Arduino Schnarchstopper mit Künstlicher Intelligenz

Schnarchen ist ärgerlich, klar. Dagegen gibt es Geräte, Kissen, Apps und viele andere Mittelchen – aber die meisten davon dürften kaum halten, was sie versprechen. Falls du schnarchst und versprochen hast, dich darum zu kümmern, kannst du mit diesem Tutorial dein Leiden mit deinem Hobby verbinden: **Du baust einen Arduino Schnarchstopper mit künstlicher Intelligenz.**

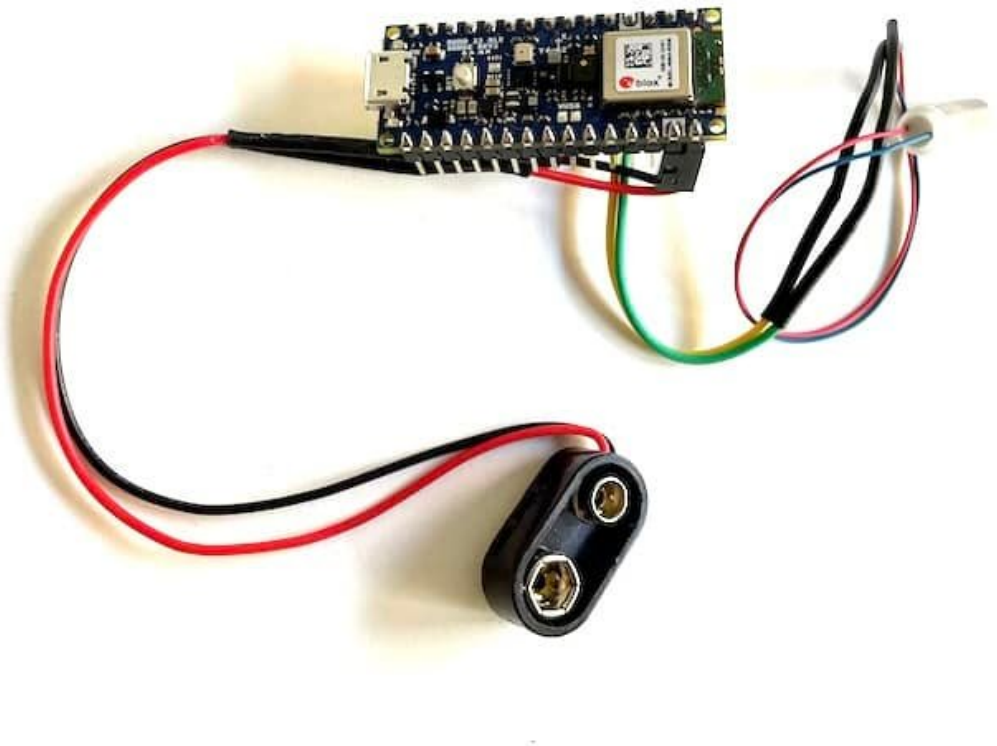
Zum Einsatz kommt ein Arduino Nano 33 BLE Sense und ein Vibrationsmotor*. Auf dem Microcontroller läuft ein KI-Modell, das über das Mikrofon des Arduinos erkennt, ob du schnarchst. Ist das der Fall, springt der Motor an und weckt dich (hoffentlich).

Aufbau des Schnarchstoppers

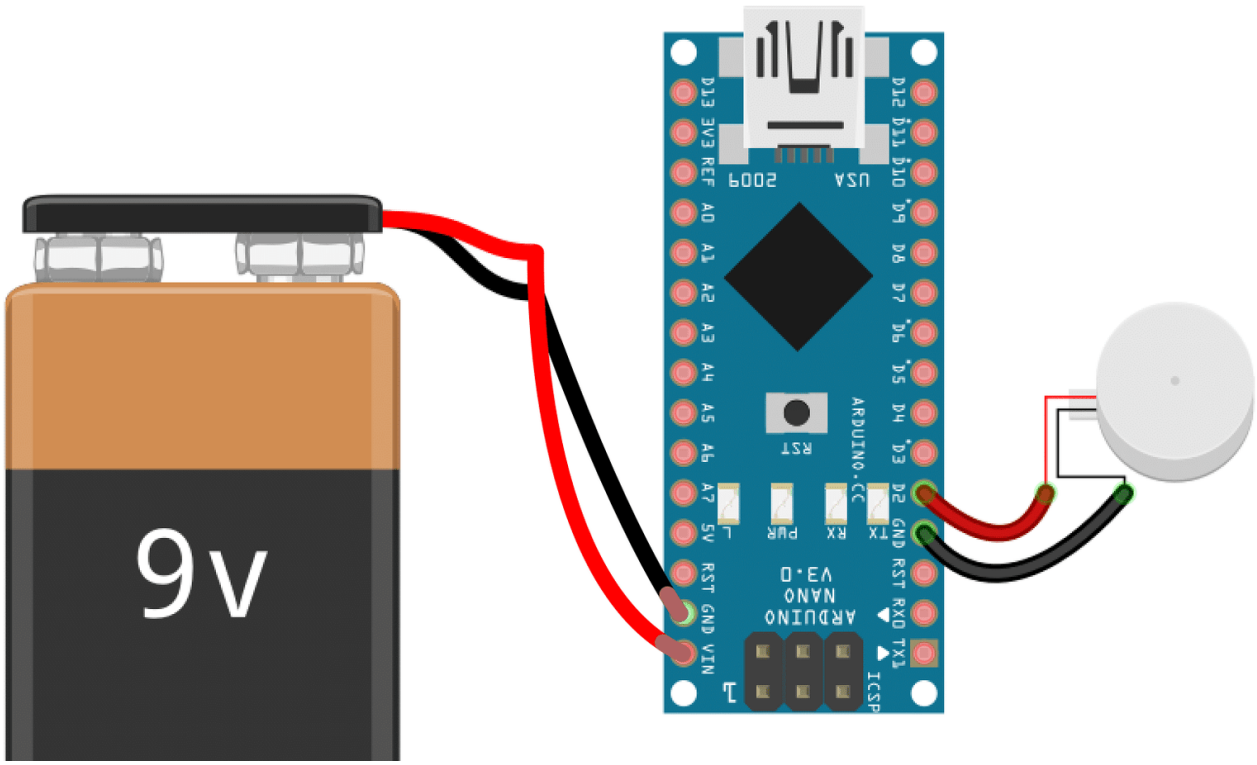
Neben dem Arduino Nano 33 BLE Sense benötigst du nur einen Vibrationsmotor und ein Verbindungskabel für eine 9V-Batterie. Der Motor sollte stark genug sein, um dich wecken zu können – wobei es natürlich auch darauf ankommt, wo du den

Schnarchstopper platzierst. Wenn du eine Armbinde verwendest, kann der Vibrationsmotor beispielsweise direkt auf deinem Oberarm oder auch oberhalb des Handgelenks direkt auf deiner Haut aufliegen. Hierfür eignet sich zum Beispiel das Band, das dem Spiel Ring Fit für die Nintendo Switch* beiliegt.

Statt Vibration kannst du natürlich auch zu anderen Mitteln greifen. Ein Piezo-Summer dürfte dich wohl mit Sicherheit aus dem Schlaf reißen – leider aber auch deine Bettgenossin oder deinen Bettgenossen. Im Folgenden bleibst du beim Vibrationsmotor. Der Aufbau sieht dann so aus:



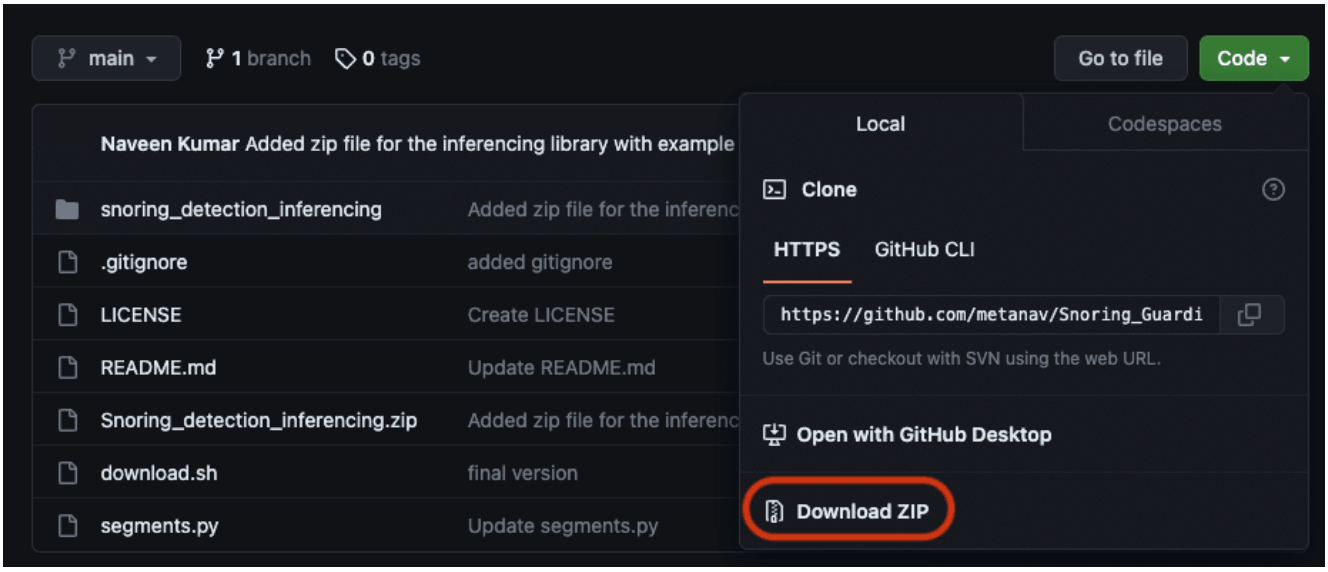
Falls du einen Arduino mit Header-Pins hast, kannst du die Kabel der beiden Bauteile entsprechend anpassen. Löte hierfür jeweils zwei Kabel mit einer Buchse an, die du dann am Arduino aufstecken kannst. Hier der Aufbau als Skizze:



Das passende KI-Modell und der Sketch

Damit der Schnarchstopper dein Schnarchen erkennt, benötigst du ein KI-Modell, das du in deinem Arduino Sketch verwendest.

Dieses Tutorial ist angelehnt an diesem Projekt auf GitHub – mit ein paar Anpassungen. Der Maker metanav hat dort schon viel Vorarbeit geleistet, die du weiterverwenden kannst. Lade dir auf GitHub oder direkt hier das Projekt als ZIP-Datei herunter:

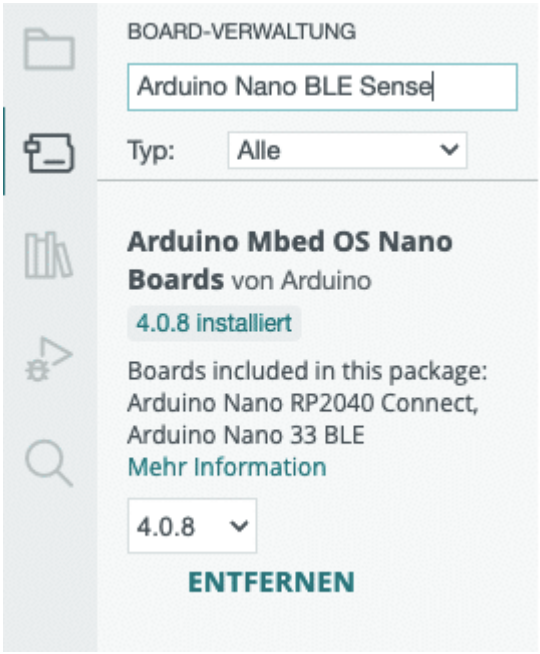


Entpacke anschließend die ZIP-Datei und öffne den Sketch **tflite_micro_snoring_detection.ino**, den du im Ordner **Snoring-Guardian-main > snoring_detection_inferencing > examples >tflite_micro_snoring_detection** findest.

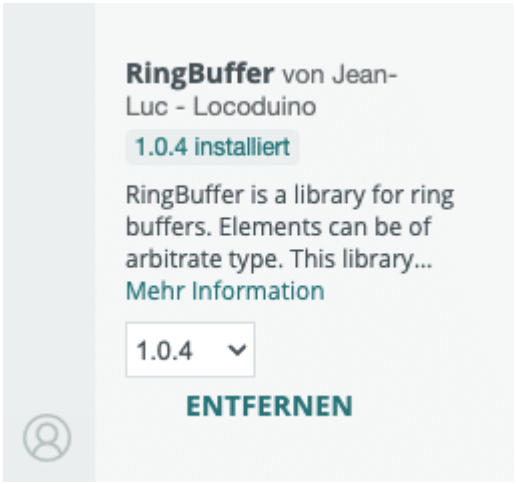
Anschließend bindest du die mitgelieferte Bibliothek ein. Darin enthalten ist das KI-Modell, das du verwenden wirst. Öffne in der Arduino IDE den Menüpunkt **Sketch > Bibliothek einbinden > ZIP-Bibliothek hinzufügen** und wähle die Datei **Snoring_detection_inferencing.zip**

Den Arduino NANO und eine weitere Bibliothek installieren

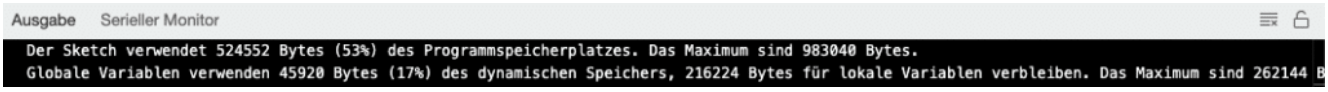
Falls du du den Arduino Nano 33 BLE Sense in der Arduino IDE noch nicht verfügbar gemacht hast, hole das schnell nach. Öffne hierfür den Boardverwalter im Menü links, suche nach **Arduino Nano BLE Sense** und installiere die aktuelle Version von **Arduino Mbed OS Nano Boards**.



Außerdem benötigst du noch die Bibliothek **RingBuf**, die du über den Bibliotheksverwalter installieren kannst. **Aber Achtung:** Die gleichnamige Bibliothek funktioniert nicht auf dem Arduino Nano. Installiere stattdessen die Bibliothek **RingBuffer** von Jean-Luc – Locoduino:



Für einen Test wähle in der Arduino IDE dein Board aus und klicke oben links auf den Haken für die Überprüfung des Sketchs. Die Kompilierung nimmt einige Zeit in Anspruch, aber wenn die richtigen Bibliotheken installiert bzw. eingebunden wurden und die Verbindung zum Arduino Nano steht, sollte sie erfolgreich abgeschlossen werden:



Anpassungen im Sketch

Du kannst den Sketch für den Schnarchstopper direkt auf deinen Arduino hochladen und verwenden, aber je nachdem, welchen Vibrationsmotor (oder welches andere Bauteil) du verwendest, musst du ein paar Kleinigkeiten anpassen.

Im Sketch kümmert sich die Funktion **void run_vibration()** um den Start des Motors. Im Beispiel-Sketch sieht sie wie folgt aus:

```
void run_vibration()
{
  if (alert)
  {
    is_motor_running = true;

    for (int i = 0; i < 2; i++)
    {
      analogWrite(vibratorPin, 30);
      delay(1000);
      analogWrite(vibratorPin, 0);
      delay(1500);
    }
    is_motor_running = false;
  } else {
    if (is_motor_running)
    {
      analogWrite(vibratorPin, 0);
    }
  }
  yield();
}
```

Hier wird der Motor 3 Mal jeweils für eine Sekunde gestartet,

mit einer Pause von 1,5 Sekunden dazwischen. Hierfür wird **analogWrite()** mit einem Wert von 30 verwendet. Der Vibrationsmotor, den ich verwende, versteht allerdings nur Ein und Aus. Falls das bei dir auch der Fall ist, ändere die betreffende Stelle folgendermaßen:

```
for (int i = 0; i < 2; i++) {
  digitalWrite(vibratorPin, HIGH);
  delay(5000);
  digitalWrite(vibratorPin, LOW);
  delay(1000);
}
```

Hier verwendest du **digitalWrite()** und sendest damit entweder ein HIGH oder LOW an den Motor. Ebenfalls sind dort die Lauf- und Pausenzeiten geändert – die fünf Sekunden zielen hier eher auf Schnarcher mit einem tiefen Schlaf.

Und noch eine Anpassung: Wenn du deinen Vibrationsmotor wie in der Skizze oben an den Pin D2 angecshlossen hast, ändere noch die entsprechende Zeile im Sketch:

```
int vibratorPin = 2;
```

Lade nun den Sketch auf deinen Arduino Nano hoch – du findest ihn ganz am Ende dieses Tutorials.

Den Schnarchstopper testen

Jetzt ist es so weit – sobald der Sketch erfolgreich auf deinem Arduino gelandet ist, öffne den Seriellen Monitor in der IDE. Dort siehst du die Vorhersagen, die das KI-Modell macht auf Basis der Geräusche, die es über das eingebaute Mikrofon des Arduinos erhält:

Im oben rot markierten Fall war das ein normales Hintergrundgeräusch (noise) mit einer Wahrscheinlichkeit von 99,219 %. Das hier jemand geschnarcht hat, war hingegen nur zu 0,781 % wahrscheinlich.

Es ist vermutlich etwas peinlich, aber imitiere nun mehrmals hintereinander typische Schnarchlaute. Du wirst sehen, dass die interne LED des Arduinos aufleuchtet und sich die Ausgabe im Seriellen Monitor entsprechend verändert. Sobald mehrmals ein Schnarchen erkannt wurde, springt auch der Vibrationsmotor an und vibriert im von dir in der Funktion **run_vibration()** definierten Rhythmus.

Als nächstes wird es Zeit für ein paar „echte“ Tests in der Nacht. Da du deinen Arduino Nano auch mit einer 9V-Batterie versorgen kannst, steht deinen Versuchen im Bett nichts im Wege. Vermutlich musst du mehrere Möglichkeiten der Positionierung des Motors ausprobieren, **um von seinen Vibrationen aufzuwachen bzw. um dem Mikrofon des Arduinos zu ermöglichen, dich einwandfrei beim Schnarchen aufzunehmen.** Sollte letzteres nicht der Fall sein, kann es zu Fehlalarmen kommen.

Und natürlich gibt es keine Garantie, dass dein neuer Schnarchstopper überhaupt zu ruhigen Nächten führt…

Entwickle dein eigenes KI-Modell

Da du bisher ein vorgefertigtes Modell verwendet hast, ist es möglich, dass es für dich nicht bestmöglich funktioniert. Jeder schnarcht schließlich anders – und die Schnarchgeräusche, die für das Training des Modells verwendet wurden, können stark von deinen eigenen abweichen.

Falls du also einen Schritt weitergehen möchtest, ist das kein Problem – nur etwas Arbeit. Auf Pollux Labs findest du Tutorials, wie du den Service **Edge Impulse** verwenden kannst, um ein persönliches KI-Modell zu entwickeln. Dort erfährst du, wie du deinen Arduino Nano 33 BLE Sense mit Edge Impulse verbindest, damit Daten sammelst und dein eigenes Modell trainierst. Im letztgenannten Tutorial geht es zwar um Bewegungsdaten, aber ähnlich funktioniert auch das Training mit Audio.

Apropos Audio, um ausreichend Schnarchtöne zu sammeln, eignet sich bereits dein Smartphone. Starte einfach eine Tonaufnahme und lass das Smartphone die Nacht über neben dir liegen. Die entsprechenden Passagen in der Audiodatei kannst du dann in Edge Impulse weiterverarbeiten.

Und nun viel Spaß und Erfolg beim Experimentieren mit deinem Schnarchstopper!

Der vollständige Sketch

Hier nun der gesamte Sketch mit den genannten Anpassungen:

```
// 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 3

/* Includes ---------------------------------------------------
-------------- */
#include <PDM.h>
#include <Scheduler.h>
#include <RingBuf.h>
#include <snore_detection_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);

bool alert = false;

RingBuf<uint8_t, 10> last_ten_predictions;
int greenLED = 23;
int vibratorPin = D2;   // Vibration motor connected to D2 PWM
pin
bool is_motor_running = false;

void run_vibration() {
  if (alert) {
    is_motor_running = true;
```

```
    for (int i = 0; i < 2; i++) {
      digitalWrite(vibratorPin, HIGH);
      delay(5000);
      digitalWrite(vibratorPin, LOW);
      delay(1000);
    }

    is_motor_running = false;
  } else {
    if (is_motor_running) {
      analogWrite(vibratorPin, LOW);
    }
  }
  yield();
}



/**
   @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);
}


void setup()
{
  Serial.begin(115200);

  pinMode(greenLED, OUTPUT);
  pinMode(greenLED, LOW);
  pinMode(vibratorPin, OUTPUT);  // sets the pin as output

  // 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;
  }

  Scheduler.startLoop(run_vibration);
}

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  (ix  ==  1  &&  !is_motor_running  &&
result.classification[ix].value > 0.9) {
        if (last_ten_predictions.isFull()) {
          uint8_t k;
          last_ten_predictions.pop(k);
        }

        last_ten_predictions.push(ix);
```

```
        uint8_t count = 0;

         for (uint8_t j = 0; j < last_ten_predictions.size();
j++) {
          count += last_ten_predictions[j];
          //ei_printf("%d, ", last_ten_predictions[j]);
        }
        //ei_printf("\n");
        ei_printf("Snoring\n");
        pinMode(greenLED, HIGH);
        if (count >= 5) {
          ei_printf("Trigger vibration motor\n");
          alert = true;
        }
      }  else {
        ei_printf("Noise\n");
        pinMode(greenLED, LOW);
        alert = false;
      }

      print_results = 0;
    }
  }
}


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