Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0624 – Laboratorio de Microcontroladores
II ciclo 2023

Laboratorio 5

Arduino Nano BLE 33: Tiny Machine, Iot

Kenny Wu Wen C08592
Oscar Fallas B92861
Grupo 01

Profesor: MSc. Marco Villalta Fallas

20 de noviembre de 2023

Índice

Índice de figuras

1. Resumen

En este laboratorio se van a utilizar el microcontrolador Arduino Nano BLE 33 Sense Lite. Se utilizara su micrófono integrado para obtener datos para entrenar utilizando tiny machine learning. Luego, se importara el modelo para probarlo con el microcontrolador con el fin de detectar ciertas palabras/oraciones con el micrófono y mandando esta información a la computadora por puerto UART/Serial para enviarlo a Thingsboard de EIE.

El repositorio para el laboratorio es el siguiente: GIT

2. Nota teórica

2.1. Tiny Machine Learning Kit

Este es el kit que se utilizara para el laboratorio. Este incluye todo lo necesario para entrar al mundo de Tiny Machine Learning. El kit incluye [4]:

- Arduino Nano BLE Sense
- Cámara OV7675
- Placa Arduino Tiny Machine Learning
- Cable USB a Micro USB

Al conectar todos los componentes quedaría de la siguiente manera:

Figura 1: Kit TML completo [1]

2.2. Arduino Nano BLE 33

Esta es una placa de de Hardaware y Software libre con microcontrolador programable, pertenece a la familia de AVR. En sí, el Arduino Nano BLE 33 Sense Lite es una placa embebida

que permiten empezar a aprender en el mundo de Machine Learning, cuál posee una placa de desarrollo compacta y versátil con 11 distintos sensores, cuenta con el microcontrolador principal el nRF52840 de Nordic Semiconductor. Entre sus sensores más importantes tenemos [5]:

- IMU LSM9DS1: Giroscopio, acelerómetro y magnetómetro.
- APDS9960: Sensor de proximidad, colo RGB, intensidad de luz y detector de gestos.
- MP34DT05: micrófono
- LPS22HB: Sensor de Presión y temperatura
- Bluetooth: Versión 5.0 con capacidad de modo Low Energy
- HTS221: Sensor de temperatura y humedad

También tenemos las especificaciones resumidas en la siguiente tabla:

Cuadro 1: Especificaciones del Arduino Nano BLE sense lite [5]

Microcontroller	nRF52840	
USB connector	Micro USB	
Pins	**Built-in LED Pin**	13
	Digital I/O Pins	14
	Analog input pins	8
	PWM pins	5
	External interrupts	All digital pins
Connectivity	**Bluetooth®**	NINA-B306
Sensors	**IMU**	LSM9DS
	Microphone	MP34DT05
	Gesture, light, proximity	APDS9960
	Barometric pressure	LPS22HB
	Temperature, humidity	HTS221
Communication	**UART**	RX/TX
	I2C	A4 (SDA), A5 (SCL)
	SPI	D11 (COPI), D12 (CIPO), D13 (SCK). Use any GPIO for Chip Select (CS).
Power	**I/O Voltage**	3.3V
	Input voltage (nominal)	5-18V
	DC Current per I/O Pin	10 mA
Clock speed	**Processor**	nRF52840 64MHz
Memory	**nRF52840**	256 KB SRAM, 1MB flash
Dimensions	**Weight**	5gr
	Width	18 mm
	Length	45 mm

2.3. nRF52840

La placa del Arduino Nano BLE 33 se basa en el microcontrolador nRF52840 que corresponde a un SoC de High-end Multiprotocol Bluetooth Low Energy. El procesador está construido bajo un procesador ARM Cortex-M4 con una unidad de punto flotante. Entre sus características más importantes a destacar tenemos [6]:

- Protocolos de comunicación integrados de alta velocidad SPI, I2C, QSPI.
- Voltaje de operación de 1.7V a 5.5V
- Bluetooth 5.3
- 1 MB Flash + 256 KB RAM

En la figura **??**, se puede observar la diferencia entre cada versión de la placa.

	nRF52805	nRF52810	nRF52811	nRF52820	nRF52832	nRF52833	nRF52840	nRF5340
Bluetooth 5.3	X	X	X	X	X	X	X	X
Bluetooth 2 Mpbs	X	X	X	X	X	X	X	X
Bluetooth Long Range			X	X		X	X	X
Bluetooth Direction Finding			X	X		X		X
Bluetooth LE Audio								X
Bluetooth mesh				X	X	X	X	X
Thread			X	X		X	X	X
Zigbee				X		X	X	X
Matter							X	X

Figura 2: Tabla comparativa entre las versiones de la placa

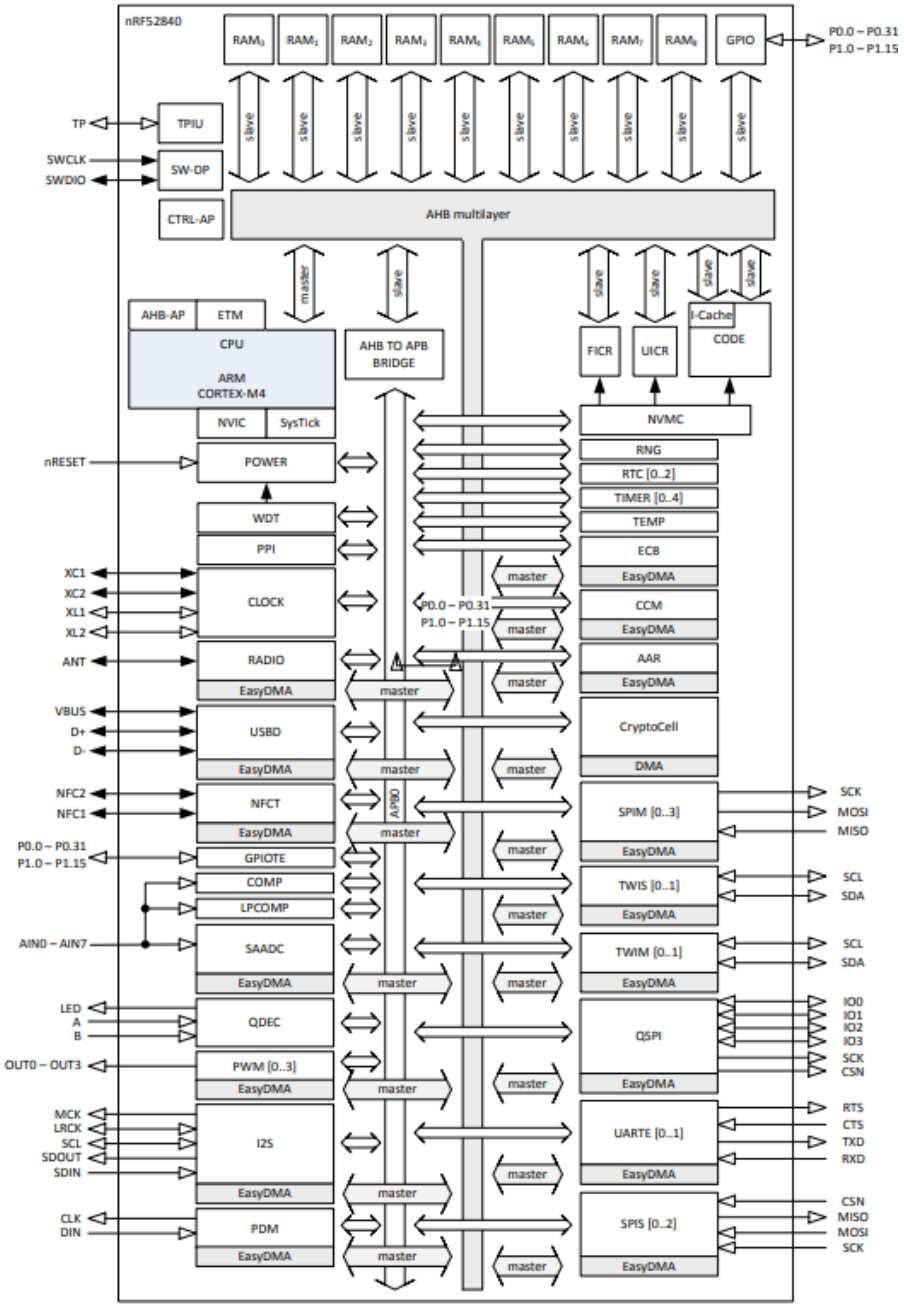

Figura 3: Diagrama nRF52840

La tabla de especificaciones la tenemos en la figura 4

Specifications

Protocol support	Bluetooth 5.3/802.15.4/ANT/2.4 GHz proprietary
Microprocessor	64 MHz 32-bit Arm Cortex-M4 with FPU
Memory	1 MB Flash + 256 KB RAM
On-air data rate	Bluetooth LE: 2 Mbps/1 Mbps/500 kbps/125 kbps 802.15.4: 250 kbps 2.4 GHz proprietary: 2 Mbps/ 1 Mbps
TX power	Programmable from +8 dBm to -20 dBm in 4 dB steps
Sensitivity	Bluetooth LE: -103 dBm at 125 kbps -95 dBm at 1 Mbps 802.15.4: -100 dBm at 250 kbps 2.4 GHz: -93 dBm at 1 Mbps -89 dBm at 2 Mbps
Radio current consumption DC/DC at 3 V	16.40 mA at +8 dBm TX power, 6.40 mA at 0 dBm TX power, 6.26 mA in RX at 1 Mbps
Oscillators	64 MHz from 32 MHz external crystal or internal 32 kHz from crystal, RC or synthesized
System current consumption DC/DC at 3 V	0.4 µA in System OFF, no RAM retention 1.86 µA in System OFF, full RAM retention 0.97 µA in System ON, no RAM retention 2.35 µA in System ON, full RAM retention 3.16 µA in System ON, full RAM retention and RTC
Hardware security	128-bit AES CCM, ECB, AAR
Security subsystem	Arm TrustZone CryptoCell 310
Digital interfaces	USB 2.0, 4 × SPI master/slave, 2 × TWI master/slave, 2 × UART, 4 × PWM, QPSI, I²S, PDM, QDEC
Analog interfaces	12-bit 200 ksps ADC, GP comparator, LP comparator
Peripherals	5 × 32 bit timer/counter, 3 × 24 real-time counter, 20 × PPI channels, 4 × GPIOTE, temperature sensor, watchdog timer, RNG
NFC	NFC-A tag
Voltage supply	1.7 to 5.5 V LDO or DC/DC
Package options	7 × 7 aQFN73 with 48 GPIOs 6 × 6 QFN48 with 30 GPIOs 3.5 × 3.6 WLCSP94 with 48 GPIOs

Figura 4: Especificaciones de la placa nRF52840

2.4. TinyML

Dado los limitados recursos de los microcontroladores tales como procesador, almacenamiento, memoria RAM se creó un nuevo concepto asociado al "Machine Learning". TinyML permite realizar análisis de los datos comunes captados por los microcontroladores, entre sus aplicaciones podemos encontrar: reconocimiento de voz, monitoreo de seguridad y salud.

2.4.1. Red Neuronal

Una red neuronal consisten en una entrada de neuronas anidadas(nodos, o unidades), una dos o incluso tres capas de dichas neuronas, y una capa final de nodos. Cada conexión se asocia una con la otra mediante un parámetro numérico conocido como peso [2].

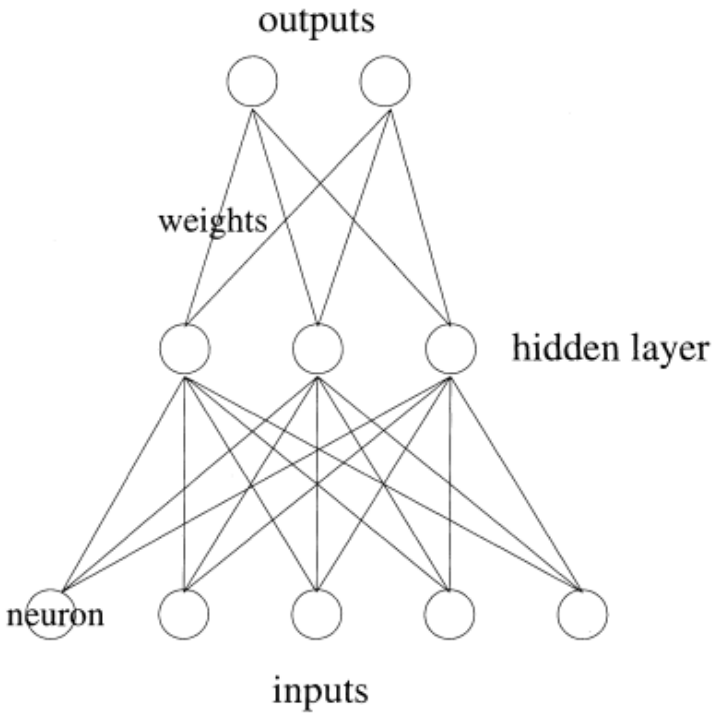

Figura 5: Diagrama de típica red neuronal. [2]

Las redes neuronales artificiales están basadas en la función del cerebro humano, el cuál mediante análisis de patrones y/o modelos proporciona un aprendizaje basado en experiencias. A este proceso se le conoce como aprendizaje automático, utilizando gran cantidad de datos y parámetros de calificación se logra entrenar un modelo para que reconozca, o prediga acontecimientos basados en el pasado.

2.4.2. Edge Impulse

Edge Impulse es una herramienta web que permite el entrenamiento de modelos y optimización de librerías para el despliegue en cualquier dispositivo como low-power MCU. La herramienta permite a los desarrolladores obtener data de varios fuentes, incluyendo sensores propios de hardware utilizado, bases de datos públicas o datos generados a través de simulaciones [3].

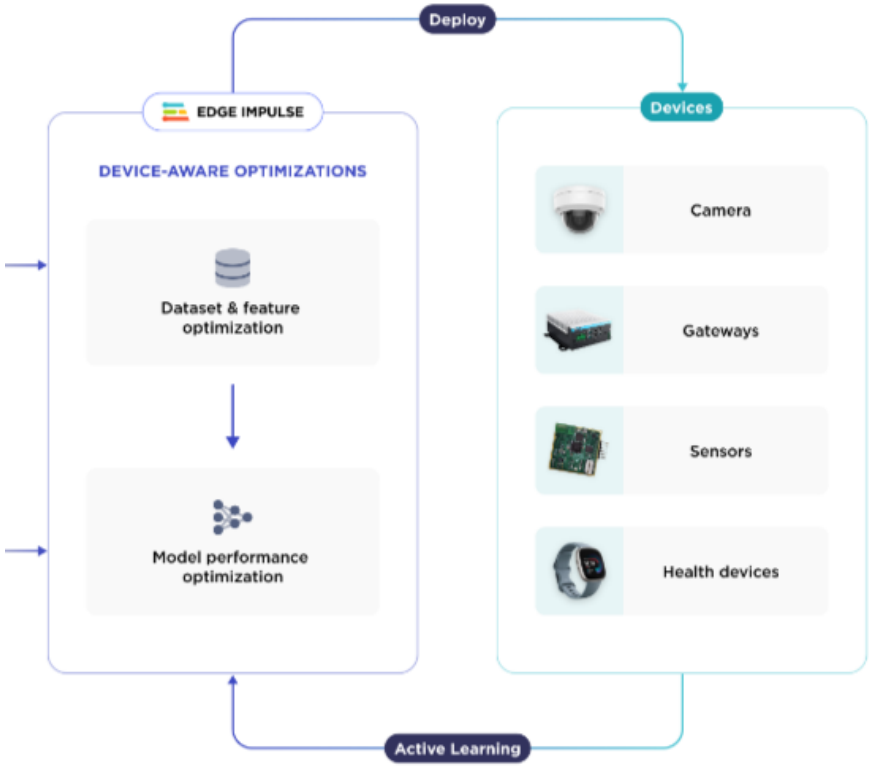

Figura 6: Diagrama de Edge Impulse. [3]

2.5. Grabar con micrófono del Arduino Nano

2.5.1. Firmware

Para esto vamos a necesitar las librerias de Adafruit Zero PDM library. Esta libreria nos permitirá convertir lo que recibe el microfono con PDM a PCM y esta información será enviada por UART a la computadora.

La libreria utilizada para utilizar el microfono del Arduino es el PDM.h, el cual tiene son los siguientes métodos:

- PMD.onReceive(función): Tiene como argumento una función o método a llamar cuando ocurre una interrupción que generalmente es causada cuando se recibe datos por el micrófono.
- PDM.setGain(entero): Tiene como argumento un entero que significa la ganancia del micrófono, por predeterminado esta en 20.
- PDM.begin(canal, frecuencia): Tiene dos argumentos, el primero para cantidad de canales y el segundo la frecuencia del micrófono que para el Arduino nano ble sense lite es de 16 KHz. Si este no se inicializa bien va a devolver un error.
- PDM.available(): Este devuelve una cantidad de bytes recibidos por el micrófono
- PDM.read(buffer, PDM.available()): Lee los datos y los guarda en el buffer

Se utilizara como configuración una frecuencia de 16 KHz y mono audio.
El código utilizado es el siguiente:

```
/*
This example reads audio data from the on-board PDM microphones, and
    prints
out the samples to the Serial console. The Serial Plotter built into the
Arduino IDE can be used to plot the audio data (Tools -> Serial Plotter)
Circuit:
- Arduino Nicla Vision, or
- Arduino Nano 33 BLE board, or
- Arduino Nano RP2040 Connect, or
- Arduino Portenta H7 board plus Portenta Vision Shield
This example code is in the public domain.
*/


#include <PDM.h>

// default number of output channels
static const char channels = 1;

// default PCM output frequency
static const int frequency = 16000;

// Buffer to read samples into, each sample is 16-bits
short sampleBuffer[1023];

// Number of audio samples read
volatile int samplesRead;

// Blinking
bool state = false;
int timeStart = 0;

void setup() {
Serial.begin(9600);
pinMode(LEDB, OUTPUT);

while (!Serial);

// Configure the data receive callback
PDM.onReceive(onPDMdata);

// Optionally set the gain
// Defaults to 20 on the BLE Sense and 24 on the Portenta Vision Shield
PDM.setGain(30);

// Initialize PDM with:
// - one channel (mono mode)
// - a 16 kHz sample rate for the Arduino Nano 33 BLE Sense
// - a 32 kHz or 64 kHz sample rate for the Arduino Portenta Vision
    Shield
if (!PDM.begin(channels, frequency)) {
  Serial.println("Failed to start PDM!");
  while (1);
```

```
}


}

void loop() {
// Wait for samples to be read
if (samplesRead) {
  // Print samples to the serial monitor or plotter
  for (int i = 0; i < samplesRead; i++) {
    if (channels == 2) {
      Serial.print("L:");
      Serial.print(sampleBuffer[i]);
      Serial.print("-R:");
      i++;
    }
    Serial.println(sampleBuffer[i]);

  }

  // Clear the read count
  samplesRead = 0;

  if (millis() - timeStart > sampleBuffer[2]) {
    digitalWrite(LEDB, state);
    state = !state;
  }
}
}

/**
Callback function to process the data from the PDM microphone.
NOTE: This callback is executed as part of an ISR.
Therefore using `Serial` to print messages inside this function isn't
   supported.
* */
void onPDMdata() {
// Query the number of available bytes
int bytesAvailable = PDM.available();

// Read into the sample buffer
PDM.read(sampleBuffer, bytesAvailable);

// 16-bit, 2 bytes per sample
samplesRead = bytesAvailable / 2;
}
```

2.6. PCM a .wav

Ahora con la información recibida por uno de los puertos de la computadora de forma serial vamos a tener que guardar esta información en un array para luego transformarla con la librería scipy.io.wavfile a un archivo .wav en la cual vamos a poder escuchar lo grabado con el micrófono.

Si se observa el codigo abajo, se tiene un samplerate igual a la frecuencia usado en el micrófono del Arduino que es de 16 KHz y además como se sabe que los datos recibidos son de 16bits, al transformarlo en .wav se toma cada dato como un entero de 16 bits con signo.

El código utilizado es:

```
from gettext import npgettext
import serial
import time
import csv
import numpy as np
import matplotlib, wave
matplotlib.use("tkAgg")
import matplotlib.pyplot as plt
ser = serial.Serial(
    port = 'COM3',\
    baudrate = 9600,\
    #parity = serial.PARITY_NONE,\
    #stopbits=serial.STOPBITS_ONE,\
    #bytesize=serial.EIGHTBITS,\
    #timeout=1\
)

filename = open('data2.csv','w',newline="")
output_file = csv.writer(filename)

#plot_window = 20
#y_var = np.array(np.zeros([plot_window]))
#plt.ion()
#fig, ax = plt.subplots()
#line, = ax.plot(y_var)

from scipy.io.wavfile import write
import numpy as np
samplerate = 16000; fs = 100
t = np.linspace(0., 1., samplerate)
amplitude = np.iinfo(np.int16).max
data = amplitude * np.sin(2. * np.pi * fs * t)
write("example.wav", samplerate, data.astype(np.int16))
sound = []
print("Connection succeded")
while (1):
    try:
        data = ser.readline()
        data = int(data[0:len(data)-2].decode("utf-8").replace("\n",""))
        output_file.writerow([data])
        sound.append(data)
        #print(data)
    except:
        break
    #time.sleep(0.5)
    #y_var = np.append(y_var, float(data))
    #y_var = y_var[1:plot_window+1]
    #line.set_ydata(y_var)
    #ax.relim()
```

```
    #ax.autoscale_view()
    #fig.canvas.draw()
    #fig.canvas.flush_events()
sound = np.array(sound) # aumentar ganancia
write("example.wav", samplerate, sound.astype(np.int16))
```

2.7. Constructor del Modelo de ML

Como se explicó anteriormente, se utilizó la herramienta de Edge Impulse con el fin generar un modelo de aprendizaje automático que pudiera reconocer una frase a base de experiencia ingresada manualmente. Para realizar el modelo se realizó lo siguiente:

1. Grabar alrededor de 50 o más iteraciones de la frase seleccionada.

2. Mezclar las grabaciones realizas con ruido blanco y oscuro con el fin de mejorar el aprendizaje del modelo y proporcionar un mejor funcionamiento en ambientes no aislados. Para este se utilizó el repositorio disponible en el enlace el cuál mediante un script de python mezcla el audio introducido con ruido ya incorporado.

3. Subir los audios preparados a la plataforma de Edge Impulse. También era necesario subir ejemplos de ruido y de audio desconocido, igualmente proporcionados por el repositorio mencionado para que el modelo pudiese reconocer cuando no es ninguna de las frases seleccionadas. Se puede observar que la data total es de alrededor de las 2 horas.

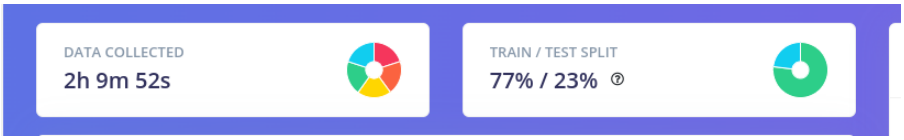

Figura 7: Cantidad de datos subidos

4. Una vez que la data se recolectó se procede a escoger un modelo de aprendizaje, en este caso se utilizó el modelo MFCC que es especial para el reconocimiento de audio. Dicho modelo utiliza algoritmo no lineal llamado Mel-scale[3]. A continuación los parámetros seleccionados para el modelo.

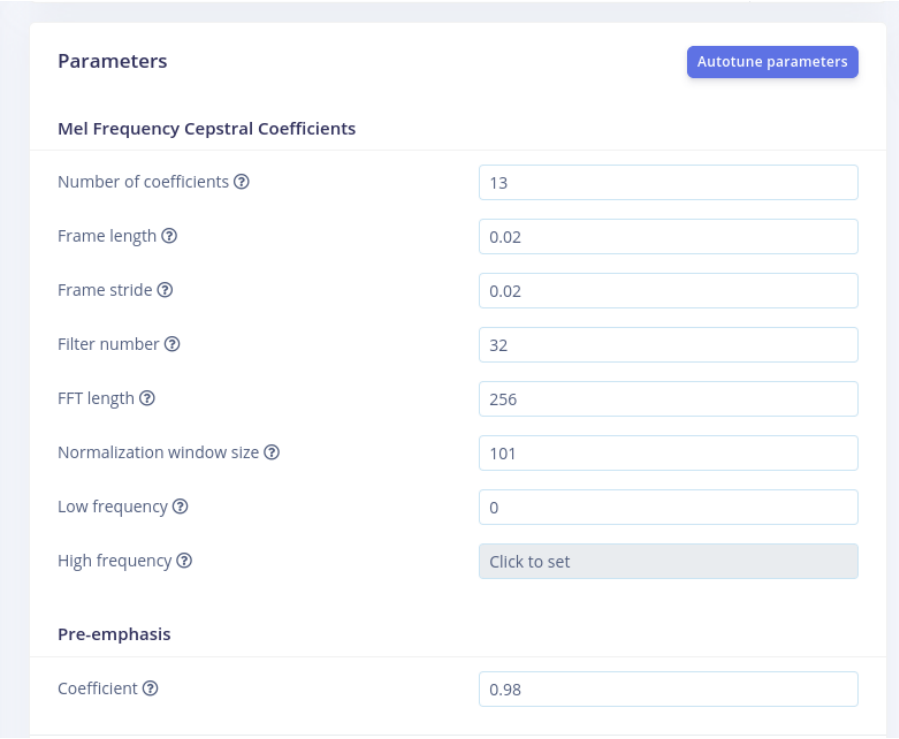

Figura 8: Parámetro MFCC

5. Una vez escogido el modelo se empieza el entrenamiento. Donde la plataforma automáticamente va introduciendo los datos recolectados y valorando al algoritmo. Al finalizar, presenta la exactitud del modelo implementado para tener idea de la veracidad, como se puede notar el modelo tiene una exactitud muy alta.

Figura 9: Eficiencia del modelo

6. Finalmente, la plataforma realiza un despliegue del algoritmo para el hardware seleccio-

nado, en este caso para Arduino la cuál da como salida una librería que se puede exportar para realizar la manipulación como sea necesaria.

2.8. Detector de palabras/oraciones

Se escogió las siguientes tres palabras/oraciones:

- abrir puerta (abrir-puerta)
- encender luz (encender-luz)
- reproducir (reproducir)

Si no es ninguna de las anteriores el modelo lo infiere como:

- Ruido (_noise)
- Desconocido (_unknown)

Además, se va a utilizar los LEDs RGB del Arduino Nano BLE para que se enciendan patrones de colores específicos dependiendo de la palabra detectada.

- Desconocido = Rojo
- abrir puerta = Verde
- encender luz = Azul
- reproducir = Azul+Verde (blanco)

Algunos parámetros importantes que se pueden modificar por el usuario es la cantidad de inferencias hechas cada segundo y eso depende de lo pesado que sea el modelo y la memoria del microcontrolador. En nuestro caso nos funciona tener cuatro inferencias en un segundo al tener un modelo de tamaño adecuado.

En el apendice se tiene el código .ino detector de palabras.

2.9. Thingsboard.py

En el apéndice vamos a encontrar el código cliente para Thingsboard utilizando MQTT parecido a lo usado en el laboratorio 4 con el microcontrolador STM32. La diferencia es en los datos recibidos son palabras con un grado de 0 a 1 de fiabilidad de que se detecto la palabra por el micrófono del Arduino Nano utilizando la inferencia del modelo obtenido por medio de tiny machine learning .

El algoritmo del código se tiene que si detecta que una palabra tiene una fiabilidad mayor a 0.7 se va a enviar la palabra al Thingsboard con la etiqueta de 'ON', todos inician con 'OFF', para que se muestre que se detecto la palabra y si se vuelve a detectar la palabra se va a poner en 'OFF'. En caso de detectar ruido o palabras desconocidas no se hace nada y en el Thingsboard se va a estar mostrando en vivo si se detecto ruido, palabra conocida o desconocida.

3. Desarrollo/Análisis de resultados

3.1. Análisis de funcionalidad electrónica

Para comprobar que el micrófono funciona se utiliza el script de .ino y .py y se obtiene un archivo .wav el cual utilizando una aplicación de reproducción de audio se puede escuchar la grabación, sin embargo, al ser pequeño el microfono no se escucha bien la grabación. En este enlace tenemos una muestra del funcionamiento de los LEDs RGB como se deseaba.

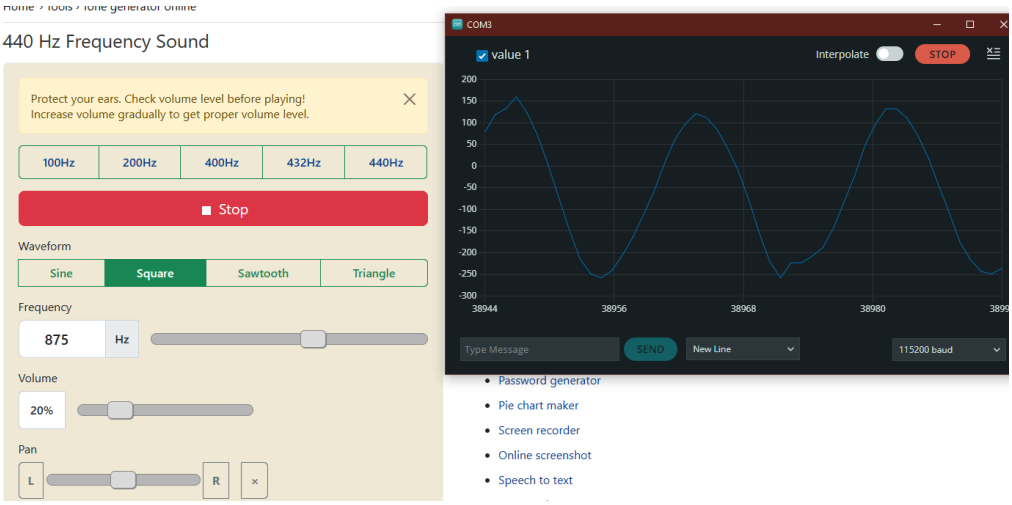

Figura 10: Prueba de micrófono con test de sonidos

3.2. Análisis de la funcionalidad del programa

Para la prueba de funcionalidad vamos a observar por Thingboard con los dashboards cuando se detecta las palabras y se activa o desactiva los cuadros. Esto se puede observar en las figuras 11, 12, 13, 14 y 15. Luego en las figuras 17, 18y 16 se observa que al detectar nuevamente las palabras se apagan sus widgets.

3.2.1. Primera detección

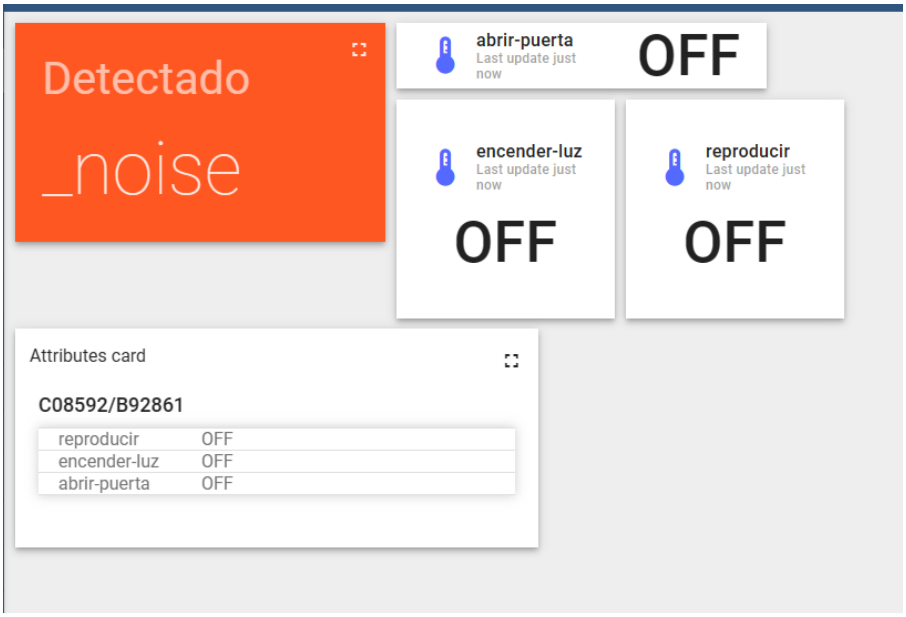

Figura 11: Detección de ruido

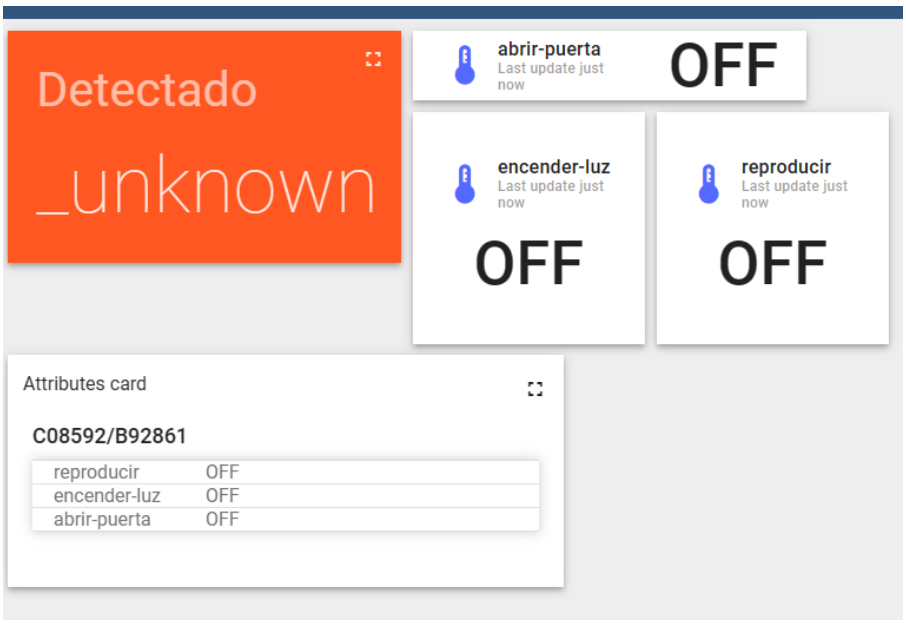

Figura 12: Detección de palabras desconocidas

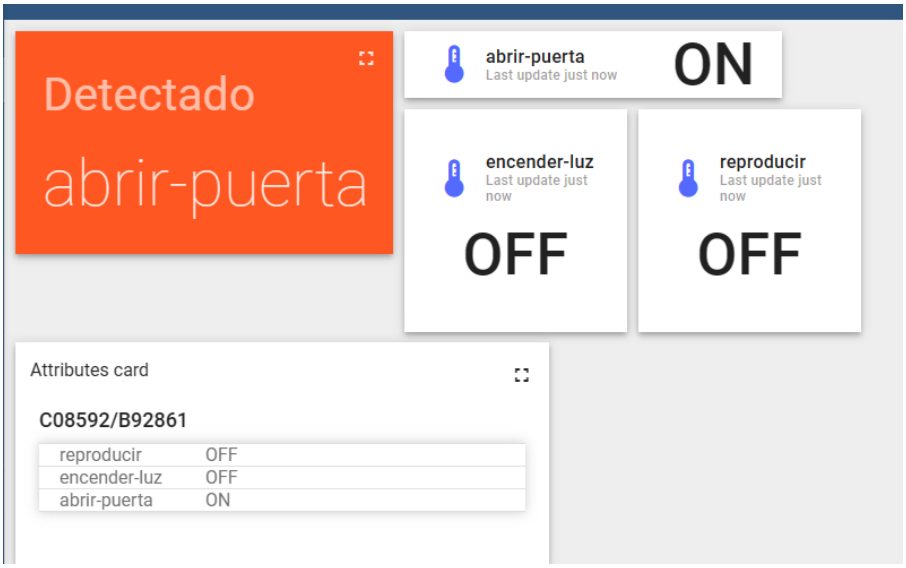

Figura 13: Detección de abrir puerta

3.2.2. Segunda detección

Figura 14: Detección de palabras reproducir

Figura 15: Detección de palabras encender luz

Figura 16: Detección de palabras segunda vez

Figura 17: Detección de abrir puerta segunda vez

Figura 18: Detección de palabras segunda vez

4. Conclusiones y recomendaciones

- Se aprendió sobre Tiny machine learning
- Se obtuvo un mayor conocimiento sobre funcionamiento de los micrófonos
- Se trabajo con ThingsBoard nuevamente y se aprovecho su dashboard mejor veces anteriores.
- Se logro utilizar las librerías necesarias para utilizar el micrófono del Arduino Nano.
- Se pudo crear un modelo con Tiny Machine Learning para inferir palabras por medio de micrófono.
- Hay que tener cuidado con manejar el microcontrolador al ser tan pequeño
- El cable para UART a veces da problemas entonces es recomendable tener varios para descartar que sea un problema del microcontrolador.

Referencias

[1] M. Rovai. (2022) Setting up your hardware and software - tinymledu. [Online]. Available: https://tinyml.seas.harvard.edu/assets/other/4D/22.03.11_Marcelo_Rovai_Handout.pdf

[2] S.-C. Wang and S.-C. Wang, "Artificial neural network," *Interdisciplinary computing in java programming*, pp. 81–100, 2003.

[3] E. Impulse. (2023) Build. train. optimize. ai for the edge. [Online]. Available: https://edgeimpulse.com/

[4] Arduino. (2023) Arduino tiny machine learning kit. [Online]. Available: https://store-usa.arduino.cc/products/arduino-tiny-machine-learning-kit?selectedStore=us

[5] ——. (2023) Nano 33 ble sense — arduino documentation. [Online]. Available: https://docs.arduino.cc/hardware/nano-33-ble-sense

[6] N. Semiconductor. (2023) Nrf52840 datasheet. [Online]. Available: https://pdf1.alldatasheet.com/datasheet-pdf/download/1425083/NORDIC/NRF52840.html

5. Apéndice

5.1. Precios de componentes

Cuadro 2: Precios de cada componente(Tienda)

Componente	Precio (Colones)	Cantidad
TML kit	45350	1

5.2. Cliente de python para ThingsBoard

```
import csv, serial
import time, json
import paho.mqtt.client as mqtt

#CLIENT METHODS
def on_connect(client, userdata, flags, rc):
    if not rc:
        client.is_connected = True
        print('Connection established. Working')
    else:
        print('Connection Faild', rc)
        client.loop_stop()

def on_disconnect(client, userdata, rc):
    if(rc == 0):
        print("Client disconneted OK")
    else:
        print("System disconnected via code:", rc)

def on_log(client, userdata, level, buf):
   print(buf)

def on_publish(client, userdata, mid):
    print("In on_pub callback mid=", mid)

# Obtain data
ser = serial.Serial(
    port = 'COM3',\
    baudrate = 115200,\
    timeout=1\
)

#SETUP
client = mqtt.Client('B92861')
client.on_connect    = on_connect
client.on_disconnect = on_disconnect
client.on_publish    = on_publish
client.on_log        = on_log
client.is_connected  = False
port         =   1883
broker       =   "iot.eie.ucr.ac.cr"
topic        =   "v1/devices/me/telemetry"
username     =   'C08592/B92861'
password     =   'rswm0cp3g4ko6ej328z4'
client.username_pw_set(password)
client.connect(broker, port)
while not client.is_connected:
    client.loop()
    time.sleep(0.5)

filename = open('data.csv','w')
```

```
output_file = csv.writer(filename)
data_saved = {'X':[], 'Y':[], 'Z':[], 'B':[], 'Alert':[]}
on_off = {1: "BATERY-LOW", 0: "---"}

print("Connection succeded")
en = "ON"
dis = "OFF"
mode = {1:en,0:dis}
palabras = ["abrir-puerta","encender-luz","reproducir"]
data_out = {"Detectado":"",
        palabras[0]: mode[0],
        palabras[1]: mode[0],
        palabras[2]: mode[0]
        }
while (1):
    data_receive = ser.readline().decode('utf-8').replace("\n","").
       replace("\r","").split(":")

    if data_receive[0][0]!='P':
        if float(data_receive[1])>0.7:
            detect = data_receive[0].replace("-","")
            data_out['Detectado'] = detect
            if detect in palabras:
                if data_out[detect]==mode[1]:
                    data_out[detect] = mode[0]
                else:
                    data_out[detect] = mode[1]

            print(data_out)
            #output_file.writerow(data)
            output = json.dumps(data_out)
            print("Topic:", topic, "output=", output)
            pub = client.publish(topic, output)
            client.loop()
```

5.3. Detector de palabras ArduinoIDE

```
/* Edge Impulse ingestion SDK
 * Copyright (c) 2022 EdgeImpulse Inc.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
```

```
 */

// If your target is limited in memory remove this macro to save 10K RAM
#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 4

/*
 ** NOTE: If you run into TFLite arena allocation issue.
 **
 ** This may be due to may dynamic memory fragmentation.
 ** Try defining "-DEI_CLASSIFIER_ALLOCATION_STATIC" in boards.local.txt
    (create
 ** if it doesn't exist) and copy this file to
 ** `<ARDUINO_CORE_INSTALL_PATH>/arduino/hardware/<mbed_core>/<
    core_version>/`.
 **
 ** See
 ** (https://support.arduino.cc/hc/en-us/articles/360012076960-Where-are-
    the-installed-cores-located-)
 ** to find where Arduino installs cores on your machine.
 **
 ** If the problem persists then there's not enough memory for this model
     and application.
 */

/* Includes
   ---------------------------------------------------------------- */
#include <PDM.h>
#include <speech_recognation2_inferencing.h>

/** 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()
{
    // LED RGB
    pinMode(LEDR, OUTPUT);
    pinMode(LEDG, OUTPUT);
    pinMode(LEDB, OUTPUT);
    // Inicializar apagados
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, HIGH);
    // put your setup code here, to run once:
    Serial.begin(115200);
    // comment out the below line to cancel the wait for USB connection (
       needed for native USB)
    while (!Serial);
    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: Could not allocate audio buffer (size %d), this
           could be due to the window length of your model\r\n",
           EI_CLASSIFIER_RAW_SAMPLE_COUNT);
        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 (++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      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[1] == 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
```