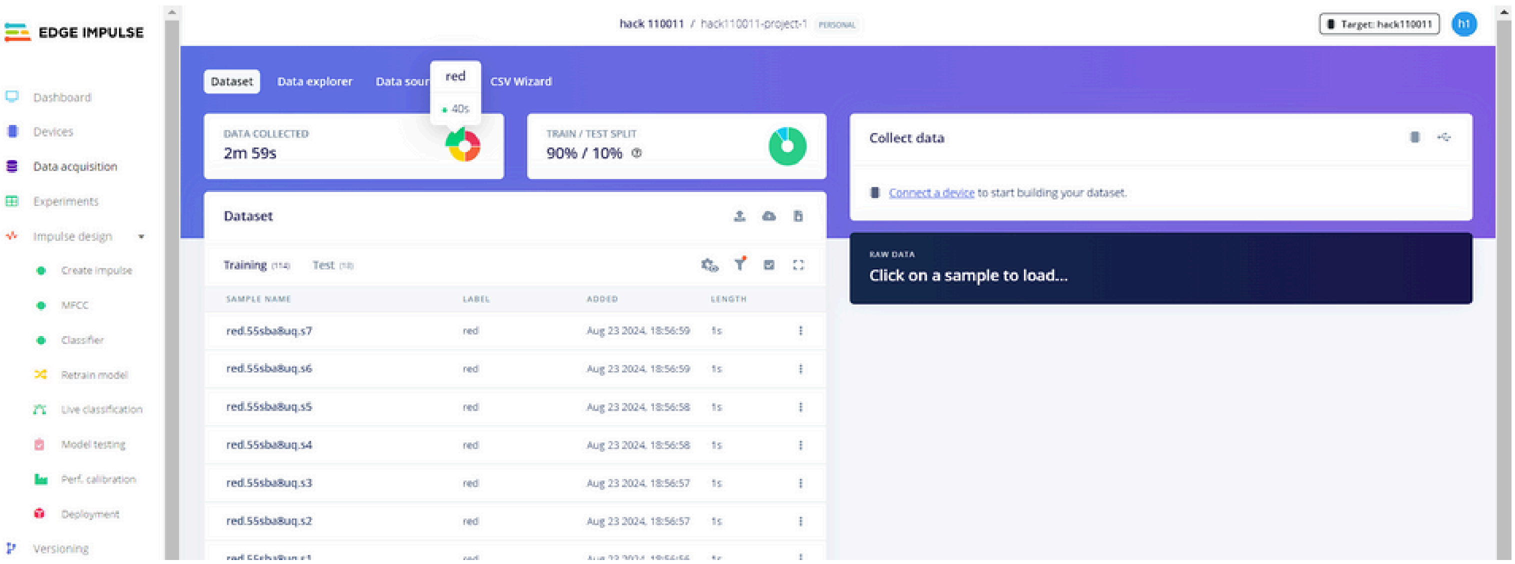
We imported this library into the Arduino IDE and uploaded the code to the Arduino Nano BLE 33 Sense board using the following steps:

1. Arduino IDE ->File -> Examples -> hack110011-project-1_inferencing ->nano_ble33_sense -> nano_ble33_sense_microphone_continuous

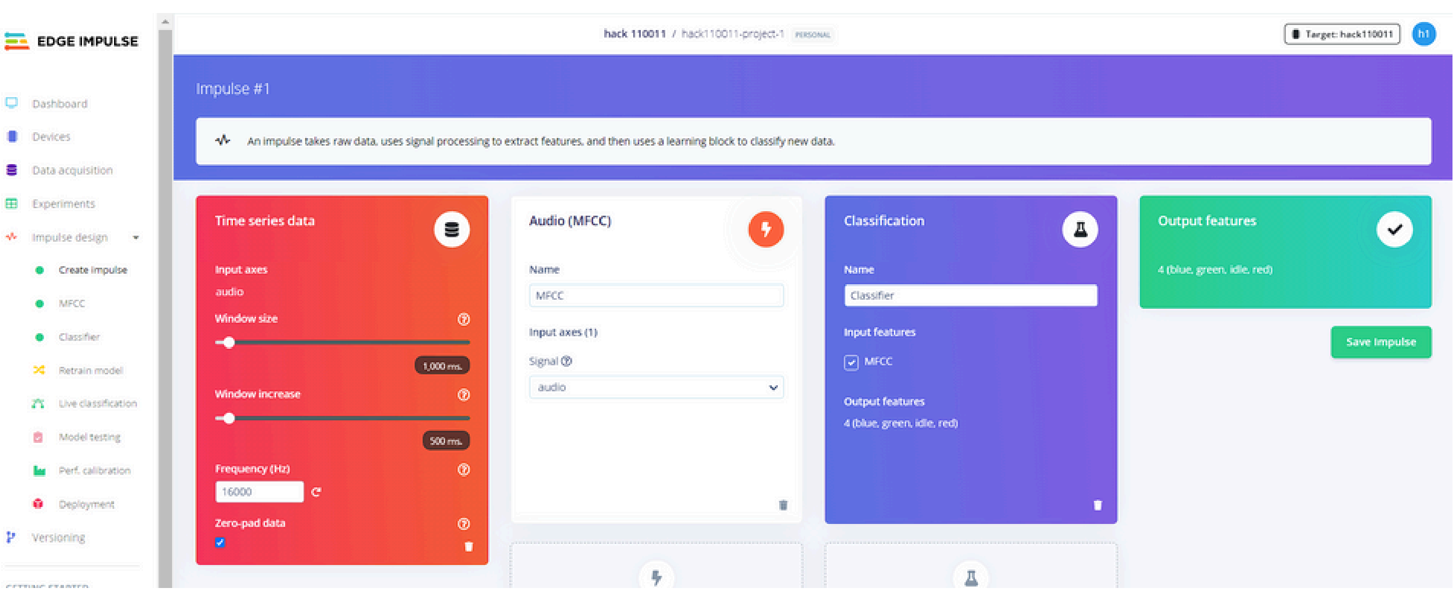
2.Uploaded the code to the Arduino Nano BLE 33 Sense and captured the output on the serial monitor.

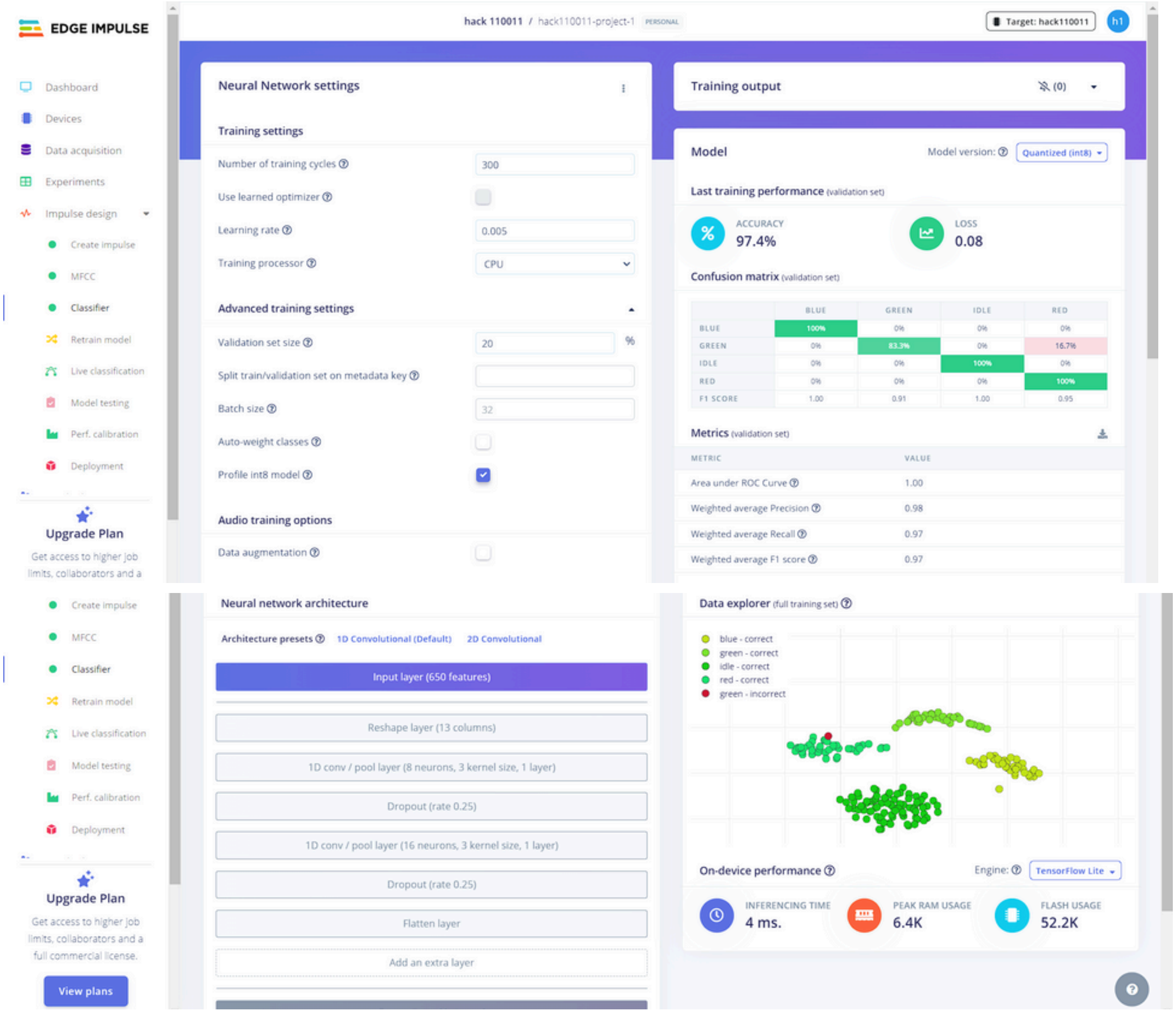
Output images :

Data Acquization

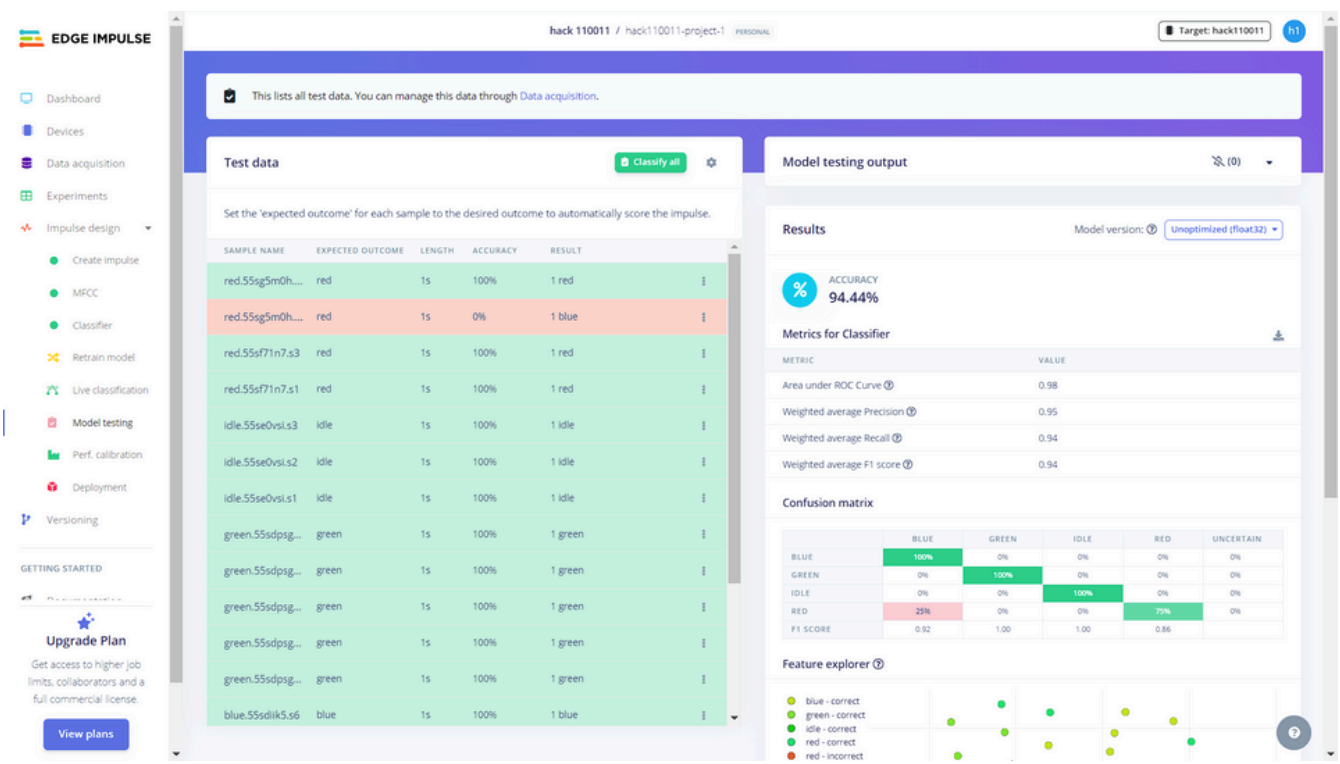


Model train and classification





Model testing



3. Output and Results

The final trained model achieved a 94.4% accuracy during testing. The deployment on the Arduino Nano BLE 33 Sense allowed us to control the LED based on voice commands, demonstrating the practical application of the developed KWS system.

4. Conclusion

This lab successfully demonstrated the development of an end-to-end KWS application using Edge Impulse. The project provided hands-on experience with audio data acquisition, feature extraction using MFCC, neural network training, model optimization, and deployment on low-power hardware like the Arduino Nano BLE 33 Sense. The achieved accuracy of 94.4% showcases the efficiency of the developed model in real-time voice-controlled applications.

Video Link

Serial monitor output:
https://drive.google.com/file/d/1bGjv4Z18J6YCAXL-i-Lzxc3rfyQZaonA/view

Demo Video:
https://drive.google.com/file/d/1Og17w0UvQneIYDnbpW71T3LwHNhTiE-K/view?usp=sharing

code:

```
#define EIDSP_QUANTIZE_FILTERBANK   0

/**
  Define the number of slices per model window. E.g. a model window of 1000 ms
  with slices per model window set to 4. Results in a slice size of 250 ms.
  For more info: https://docs.edgeimpulse.com/docs/continuous-audio-sampling
*/
#define EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW 3

/* Includes ---------------------------------------------------------------- */
#include <PDM.h>
#include <hack110011-project-1_inferencing.h>

#define RED     22
#define BLUE    24
#define GREEN   23
#define LED_PWR 25

/** Audio buffers, pointers and selectors */
typedef struct {
  signed short *buffers[2];
  unsigned char buf_select;
  unsigned char buf_ready;
  unsigned int buf_count;
  unsigned int n_samples;
} inference_t;

static inference_t inference;
static bool record_ready = false;
static signed short *sampleBuffer;
static bool debug_nn = false; // Set this to true to see e.g. features generated from the raw signal
static int print_results = -(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW);

/**
  @brief      Arduino setup function
*/
void setup()
{
  pinMode(RED, OUTPUT);
  pinMode(BLUE, OUTPUT);
  pinMode(GREEN, OUTPUT);
```

```
  pinMode(GREEN, OUTPUT);
  //pinMode(LED_PWR, OUTPUT);

  // put your setup code here, to run once:
  Serial.begin(115200);

  Serial.println("Edge Impulse Inferencing Demo");

  // summary of inferencing settings (from model_metadata.h)
  ei_printf("Inferencing settings:\n");
  ei_printf("\tInterval: %.2f ms.\n", (float)EI_CLASSIFIER_INTERVAL_MS);
  ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
  ei_printf("\tSample length: %d ms.\n", EI_CLASSIFIER_RAW_SAMPLE_COUNT / 16);
  ei_printf("\tNo. of classes: %d\n", sizeof(ei_classifier_inferencing_categories) /
          sizeof(ei_classifier_inferencing_categories[0]));

  run_classifier_init();
  if (microphone_inference_start(EI_CLASSIFIER_SLICE_SIZE) == false) {
    ei_printf("ERR: Failed to setup audio sampling\r\n");
    return;
  }
}

/**
 * @brief      Arduino main function. Runs the inferencing loop.
 */
void loop()
{
  bool m = microphone_inference_record();
  if (!m) {
    ei_printf("ERR: Failed to record audio...\n");
    return;
  }

  signal_t signal;
  signal.total_length = EI_CLASSIFIER_SLICE_SIZE;
  signal.get_data = &microphone_audio_signal_get_data;
  ei_impulse_result_t result = {0};
```

```
  EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, debug_nn);
  if (r != EI_IMPULSE_OK) {
    ei_printf("ERR: Failed to run classifier (%d)\n", r);
    return;
  }

  if (result.classification[0].value >= 0.7) {
    digitalWrite(BLUE, LOW);
    digitalWrite(GREEN, HIGH);
    digitalWrite(RED, HIGH);
  }
  else if (result.classification[1].value >= 0.7) {
    digitalWrite(BLUE, HIGH);
    digitalWrite(GREEN, LOW);
    digitalWrite(RED, HIGH);
  }
  else if (result.classification[2].value >= 0.7) {
    digitalWrite(BLUE, HIGH);
    digitalWrite(GREEN, HIGH);
    digitalWrite(RED, HIGH);
  }
  else if (result.classification[3].value >= 0.7) {
    digitalWrite(BLUE, HIGH);
    digitalWrite(GREEN, HIGH);
    digitalWrite(RED, LOW);
  }

  if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)) {
    // print the predictions
    ei_printf("Predictions ");
    ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
              result.timing.dsp, result.timing.classification, result.timing.anomaly);
    ei_printf(": \n");
    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
      ei_printf("    %s: %.5f\n", result.classification[ix].label,
                result.classification[ix].value);
    }
#if EI_CLASSIFIER_HAS_ANOMALY == 1
    ei_printf("    anomaly score: %.3f\n", result.anomaly);
#endif
```

```
    print_results = 0;
  }
}

/**
   @brief      Printf function uses vsnprintf and output using Arduino Serial
   @param[in]  format     Variable argument list
*/
void ei_printf(const char *format, ...) {
  static char print_buf[1024] = { 0 };

  va_list args;
  va_start(args, format);
  int r = vsnprintf(print_buf, sizeof(print_buf), format, args);
  va_end(args);

  if (r > 0) {
    Serial.write(print_buf);
  }
}

/**
   @brief      PDM buffer full callback
               Get data and call audio thread callback
*/
static void pdm_data_ready_inference_callback(void)
{
  int bytesAvailable = PDM.available();

  // read into the sample buffer
  int bytesRead = PDM.read((char *)&sampleBuffer[0], bytesAvailable);

  if (record_ready == true) {
    for (int i = 0; i<bytesRead >> 1; i++) {
      inference.buffers[inference.buf_select][inference.buf_count++] = sampleBuffer[i];

      if (inference.buf_count >= inference.n_samples) {
        inference.buf_select ^= 1;
        inference.buf_count = 0;
        inference.buf_ready = 1;
      }
```

```
            }
        }
    }
}

/**
  @brief      Init inferencing struct and setup/start PDM
  @param[in]  n_samples  The n samples
  @return     { description_of_the_return_value }
*/
static bool microphone_inference_start(uint32_t n_samples)
{
    inference.buffers[0] = (signed short *)malloc(n_samples * sizeof(signed short));

    if (inference.buffers[0] == NULL) {
        return false;
    }

    inference.buffers[1] = (signed short *)malloc(n_samples * sizeof(signed short));

    if (inference.buffers[0] == NULL) {
        free(inference.buffers[0]);
        return false;
    }

    sampleBuffer = (signed short *)malloc((n_samples >> 1) * sizeof(signed short));

    if (sampleBuffer == NULL) {
        free(inference.buffers[0]);
        free(inference.buffers[1]);
        return false;
    }

    inference.buf_select = 0;
    inference.buf_count = 0;
    inference.n_samples = n_samples;
    inference.buf_ready = 0;

    // configure the data receive callback
    PDM.onReceive(&pdm_data_ready_inference_callback);

```

```
    PDM.setBufferSize((n_samples >> 1) * sizeof(int16_t));

    // initialize PDM with:
    // - one channel (mono mode)
    // - a 16 kHz sample rate
    if (!PDM.begin(1, EI_CLASSIFIER_FREQUENCY)) {
        ei_printf("Failed to start PDM!");
    }

    // set the gain, defaults to 20
    PDM.setGain(127);

    record_ready = true;

    return true;
}

/**
 * @brief      Wait on new data
 * @return     True when finished
 */
static bool microphone_inference_record(void)
{
    bool ret = true;

    if (inference.buf_ready == 1) {
        ei_printf(
            "Error sample buffer overrun. Decrease the number of slices per model window "
            "(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)\n");
        ret = false;
    }

    while (inference.buf_ready == 0) {
        delay(1);
    }

    inference.buf_ready = 0;

    return ret;
}
```

```
/**
    Get raw audio signal data
*/
static int microphone_audio_signal_get_data(size_t offset, size_t length, float *out_ptr)
{
  numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1][offset], out_ptr, length);

  return 0;
}

/**
    @brief      Stop PDM and release buffers
*/
static void microphone_inference_end(void)
{
  PDM.end();
  free(inference.buffers[0]);
  free(inference.buffers[1]);
  free(sampleBuffer);
}

#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR != EI_CLASSIFIER_SENSOR_MICROPHONE
#error "Invalid model for current sensor."
#endif
```