CSCE 5612 Embedded Hardware/Software Design
Spring 2025

Lab #4: Activity Recognition with ML and IMU

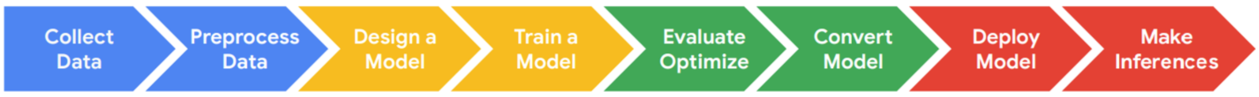
Similar to the previous project, you will need the Seeed Xiao nRF52 board, a USB-C cable, the Arduino IDE, and a LiPo battery.

Software and Libraries Needed

- Install the built-in accelerator library Seeed_Arduino_LSM6DS3;
- Install the Arduino TensorFlow Lite Library.
- Python – Have the latest version installed. Also, you need to install the 'bleak' library. https://pypi.org/project/bleak/0.12.0/

2. Learning Outcomes

- Learning about how to collect wearable sensor data in a controlled environment
- Learning how to label the data
- Learning about the evaluation of an embedded device



3. Phase I (Gesture Recognition)

- Identify the MAC address of your nRF board. Open a new sketch in Arduino IDE and run this code to find your MAC address.

```
#include <Arduino.h>
#include <ArduinoBLE.h>

void setup() {
  Serial.begin(115200);
  delay(2000);  // give USB time
  Serial.println("BOOT");

  // Start BLE
  if (!BLE.begin()) {
    Serial.println("starting BLE failed!");
    while (1) { delay(10); }
  }

  // Print LOCAL BLE address (MAC-like)
  Serial.print("BLE Address: ");
  Serial.println(BLE.address());

  // (Optional) advertise so you can also see it in scanning apps
  BLE.setLocalName("Data Collection");
  BLE.advertise();

  Serial.println("Advertising...");
}

void loop() {
  // nothing
}
```

1. Training IMU data from

We will obtain some sensor data to use for training the model later. First, we will run a simple Arduino program to stream sensor data from the Seeed Xiao board.

Copy the following code and upload the code to your board:

```
#include <Arduino.h>
#include <Wire.h>
#include <math.h>
#include <LSM6DS3.h>
#include <ArduinoBLE.h>

// ---------------- IMU ----------------
LSM6DS3 myIMU(I2C_MODE, 0x6A);  // keep exactly as your working code
float aX, aY, aZ;

// Motion trigger + sampling
const float    accelerationThreshold = 1.0f;   // sum(|ax|+|ay|+|az|) in g
const int      numSamples = 119;
const uint32_t samplePeriodMs = 10;            // 100 Hz
```

```
int samplesRead = numSamples;
uint32_t eventCount = 0;

// ---------------- BLE ----------------
// Use your UUIDs (same as ESP32 version)
static const char* SERVICE_UUID = "12345678-1234-5678-1234-56789abcdef0";
static const char* CHAR_UUID    = "12345678-1234-5678-1234-56789abcdef1";

BLEService imuService(SERVICE_UUID);

// Notify-only characteristic (add BLERead too if you want debugging)
BLECharacteristic imuChar(CHAR_UUID, BLENotify, 8);
// 8 bytes payload: ctr(2) + ax(2) + ay(2) + az(2)

// Pack signed 16-bit little-endian
static inline void pack_i16(uint8_t* buf, int idx, int16_t v) {
  buf[idx]     = (uint8_t)(v & 0xFF);
  buf[idx + 1] = (uint8_t)((v >> 8) & 0xFF);
}

static uint16_t ctr = 0;

void setup() {
  Serial.begin(115200);
  delay(2000);
  Serial.println("BOOT");

  // I2C
  Wire.begin();
  Wire.setClock(400000);

  // IMU init
  int imuStatus = myIMU.begin();
  if (imuStatus != 0) {
    Serial.print("IMU initialization failed, status = ");
    Serial.println(imuStatus);
  } else {
    Serial.println("IMU initialized");
  }

  // BLE init (ArduinoBLE)
  if (!BLE.begin()) {
    Serial.println("starting BLE failed!");
    while (1) { delay(10); }
  }

  BLE.setLocalName("XIAO_IMU_EVT");
  BLE.setDeviceName("XIAO_IMU_EVT");
  // Print local BLE address
  Serial.print("BLE Address: ");
  Serial.println(BLE.address());

  // Setup service + characteristic
  BLE.setAdvertisedService(imuService);
  imuService.addCharacteristic(imuChar);
  BLE.addService(imuService);

  // Start advertising
  BLE.advertise();

  Serial.println("Advertising BLE...");
  Serial.println("Ready. Waiting for motion...");
  Serial.println("Will notify ONLY during motion event (119 samples @100Hz).");
}

void loop() {
  // Required for ArduinoBLE background processing
  BLE.poll();

  // Detect connection state (optional prints)
  BLEDevice central = BLE.central();
  if (central && central.connected()) {
    static bool printed = false;
    if (!printed) {
      Serial.print("Connected to central: ");
      Serial.println(central.address());
      printed = true;
    }
  } else {
    static bool printedDisc = false;
    static String last = "";
    // Reset printed flag when not connected
    // (simple, avoids spam)
  }

  // -------- Wait for motion --------
  while (samplesRead == numSamples) {
    BLE.poll();

    aX = myIMU.readFloatAccelX();
    aY = myIMU.readFloatAccelY();
    aZ = myIMU.readFloatAccelZ();

    float aSum = fabs(aX) + fabs(aY) + fabs(aZ);
    if (aSum >= accelerationThreshold) {
      samplesRead = 0;
      eventCount++;

      Serial.print("\n=== Motion event #");
      Serial.print(eventCount);
      Serial.println(" detected ===");
      break;
    }

    delay(5);
  }

  // -------- Collect + notify samples --------
  while (samplesRead < numSamples) {
    BLE.poll();

    aX = myIMU.readFloatAccelX();
    aY = myIMU.readFloatAccelY();
    aZ = myIMU.readFloatAccelZ();

    // scale into int16 for BLE packet
    int16_t ax_mg = (int16_t)(aX * 1000.0f);
    int16_t ay_mg = (int16_t)(aY * 1000.0f);
    int16_t az_mg = (int16_t)(aZ * 1000.0f);

    uint8_t pkt[8];
    pack_i16(pkt, 0, (int16_t)ctr++);
    pack_i16(pkt, 2, ax_mg);
    pack_i16(pkt, 4, ay_mg);
    pack_i16(pkt, 6, az_mg);

    // Notify ONLY if a central is connected + subscribed
    // (writeValue works even if not subscribed, but no one receives it)
    imuChar.writeValue(pkt, sizeof(pkt));

    samplesRead++;
    delay(samplePeriodMs);

    if (samplesRead == numSamples) {
      Serial.println("Finished logging a motion event.\n");
    }
  }
}
```

Note: It’s important that you change the name of the device on this line:

```
BLE.setLocalName("XIAO_IMU_EVT");
BLE.setDeviceName("XIAO_IMU_EVT");
```

2. Capturing Gestures Training Data

Py script for windows:

```
import asyncio
from datetime import datetime
from bleak import BleakClient

ADDRESS = "ed:b1:c7:20:c8:7c"  # <-- Replace your nRF BLE Address
CHAR_UUID = "12345678-1234-5678-1234-56789abcdef1"
CSV_PATH = "accel_log.csv"

def int16_le(b0, b1):
   return int.from_bytes(bytes([b0, b1]), byteorder="little", signed=True)

def handle_notify(sender, data: bytearray):
   # Expect exactly 8 bytes: ctr + ax + ay + az (all int16 little-endian)
   if len(data) != 8:
      try:
         txt = data.decode("utf-8", errors="ignore").strip()
         print("Non-8B packet:", len(data), "as text:", txt)
      except Exception:
         print("Non-8B packet:", len(data), "raw:", list(data))
      return

   ctr = int16_le(data[0], data[1])

   ax_mg = int16_le(data[2], data[3])
   ay_mg = int16_le(data[4], data[5])
   az_mg = int16_le(data[6], data[7])

   ax_g = ax_mg / 1000.0
   ay_g = ay_mg / 1000.0
   az_g = az_mg / 1000.0

   ts = datetime.now().isoformat(timespec="milliseconds")
   line = f"{ts},{ctr},{ax_g:.3f},{ay_g:.3f},{az_g:.3f}"
```

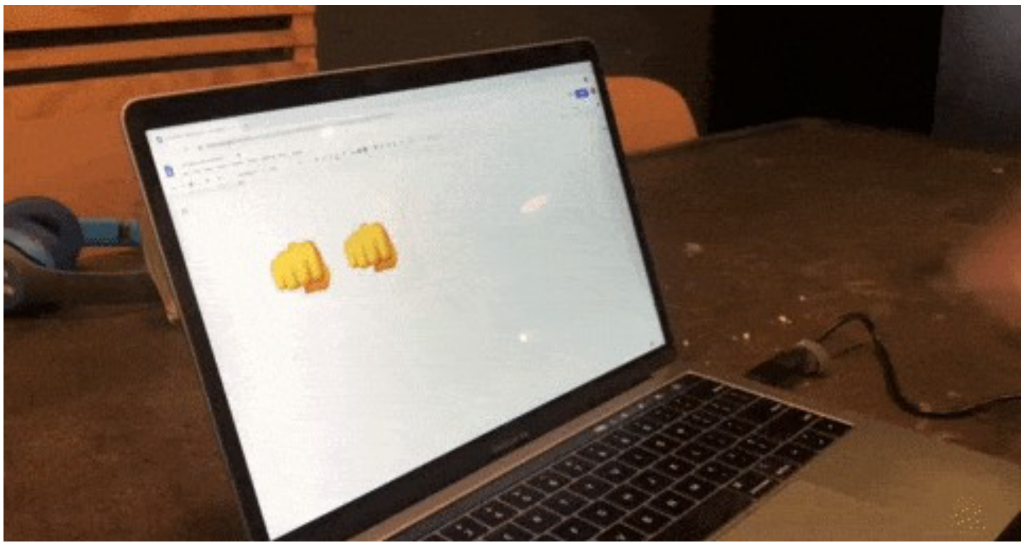
```
    print(line)
    with open(CSV_PATH, "a", encoding="utf-8") as f:
        f.write(line + "\n")

async def main():
    print("Connecting to", ADDRESS)
    async with BleakClient(ADDRESS) as client:
        print("Connected:", client.is_connected)

        # Write header if file doesn't exist
        try:
            with open(CSV_PATH, "r", encoding="utf-8"):
                pass
        except FileNotFoundError:
            with open(CSV_PATH, "w", encoding="utf-8") as f:
                f.write("timestamp,ctr,ax_g,ay_g,az_g\n")

        await client.start_notify(CHAR_UUID, handle_notify)
        print("Streaming ACCEL... Ctrl+C to stop.")
        while True:
            await asyncio.sleep(1)

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("\nStopped.")
```

Py Script for MacOS:

```
import asyncio
from datetime import datetime
from bleak import BleakClient, BleakScanner

# ---------- CONFIG ----------
TARGET_NAME = "XIAO_IMU_EVT"  # <-- ONLY change this to match BLE.setLocalName(...)
TARGET_SERVICE_UUID = "12345678-1234-5678-1234-56789abcdef0"  # optional filter
(recommended)
CHAR_UUID = "12345678-1234-5678-1234-56789abcdef1"
CSV_PATH = "accel_log.csv"
SCAN_TIMEOUT = 8.0
# ---------------------------
```

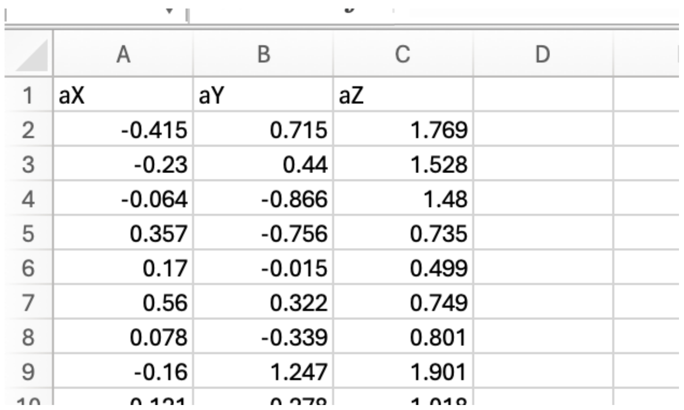
```python
def int16_le(b0, b1):
    return int.from_bytes(bytes([b0, b1]), byteorder="little", signed=True)

def handle_notify(sender, data: bytearray):
    # Expect exactly 8 bytes: ctr + ax + ay + az (int16 little-endian)
    if len(data) != 8:
        try:
            txt = data.decode("utf-8", errors="ignore").strip()
            print("Non-8B packet:", len(data), "as text:", txt)
        except Exception:
            print("Non-8B packet:", len(data), "raw:", list(data))
        return

    ctr = int16_le(data[0], data[1])
    ax_mg = int16_le(data[2], data[3])
    ay_mg = int16_le(data[4], data[5])
    az_mg = int16_le(data[6], data[7])

    ax_g = ax_mg / 1000.0
    ay_g = ay_mg / 1000.0
    az_g = az_mg / 1000.0

    ts = datetime.now().isoformat(timespec="milliseconds")
    line = f"{ts},{ctr},{ax_g:.3f},{ay_g:.3f},{az_g:.3f}"

    print(line)
    with open(CSV_PATH, "a", encoding="utf-8") as f:
        f.write(line + "\n")

def ensure_csv_header():
    try:
        with open(CSV_PATH, "r", encoding="utf-8"):
            pass
    except FileNotFoundError:
        with open(CSV_PATH, "w", encoding="utf-8") as f:
            f.write("timestamp,ctr,ax_g,ay_g,az_g\n")

async def find_target_device():
    print(f"Scanning for BLE devices ({SCAN_TIMEOUT}s)...")
    devices = await BleakScanner.discover(timeout=SCAN_TIMEOUT)

    # Print what macOS sees (addresses look like UUIDs on macOS)
    for d in devices:
        print(f"  - name={d.name!r}  address={d.address}")

    # Prefer service-UUID filter when available (more reliable than name)
    if TARGET_SERVICE_UUID:
        svc = TARGET_SERVICE_UUID.lower()
        print(f"Searching for device advertising service UUID {TARGET_SERVICE_UUID} ...")
        device = await BleakScanner.find_device_by_filter(
            lambda d, ad: svc in [s.lower() for s in (ad.service_uuids or [])],
            timeout=SCAN_TIMEOUT,
        )
        if device:
            return device

    # Fallback: match by name substring
    if TARGET_NAME:
        for d in devices:
            if d.name and TARGET_NAME.lower() in d.name.lower():
                return d

    return None

async def main():
    device = await find_target_device()
    if not device:
        raise RuntimeError(
            "Target device not found.\n"
            "- Make sure it is advertising BLE\n"
            "- Disconnect it from Windows/phone/other centrals\n"
            "- Set TARGET_NAME correctly (must match BLE.setLocalName)\n"
            "- Or verify TARGET_SERVICE_UUID matches your firmware"
        )

    print("Connecting to:", device.name, device.address)
    ensure_csv_header()

    async with BleakClient(device) as client:
        print("Connected:", client.is_connected)

        await client.start_notify(CHAR_UUID, handle_notify)
        print("Streaming ACCEL... Ctrl+C to stop.")
        while True:
            await asyncio.sleep(1)

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("\nStopped.")
```

- Now Try to receive this IMU data from the device via BLE.
- You can stream and log it onto your PC. We will be using python for this step.
- Please do not forget to install the ‘bleak’ library.
- Run this python script
- Set the csv to punch.csv and perform punching gesture.
- Repeat the same movement at least 10 times; the more training data, the better!
- Repeat the same steps for flex gesture and set that csv to flex.csv



3. Checking the CSV files:

Open the punch.csv and flex.csv and make sure that the first line of the CSV files is aX, aY, aZ. Check below for reference:



	A	B	C	D
1	aX	aY	aZ	
2	-0.415	0.715	1.769	
3	-0.23	0.44	1.528	
4	-0.064	-0.866	1.48	
5	0.357	-0.756	0.735	
6	0.17	-0.015	0.499	
7	0.56	0.322	0.749	
8	0.078	-0.339	0.801	
9	-0.16	1.247	1.901	

4. Training Data in TensorFlow:

Here, we will use the Google Colab project that was created by the Arduino team. Colab provides a Jupyter notebook that allows us to run our TensorFlow training in a web browser. The collab will set up the environment, train the model, convert the model into a TensorFlow Lite mode, and encode the model in an Arduino header file.

Click here to access the Google Colab project.

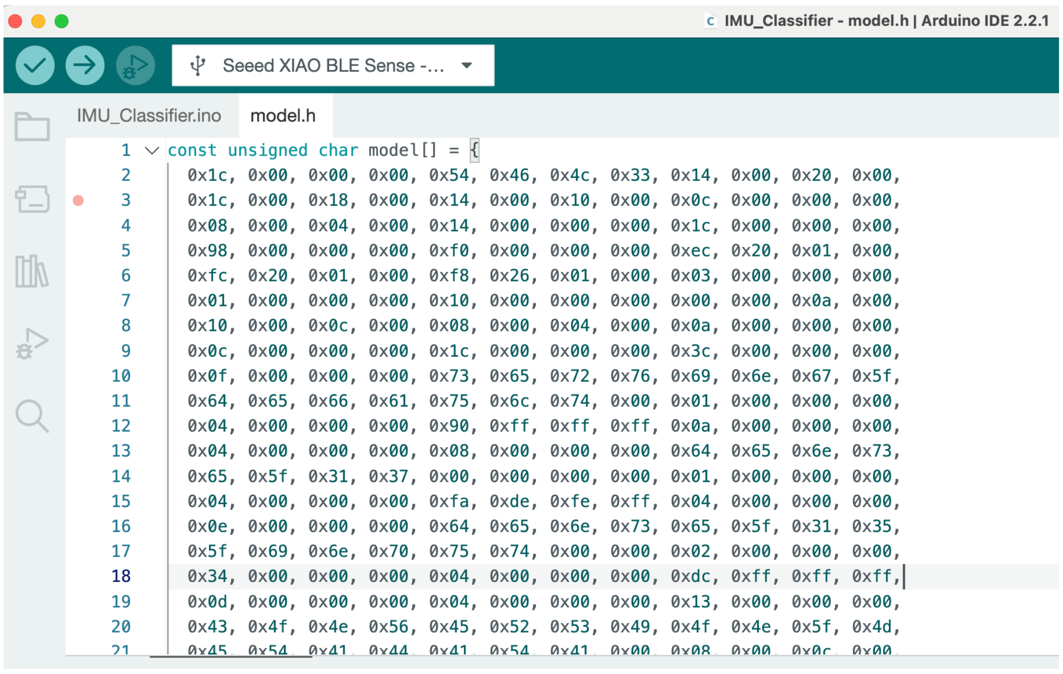
Upload the collected data by clicking on "Files" which can be found on the right menu. Then drag and drop the CSV files.

5. Modifications in Google Colab Project:

Comment out the Gyroscope data, which we will not use for this phase. When we define the tensors, you will need to remove some lines from the plot block that are for the Gyroscope.

6. Uploading the Code to Xiao Board:

On Arduino IDE, open "IMU_Classifier" from File > Examples > Seeed Arduino LSM6D3. Replace model.h content with the model you downloaded from the Colab notebook.


IMU_Classifier - model.h | Arduino IDE 2.2.1
Seeed XIAO BLE Sense -...
IMU_Classifier.ino
model.h

```
const unsigned char model[] = {
  0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x14, 0x00, 0x20, 0x00,
  0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0c, 0x00, 0x00, 0x00,
  0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00,
  0x98, 0x00, 0x00, 0x00, 0xf0, 0x00, 0x00, 0x00, 0xec, 0x20, 0x01, 0x00,
  0xfc, 0x20, 0x01, 0x00, 0xf8, 0x26, 0x01, 0x00, 0x03, 0x00, 0x00, 0x00,
  0x01, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0a, 0x00,
  0x10, 0x00, 0x0c, 0x00, 0x08, 0x00, 0x04, 0x00, 0x0a, 0x00, 0x00, 0x00,
  0x0c, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00,
  0x0f, 0x00, 0x00, 0x00, 0x73, 0x65, 0x72, 0x76, 0x69, 0x6e, 0x67, 0x5f,
  0x64, 0x65, 0x66, 0x61, 0x75, 0x6c, 0x74, 0x00, 0x01, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00, 0x90, 0xff, 0xff, 0xff, 0x0a, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x64, 0x65, 0x6e, 0x73,
  0x65, 0x5f, 0x31, 0x37, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00, 0xfa, 0xde, 0xfe, 0xff, 0x04, 0x00, 0x00, 0x00,
  0x0e, 0x00, 0x00, 0x00, 0x64, 0x65, 0x6e, 0x73, 0x65, 0x5f, 0x31, 0x35,
  0x5f, 0x69, 0x6e, 0x70, 0x75, 0x74, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00,
  0x34, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0xdc, 0xff, 0xff, 0xff,
  0x0d, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x00,
  0x43, 0x4f, 0x4e, 0x56, 0x45, 0x52, 0x53, 0x49, 0x4f, 0x4e, 0x5f, 0x4d,
  0x45, 0x54, 0x41, 0x44, 0x41, 0x54, 0x41, 0x00, 0x08, 0x00, 0x0c, 0x00,
```

Now, open the serial monitor and perform your gestures! You should see the confidence of each gesture printed (0 = low confidence, 1 = high confidence).

Phase II (Activity Recognition)

In this project, we will collect data to build an activity tracker based on the Xiao board. By the end of this project, we will be able to detect several different activities. We will collect data on daily physical activities. Then, we will create a dataset from the collected data. Therefore, we will have a large dataset to work with.

1. Data collection: We will collect data using the built-in IMU (LSM6DS3) on the Arduino board. We learned from the previous pahase that IMU could help us detect daily physical activities. However, to build and test a system, we need to have sufficient data from a large group of participants. Therefore, we follow a specific protocol to collect data that can be merged and analyzed as the final dataset. The activities that we are interested in are:

1. Sitting/Working with a computer
2. Standing
3. Walking
4. Ascending stairs
5. Descending stairs

We will use our smartphone timer to control the duration of each activity. For each activity, we will collect **1 minute of data**, followed by a **1-minute rest period**. The IMU sampling frequency must be set to **100 Hz (100 samples per second)**. This results in **6,000 samples per activity** (60 seconds × 100 Hz) and **30,000 samples total** for five activities.

Instead of storing data on the microSD card, we will stream the IMU data over **Bluetooth Low Energy (BLE)** and log it directly on the PC using our Python receiver script. The PC will automatically attach timestamps using the computer's system clock. This removes the need for an onboard RTC while still providing accurate timing for labeling.

During data collection, the board should be placed in the participant's pocket, and it is important that the board orientation is kept **consistent across all participants** to avoid introducing unwanted variation in the dataset. The activities should be performed as shown in the reference images.

Because the timestamps come from the PC, we can synchronize activity labels using the same system clock. For example, if walking occurs from **5:00 PM to 5:01 PM**, we can later label that portion of the logged CSV data as *walking*. This ensures accurate alignment between the event timer and the recorded sensor stream.

3. Labeling the data:

a. WE can log the data in the same – "Sitting.csv", "Standing.csv"…etc

b. We will need a separate file for each physical activity, so you will need to break the collected data into five different CSV files.

c. Remove time and label columns and Save the CSV file for the next step.

4. Training the machine learning model: Use the same Colab notebook, but this time adjust it for 5 labels instead of 2.

5. Deploying the model on the microcontroller: Follow the same process, but make sure to adjust the Arduino code to increase the number of labels from 2 to 5.

6. Now add the timestamp to the classification code, and store the current time and current activities into the SD card, similar to this example:

10:45:5, Sitting

10:45:18, Walking

5. What to Submit:

- Your collected data for both phases
- The trained models (model.h files)
- Screenshots of the graphs during the training process.
- The Arduino code for the final version of the Phase II project (step 6).
- Demo (No need to submit; the instructor will check out your work!)