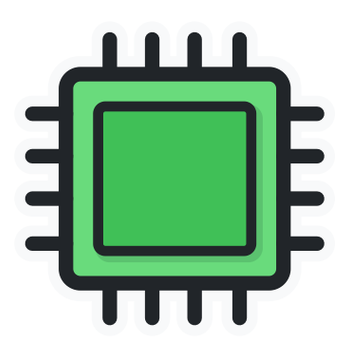


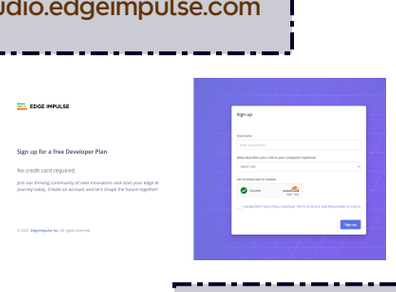
Embedded A.I - 4_TinyML_AudioClassification

TinyML

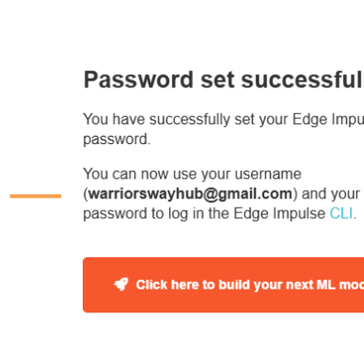
- TinyML refers to deploying machine learning models on tiny devices (like MCUs or edge devices) with:
  - Limited CPU (e.g., 240MHz)
  - Limited memory (e.g., 512KB–4MB RAM)
  - Low power requirements
  - No internet connection
  - It uses lightweight models optimized to run locally on embedded hardware.
- ML Workflow
  - #1 — Problem
  - #2 — Data Collection
  - #3 — Data Cleaning & Labeling
  - #4 — Feature Extraction
  - #5 — Model Training & Evaluation
  - #6 — Deployment
  - #7 — Inference

Create Edge Impulse Account

- visit — https://studio.edgeimpulse.com
- Create Account



- Sign up & create a new project — Project Name: Keyword Spotter AI / Type: Audio classification
- To use the CLI you will need to set a password. Go to your user profile settings in Studio to set one



Collect Audio Dataset

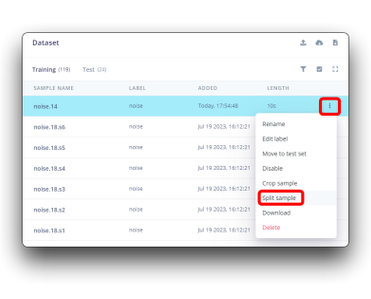
- ✅ Method 1: OFFLINE METHOD using SDCard
  - #1
    - Connect SDCard with Laptop using Card Reader
    - Format SD Card with FAT32 — and — Plug in SD Card in the XIAO
  - #2 — Save recorded sound samples as .wav audio files to a microSD card
    - Open Arduino IDE
    - Tools -> "PSRAMLOPI PSRAM" -> OPI PSRAM
  - #3 — Compile & Upload the Code
    - If your ESP32 version is 2.0.x

```
/*
 * WAV Recorder for Seeed XIAO ESP32S3 Sense
 *
 * NOTE: To execute this code, we will need to use the PSRAM
 * function of the ESP-32 chip, so please turn it on before uploading.
 * Tools>PSRAM: "OPI PSRAM"
 *
 * Adapted by M.Rovai @May23 from original Seeed code
*/

#include <I2S.h>
#include "FS.h"
#include "SD.h"
#include "SPI.h"

// make changes as needed
#define RECORD_TIME   10  // seconds, The maximum value is 240
#define WAV_FILE_NAME "data"

// do not change for best
#define SAMPLE_RATE 16000U
#define SAMPLE_BITS 16
#define WAV_HEADER_SIZE 44
#define VOLUME_GAIN 2

int fileNumber = 1;
String baseFileName;
bool isRecording = false;

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

  I2S.setAllPins(-1, 42, 41, -1, -1);
  if (!I2S.begin(PDM_MONO_MODE, SAMPLE_RATE, SAMPLE_BITS)) {
    Serial.println("Failed to initialize I2S!");
    while (1) ;
  }
  if(!SD.begin(21)){
    Serial.println("Failed to mount SD Card!");
    while (1) ;
  }
  Serial.printf("Enter with the label name\n");
  //record_wav();
}

void loop() {
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');
    command.trim();
    if (command == "rec") {
      isRecording = true;
    } else {
      baseFileName = command;
      fileNumber = 1; // reset file number each time a new base file name is set
      Serial.printf("Send rec for starting recording label \n");
    }
  }
  if (isRecording && baseFileName != "") {
    String fileName = "/" + baseFileName + "." + String(fileNumber) + ".wav";
    fileNumber++;
    record_wav(fileName);
    delay(1000); // delay to avoid recording multiple files at once
    isRecording = false;
  }
}

void record_wav(String fileName)
{
  uint32_t sample_size = 0;
  uint32_t record_size = (SAMPLE_RATE * SAMPLE_BITS / 8) * RECORD_TIME;
  uint8_t *rec_buffer = NULL;
  Serial.printf("Start recording ...\n");

  File file = SD.open(fileName.c_str(), FILE_WRITE);
  // Write the header to the WAV file
  uint8_t wav_header[WAV_HEADER_SIZE];
  generate_wav_header(wav_header, record_size, SAMPLE_RATE);
  file.write(wav_header, WAV_HEADER_SIZE);

  // PSRAM malloc for recording
  rec_buffer = (uint8_t *)ps_malloc(record_size);
  if (rec_buffer == NULL) {
    Serial.printf("malloc failed!\n");
    while(1) ;
  }
  Serial.printf("Buffer: %d bytes\n", ESP.getPsramSize() - ESP.getFreePsram());

  // Start recording
  esp_i2s::i2s_read(esp_i2s::I2S_NUM_0, rec_buffer, record_size, &sample_size, portMAX_DELAY);
  if (sample_size == 0) {
    Serial.printf("Record Failed!\n");
  } else {
    Serial.printf("Record %d bytes\n", sample_size);
  }

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

  // Write data to the WAV file
  Serial.printf("Writing to the file ...\n");
  if (file.write(rec_buffer, record_size) != record_size)
    Serial.printf("Write file Failed!\n");

  free(rec_buffer);
  file.close();
  Serial.printf("Recording complete: \n");
  Serial.printf("Send rec for a new sample or enter a new label\n\n");
}

void generate_wav_header(uint8_t *wav_header, uint32_t wav_size, uint32_t sample_rate)
{
  // See this for reference: http://soundfile.sapp.org/doc/WaveFormat/
  uint32_t file_size = wav_size + WAV_HEADER_SIZE - 8;
  uint32_t byte_rate = SAMPLE_RATE * SAMPLE_BITS / 8;
  const uint8_t set_wav_header[] = {
    'R', 'I', 'F', 'F', // ChunkID
    file_size, file_size >> 8, file_size >> 16, file_size >> 24, // ChunkSize
    'W', 'A', 'V', 'E', // Format
    'f', 'm', 't', ' ', // Subchunk1ID
    0x10, 0x00, 0x00, 0x00, // Subchunk1Size (16 for PCM)
    0x01, 0x00, // AudioFormat (1 for PCM)
    0x01, 0x00, // NumChannels (1 channel)
    sample_rate, sample_rate >> 8, sample_rate >> 16, sample_rate >> 24, // SampleRate
    byte_rate, byte_rate >> 8, byte_rate >> 16, byte_rate >> 24, // ByteRate
    0x02, 0x00, // BlockAlign
    0x10, 0x00, // BitsPerSample (16 bits)
    'd', 'a', 't', 'a', // Subchunk2ID
    wav_size, wav_size >> 8, wav_size >> 16, wav_size >> 24, // Subchunk2Size
  };
  memcpy(wav_header, set_wav_header, sizeof(set_wav_header));
}
```

    - If your ESP32 version is 3.0.x

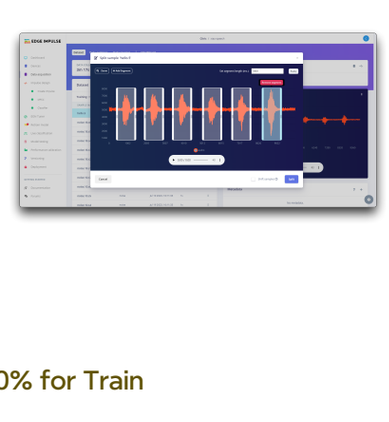
```
/*
 * WAV Recorder for Seeed XIAO ESP32S3 Sense
 *
 * NOTE: To execute this code, we will need to use the PSRAM
 * function of the ESP-32 chip, so please turn it on before uploading.
 * Tools>PSRAM: "OPI PSRAM"
 *
 * Adapted by M.Rovai @May23 from original Seeed code
*/

#include <ESP_I2S.h>
#include "FS.h"
#include "SD.h"
#include "SPI.h"

// make changes as needed
#define RECORD_TIME   10  // seconds, The maximum value is 240
#define WAV_FILE_NAME "data"

// do not change for best
#define SAMPLE_RATE 16000U
#define SAMPLE_BITS 16
#define WAV_HEADER_SIZE 44
#define VOLUME_GAIN 2

I2SClass I2S;
String baseFileName;

int fileNumber = 1;
bool isRecording = false;

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

  // setup 42 PDM clock and 41 PDM data pins
  I2S.setPinsPdmRx(42, 41);
  if (!I2S.begin(I2S_MODE_PDM_RX, 16000, I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_MODE_MONO)) {
    Serial.println("Failed to initialize I2S!");
    while (1) ;
  }
  if(!SD.begin(21)){
    Serial.println("Failed to mount SD Card!");
    while (1) ;
  }
  Serial.printf("Enter with the label name\n");
  //record_wav();
}

void loop() {
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');
    command.trim();
    if (command == "rec") {
      isRecording = true;
    } else {
      baseFileName = command;
      fileNumber = 1; // reset file number each time a new base file name is set
      Serial.printf("Send rec for starting recording label \n");
    }
  }
  if (isRecording && baseFileName != "") {
    String fileName = "/" + baseFileName + "." + String(fileNumber) + ".wav";
    fileNumber++;
    record_wav(fileName);
    delay(1000); // delay to avoid recording multiple files at once
    isRecording = false;
  }
}

void record_wav(String fileName)
{
  uint32_t sample_size = 0;
  uint32_t record_size = (SAMPLE_RATE * SAMPLE_BITS / 8) * RECORD_TIME;
  uint8_t *rec_buffer = NULL;
  Serial.printf("Start recording ...\n");

  File file = SD.open(fileName.c_str(), FILE_WRITE);
  // Write the header to the WAV file
  uint8_t wav_header[WAV_HEADER_SIZE];
  generate_wav_header(wav_header, record_size, SAMPLE_RATE);
  file.write(wav_header, WAV_HEADER_SIZE);

  // PSRAM malloc for recording
  rec_buffer = (uint8_t *)ps_malloc(record_size);
  if (rec_buffer == NULL) {
    Serial.printf("malloc failed!\n");
    while(1) ;
  }
  Serial.printf("Buffer: %d bytes\n", ESP.getPsramSize() - ESP.getFreePsram());

  // Start recording
  //esp_i2s::i2s_read(esp_i2s::I2S_NUM_0, rec_buffer, record_size, &sample_size, portMAX_DELAY);
  uint32_t bytes_read = 0;
while (bytes_read < record_size) {
  int bytes = I2S.read();
  if (bytes >= 0) {
    rec_buffer[bytes_read++] = bytes & 0xFF;
    rec_buffer[bytes_read++] = (bytes >> 8) & 0xFF;
  }
}
sample_size = bytes_read;

  if (sample_size == 0) {
    Serial.printf("Record Failed!\n");
  } else {
    Serial.printf("Record %d bytes\n", sample_size);
  }

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

  // Write data to the WAV file
  Serial.printf("Writing to the file ...\n");
  if (file.write(rec_buffer, record_size) != record_size)
    Serial.printf("Write file Failed!\n");

  free(rec_buffer);
  file.close();
  Serial.printf("Recording complete: \n");
  Serial.printf("Send rec for a new sample or enter a new label\n\n");
}

void generate_wav_header(uint8_t *wav_header, uint32_t wav_size, uint32_t sample_rate)
{
  // See this for reference: http://soundfile.sapp.org/doc/WaveFormat/
  uint32_t file_size = wav_size + WAV_HEADER_SIZE - 8;
  uint32_t byte_rate = SAMPLE_RATE * SAMPLE_BITS / 8;
  const uint8_t set_wav_header[] = {
    'R', 'I', 'F', 'F', // ChunkID
    file_size, file_size >> 8, file_size >> 16, file_size >> 24, // ChunkSize
    'W', 'A', 'V', 'E', // Format
    'f', 'm', 't', ' ', // Subchunk1ID
    0x10, 0x00, 0x00, 0x00, // Subchunk1Size (16 for PCM)
    0x01, 0x00, // AudioFormat (1 for PCM)
    0x01, 0x00, // NumChannels (1 channel)
    sample_rate, sample_rate >> 8, sample_rate >> 16, sample_rate >> 24, // SampleRate
    byte_rate, byte_rate >> 8, byte_rate >> 16, byte_rate >> 24, // ByteRate
    0x02, 0x00, // BlockAlign
    0x10, 0x00, // BitsPerSample (16 bits)
    'd', 'a', 't', 'a', // Subchunk2ID
    wav_size, wav_size >> 8, wav_size >> 16, wav_size >> 24, // Subchunk2Size
  };
  memcpy(wav_header, set_wav_header, sizeof(set_wav_header));
}
```

  - In Arduino Serial monitor
    - #4
      - Send the label — noise
      - Send the Command "Rec" to start recording — rec
      - do above step againa and again to collect more audio for "activate" voice
    - #5
      - Send the label — activate
      - Send the Command "Rec" to start recording — rec
      - do above step againa and again to collect more audio for "activate" voice
    - #6
      - Send new label — warning
      - Send the Command "Rec" to start recording for new label — rec
      - do above step againa and again to collect more audio for "warning" voice
    - #7
      - Send new label — stand down
      - Send the Command "Rec" to start recording for new label — rec
      - do above step againa and again to collect more audio for "stand down" voice
  - #10 — Uploading collected sound data
    - When the raw dataset is defined and collected, we should initiate a new project at Edge Impulse.
    - Once the project is created, select the Upload Existing Data tool in the Data Acquisition section. — Choose the files to be uploaded
    - And upload them to the Studio (You can automatically split data in train/test). Repete to all classes and all raw data.
    - All data on dataset have a 1s length, but the samples recorded in the previous section have 10s and must be split into 1s samples to be compatible.

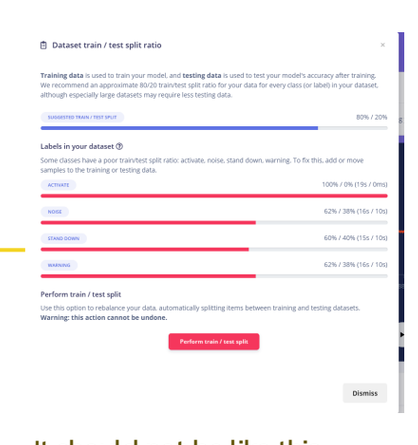


Click on three dots after the sample name and select Split sample.

    - Once inside the tool, split the data into 1-second records. If necessary, add or remove segments.



    - This procedure should be repeated for all samples.
    - Observe Train / Test Split
      - 100% Data
        - 80% for Train
        - 20% for Test
      - 100% Data
        - 75% for Train
        - 25% for Test
      - Even Splitting — 4 Classes
        - Noise
        - Activate
        - Warning
        - Stand down
        - Need Equal Balanced



It should not be like this

      - click Perform train test split
  - Note — Keep samples 1 second in length
- ❌ Method 2: ONLINE METHOD using CLI daemon (53)

Create Impulse (Pre-Process / Model definition) (14)

Pre-Processing (MFCC) (9)

Model Design and Training (6)

Test Model in Studio

Export for Deployment

(1)