



Wearable Posture Correction Device

A wearable device for knee injury rehabilitation, utilising machine learning and sensor systems to monitor and correct posture in real-time. Visual and haptic cues provide non-intrusive assistance, allowing users to maintain correct posture during rehabilitation without stress or disruption.


Tools:



Keywords:

#Physical Computing #Machine Learning #Wearable Technology
#Bioengineering #Sensor Systems

Website:

 <https://canacechen.com/posturafit.html>

Research

Importance of Posture in Health and Rehabilitation

Posture plays a vital role in maintaining musculoskeletal health, with poor posture being linked to pain, discomfort, and long-term damage (Kendall et al., 2005). This is particularly crucial during the rehabilitation process of knee injuries, where proper posture and movement are key to recovery. For example, patients with anterior cruciate ligament (ACL) injuries must avoid excessive knee bending to prevent strain and facilitate healing (Pamboris et al., 2023). The ability to monitor and correct knee positioning during rehabilitation is therefore essential for avoiding further damage and optimising recovery.



Lumo Lift
(tracks upper body posture)



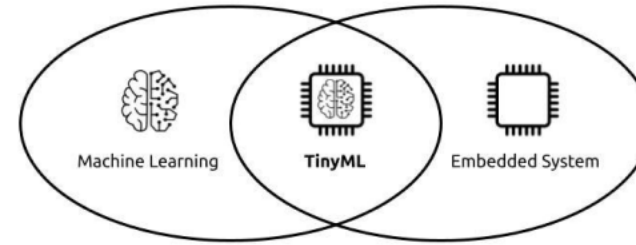
Alex Posture Tracker
(tracks head and neck posture)

Limitations of Current Posture Correction Solutions

Current posture correction solutions often struggle with high power consumption, cost, and usability (Rodgers et al., 2020). In contrast, the designed wearable system integrates both visual (LED) and haptic (vibration) feedback while remaining low power consuming and cost-effective. The wearable design also minimises cognitive load, offering simple, intuitive posture correction that doesn't disturb the user or require attention to a screen (Jain et al., 2020).

Wearables Devices for Posture Monitoring

Wearable technologies have become increasingly prevalent in health and wellness, especially for posture monitoring and correction. Devices such as smart sensors or posture-correcting wearables can provide users with continuous feedback on their body positioning (Patel et al., 2012). The use of wearable technology allows for unobtrusive tracking of posture without disrupting the user's daily activities (Huang et al., 2023). This led to my decision to develop a wearable posture correction system that provides immediate feedback and seamlessly integrates into the user's routine.



Machine Learning in Wearable Health Devices

Machine learning (ML) enhances wearable devices by enabling them to process and analyse data in real time, providing intelligent decision-making capabilities on small, low-power devices. With TinyML, machine learning models can run directly on microcontrollers, allowing for complex analysis without the need for external computing resources. This combination of ML and microcontrollers unlocks new potential for wearable health technologies, making them more accurate, autonomous, and energy-efficient. It enables advanced features in devices that are small, cost-effective, and capable of running on limited power (Srinivasan et al., 2019).

User-experience Design

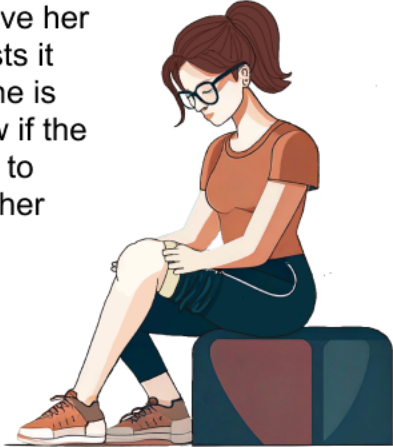


Persona

Sarah, a 32-year-old software developer recovering from a knee injury, so she decided to try the wearable knee strap to help with her rehabilitation. The device monitors whether her knee is straight or bent and provides real-time feedback to help her maintain the correct knee position during her recovery process.

1. Device Setup

Sarah puts the knee sleeve just above her knee and adjusts it comfortably. She is curious to know if the device is going to assist her with her rehabilitation effectively.



2. Powering On

She presses the button, and the device turns on, starting to monitor her knee's posture immediately.



3. Continuous Monitoring and Feedback

The device continuously monitors her knee's posture, glowing green for correct alignment and turning red with gentle haptic feedback (vibration) when the knee position is incorrect.



5. Simple Charging

The battery lasts throughout the day and requires only occasional recharging, which Sarah does by simply connecting it with the USB cable.



4. Comfortable Wear

The knee sleeve is lightweight and non-intrusive, allowing Sarah to wear it while working, exercising, or carrying out daily activities without disruption.



6. Accelerated Recovery

Over time, Sarah finds that the device has become an essential part of her rehabilitation process. With the device's help, Sarah maintains good posture, speeding up her recovery and released her from worrying about overbending her knee.



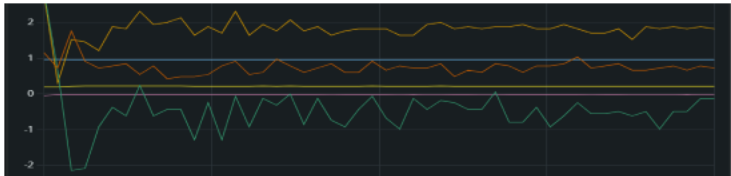
Data Collection



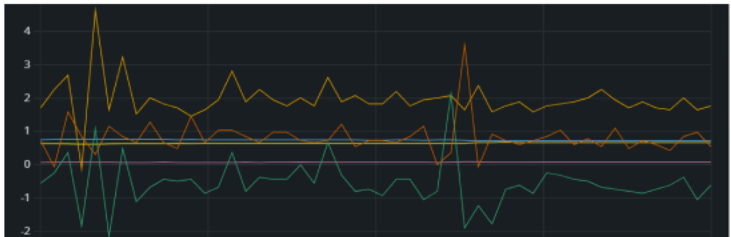
```
sketch: Arduino
void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }
  Serial.println("Arduino Nano 33 BLE");
}

void loop() {
  float x, y, z;
  // get current time
  unsigned long currentMillis = millis();
  // output data every 1000 milliseconds.
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    if (!Serial) return; // no need to do anything
    // calculate the magnitude of the gyroscope data (X, Y, Z)
    float gyro_magnitude = sqrt(gyro_x*gyro_x + gyro_y*gyro_y + gyro_z*gyro_z);
    // calculate the magnitude of the accelerometer data (X, Y, Z)
    float accel_magnitude = sqrt(accel_x*accel_x + accel_y*accel_y + accel_z*accel_z);
    // combine the features and labels
    float features[] = {gyro_magnitude, accel_magnitude, x, y, z};
    float label = correct;
    // output the features and labels
    Serial.print("Features: ");
    for (int i = 0; i < 5; i++) {
      Serial.print(features[i]);
      Serial.print(" ");
    }
    Serial.print("Label: ");
    Serial.println(label);
  }
}
```

The initial data collection only collected the x, y and z axis of the gyroscope. 1000 data were collected from 5 participants. The collected data did not train the model successfully.



x, y, z of accelerometer and gyroscope when standing straight



x, y, z of accelerometer and gyroscope when standing knee bent

The second data collection collected the x, y and z axis for both the gyroscope and the accelerometer. The sensor was positioned horizontally at the front above the knee instead of vertically on the outer side. A total of 2000 data were collected.

Training Model

```
jupyter PostureModel-01 Last Checkpoint: 17 hours ago (autosaved)
# Preview the data
data.head()

Out[11]:
Timestamp  Gyro_X  Gyro_Y  Gyro_Z  Accel_X  Accel_Y  Accel_Z  Label
0         1000    1.80   -1.04   4.76   0.19   0.00   -0.11  Wrong
1         12000    1.28   0.73   8.32   0.28   0.04   0.01  Wrong
2         12640    1.02   2.82   1.26   0.07   0.41   0.83  Correct
3         12648    0.08   -1.04   2.28   0.01   0.08   0.06  Correct
4         13000    5.86   5.88   4.39   0.32   0.99   -0.15  Wrong

In [12]: # Drop participant and Timestamp columns
data = data.drop(columns=['Timestamp'])

# Map 'correct' and 'wrong' labels to 1 and 0
data['label'] = data['label'].map({'correct': 1, 'wrong': 0})

# Normalize the gyroscope data (X, Y, Z) by subtracting the mean and dividing by the standard deviation
data[['Gyro_X', 'Gyro_Y', 'Gyro_Z']] = (data[['Gyro_X', 'Gyro_Y', 'Gyro_Z']] - data[['Gyro_X', 'Gyro_Y', 'Gyro_Z']].mean()) / data[['Gyro_X', 'Gyro_Y', 'Gyro_Z']].std()

# Normalize the accelerometer data (X, Y, Z) by subtracting the mean and dividing by the standard deviation
data[['Accel_X', 'Accel_Y', 'Accel_Z']] = (data[['Accel_X', 'Accel_Y', 'Accel_Z']] - data[['Accel_X', 'Accel_Y', 'Accel_Z']].mean()) / data[['Accel_X', 'Accel_Y', 'Accel_Z']].std()

# Calculate the magnitude of the gyroscope data (X, Y, Z)
data['Gyro_magnitude'] = np.sqrt(data['Gyro_X']**2 + data['Gyro_Y']**2 + data['Gyro_Z']**2)

# Calculate the magnitude of the accelerometer data (X, Y, Z)
data['Accel_magnitude'] = np.sqrt(data['Accel_X']**2 + data['Accel_Y']**2 + data['Accel_Z']**2)

# Update the feature set to include the gyroscope and accelerometer data, along with their magnitudes
features = data[['Gyro_X', 'Gyro_Y', 'Gyro_Z', 'Gyro_magnitude', 'Accel_X', 'Accel_Y', 'Accel_Z', 'Accel_magnitude']]

# Separate the features and labels
X = features
y = data['label']

# Check the processed data
X.head(), y.head()
```

Processing data

```
jupyter PostureModel-01 Last Checkpoint: 17 hours ago (unsaved changes)
In [5]: from sklearn.model_selection import train_test_split

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the split data sizes
X_train.shape, X_test.shape, y_train.shape, y_test.shape

Out[5]: ((1568, 8), (392, 8), (1568, 1), (392, 1))

In [10]: import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

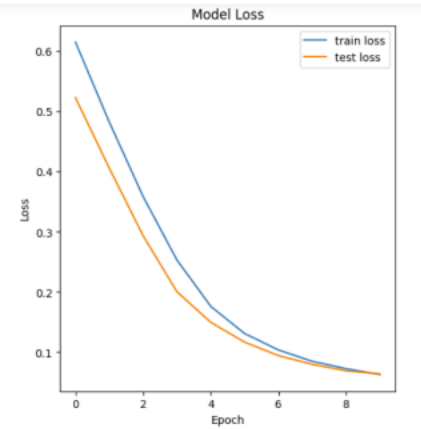
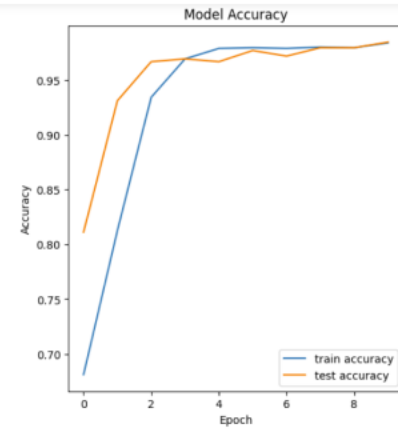
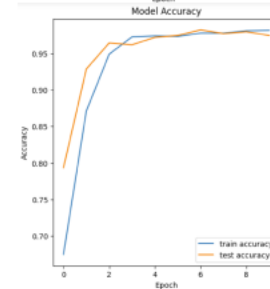
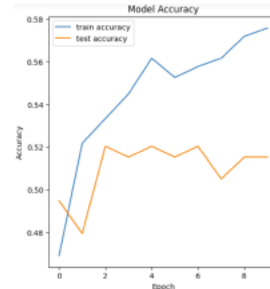
# Define the model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)), # 64 neurons in the hidden layer
    Dense(32, activation='relu'), # Another hidden layer
    Dense(1, activation='sigmoid') # Sigmoid for binary classification (correct or wrong)
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Summarize the model architecture
model.summary()

Model: "sequential_1"
Layer (type) Output Shape Param #
-----
dense_1 (Dense) (None, 64) 576
dense_2 (Dense) (None, 32) 2080
dense_3 (Dense) (None, 1) 33
Total params: 2,689
Trainable params: 2,689
Nontrainable params: 0
```

Designing model



```
In [12]: # Evaluate the model on the test data
test_loss, test_acc = model.evaluate(X_test, y_test)

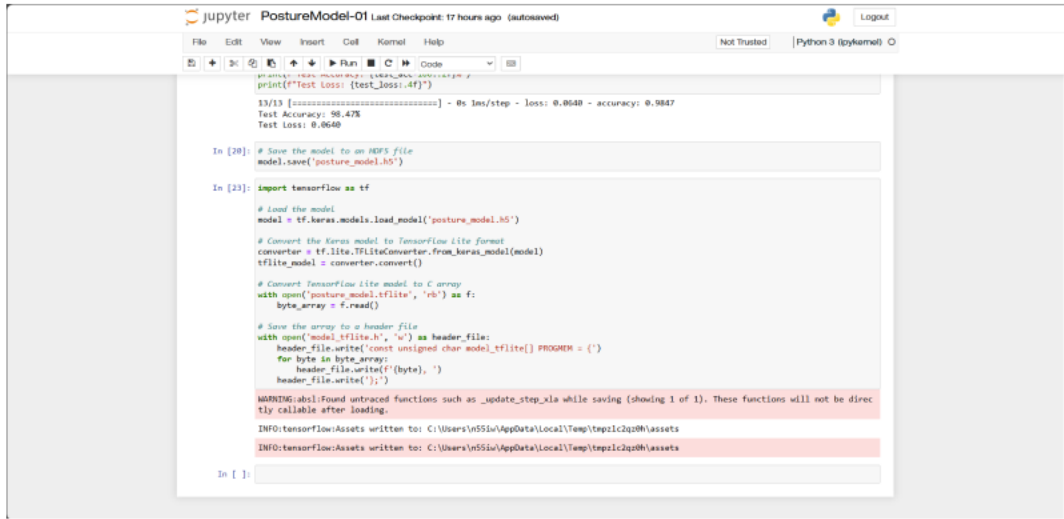
print(f"Test Accuracy: {(test_acc*100):.2f}%")
print(f"Test Loss: {(test_loss:.4f)}")

13/13 [=====] - 0s 1ms/step - loss: 0.0640 - accuracy: 0.9847
Test Accuracy: 98.47%
Test Loss: 0.0640
```

With the initial dataset, the trained mode only got 51% accuracy. With the second dataset, the model was train to 98.47% accuracy (97.54% before fine-tuning).

Feedback Communication

Implementing the trained model to microcontroller



```
print("Test Loss: %f" % test_loss)
print("Test Accuracy: %f" % test_acc)

13/13 [#####] - 0s 1ms/step - loss: 0.0640 - accuracy: 0.9847
Test Accuracy: 98.47%
Test Loss: 0.0640

In [20]: # Save the model to an HDF5 file
model.save('posture_model.h5')

In [23]: import tensorflow as tf

# Load the model
model = tf.keras.models.load_model('posture_model.h5')

# Convert the Keras model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

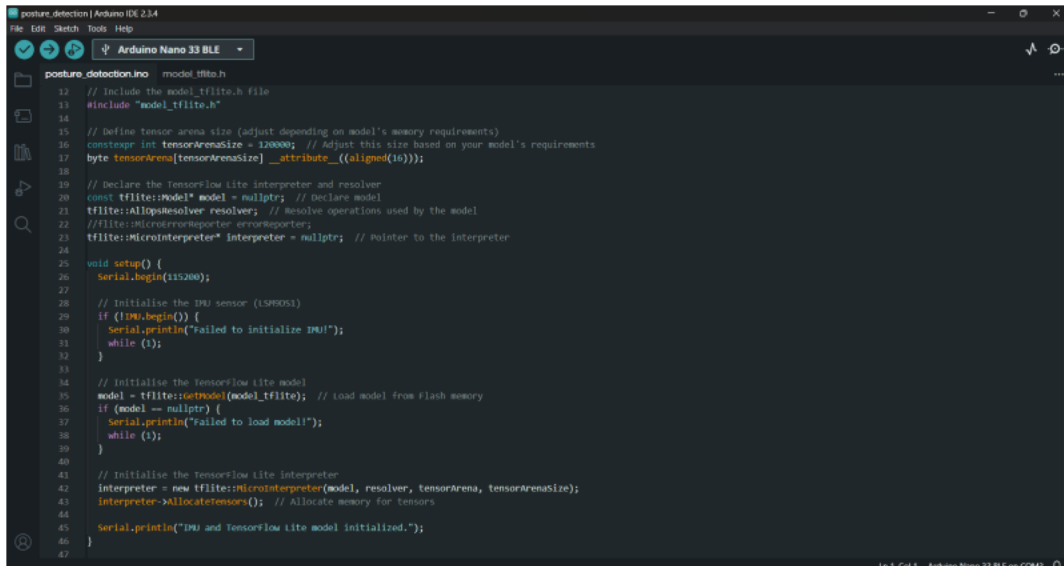
# Convert TensorFlow Lite model to C array
with open('posture_model.tflite', 'wb') as f:
    byte_array = tflite_model
    f.write(byte_array)

# Save the array to a header file
with open('model_tflite.h', 'w') as header_file:
    header_file.write('const unsigned char model_tflite[] PROGMEM = {')
    for byte in byte_array:
        header_file.write('%d, ' % byte)
    header_file.write('};')

WARNING:absl:Found untraced functions such as _update_step_xla while saving (showing 1 of 1). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: C:\Users\N55iu\AppData\Local\Temp\tpcic2q0h\assets
INFO:tensorflow:Assets written to: C:\Users\N55iu\AppData\Local\Temp\tpcic2q0h\assets

In [ ]:
```

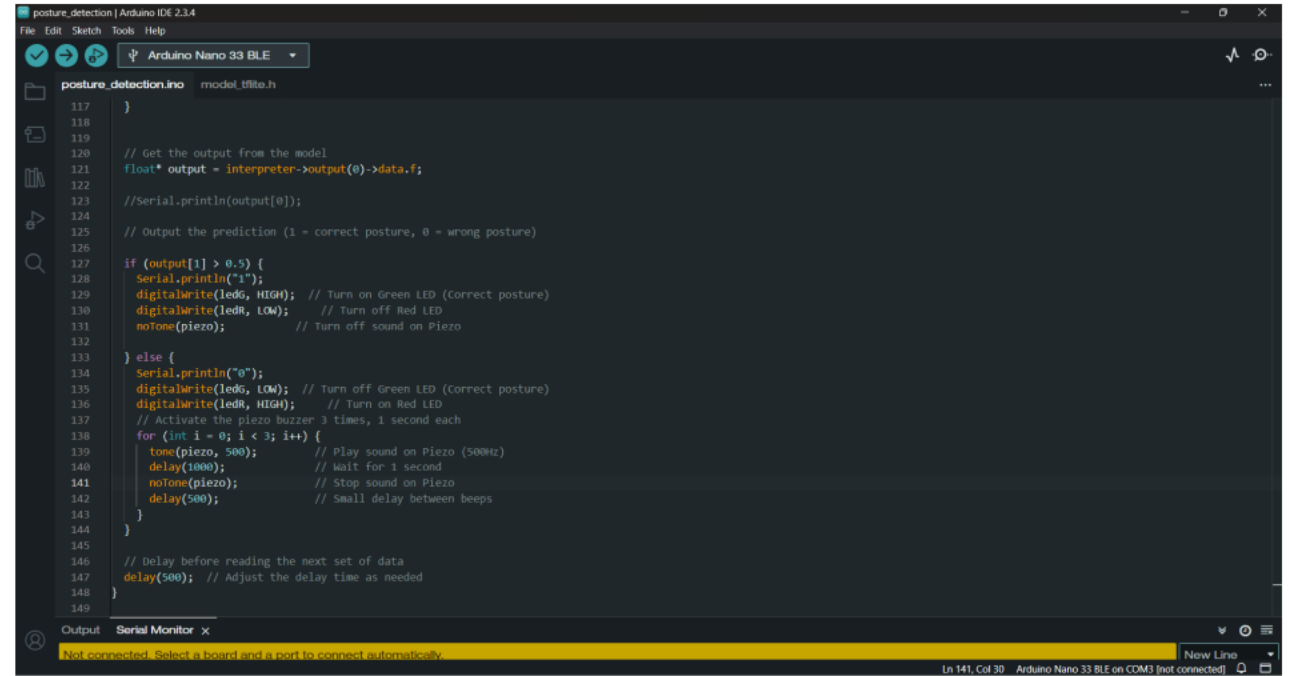
Converting the Model for Deployment (TensorFlow Lite Format)



```
12 // Include the model_tflite.h file
13 #include "model_tflite.h"
14
15 // Define tensor arena size (adjust depending on model's memory requirements)
16 constexpr int tensorArenaSize = 128000; // Adjust this size based on your model's requirements
17 byte tensorArena[tensorArenaSize] __attribute__((aligned(16)));
18
19 // Declare the TensorFlow Lite interpreter and resolver
20 const tflite::Model* model = nullptr; // Declare model
21 tflite::AllOpsResolver resolver; // Resolve operations used by the model
22 // tflite::MicroErrorReporter errorReporter;
23 tflite::MicroInterpreter* interpreter = nullptr; // Pointer to the interpreter
24
25 void setup() {
26     Serial.begin(115200);
27
28     // Initialize the IMU sensor (LSM9DS1)
29     if (!IMU.begin()) {
30         Serial.println("Failed to initialize IMU!");
31         while (1);
32     }
33
34     // Initialize the TensorFlow Lite model
35     model = tflite::LoadModel(model_tflite); // Load model from Flash memory
36     if (model == nullptr) {
37         Serial.println("Failed to load model!");
38         while (1);
39     }
40
41     // Initialize the TensorFlow Lite interpreter
42     interpreter = new tflite::MicroInterpreter(model, resolver, tensorArena, tensorArenaSize);
43     interpreter->AllocateTensors(); // Allocate memory for tensors
44
45     Serial.println("IMU and TensorFlow Lite model initialized.");
46 }
47
```

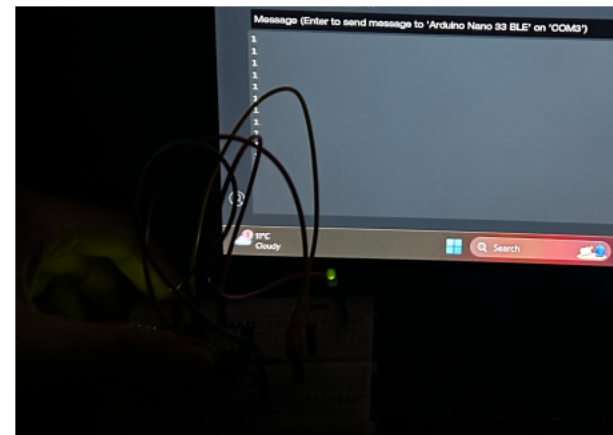
Combining the Model with IMU Sensor for Real-time Detection

Setting Indicator Light and Haptic Feedback

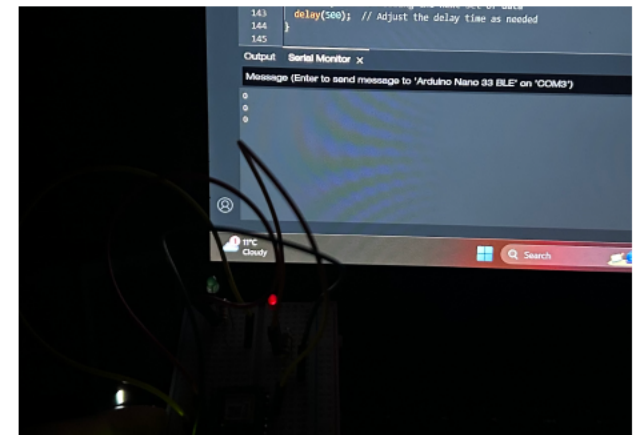


```
117 }
118
119 // Get the output from the model
120 float* output = interpreter->output(0)->data.f;
121
122 // Serial.println(output[0]);
123
124 // Output the prediction (1 = correct posture, 0 = wrong posture)
125
126
127 if (output[0] > 0.5) {
128     Serial.println("1");
129     digitalWrite(ledG, HIGH); // Turn on Green LED (Correct posture)
130     digitalWrite(ledR, LOW); // Turn off Red LED
131     noTone(piezo); // Turn off sound on Piezo
132 }
133 else {
134     Serial.println("0");
135     digitalWrite(ledG, LOW); // Turn off Green LED (Correct posture)
136     digitalWrite(ledR, HIGH); // Turn on Red LED
137     // Activate the piezo buzzer 3 times, 1 second each
138     for (int i = 0; i < 3; i++) {
139         tone(piezo, 500); // Play sound on Piezo (500Hz)
140         delay(1000); // Wait for 1 second
141         noTone(piezo); // Stop sound on Piezo
142         delay(500); // Small delay between beeps
143     }
144 }
145
146 // Delay before reading the next set of data
147 delay(500); // Adjust the delay time as needed
148
149
```

Communicating Detection Result to User through Visual and Haptic Cues



Green Light for Correct Posture



Red Light + Haptic Vibration for Wrong Posture



Result Showcase

