AMRITA
VISHWA VIDYAPEETHAM

HARDWARE REPORT

24AIM113 & 24AIM114

Introduction to NN, CNN and GNN

Analog system design

Team Members

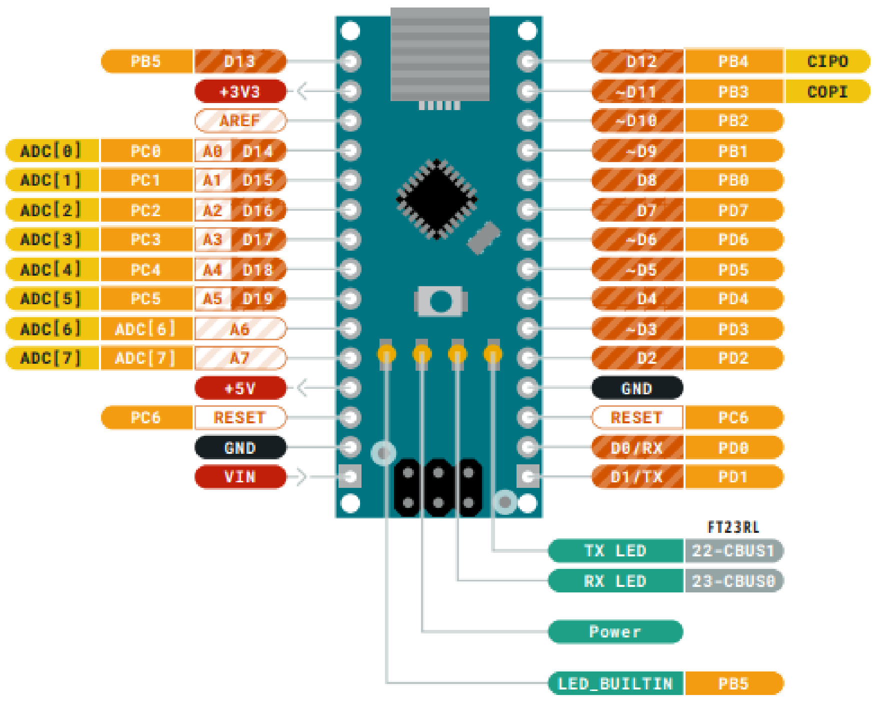
Group 7

CB.AI.U4AIM24004	ANITRA R
CB.AI.U4AIM24028	NARESH L
CB.AI.U4AIM24029	NIMISHA PATEL
CB.AI.U4AIM24050	YATISH S

Faculty In-Charge: Dr. Amrutha V & Dr. Snigdhatanu Acharya

Components used and their specifications

1. Arduino Nano



Operating voltage	Input voltage	Power consumption	Flash Memory	Clock Speed	DC per I/O Pins	Analog IN pins	Digital I/O Pins
5V	7-12V	19 mA	32 KB	16 MHz	20 mA	8	22 (6 are PWM)

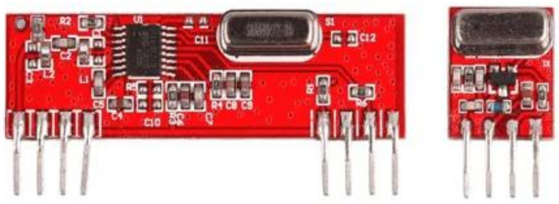
Arduino Nano is a small, complete and breadboard-friendly board based on the ATmega328. It lacks only a DC power jack and works with a Mini-B-USB cable instead of a standard one.

2. Silicon Tube Peristaltic Pump

Voltage	Operating current	Operating Temperature (°C)	Noise Level (Db)	Length (mm)	Width (mm)	Height (mm)	Flow rate (ml/min)
6 VDC	0.35	0-40	<40	67	55	41	10.5

A **peristaltic pump**, also commonly known as a **roller pump**, is a type of positive displacement pump used for pumping a variety of fluids. The fluid is contained in a flexible tube fitted inside a circular pump casing. Most peristaltic pumps work through rotary motion, though linear peristaltic pumps have also been made. The rotor has a number of "wipers" or "rollers" attached to its external circumference, which compress the flexible tube as they rotate by. The part of the tube under compression is closed, forcing the fluid to move through the tube.

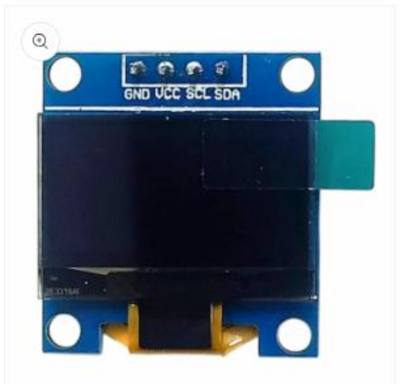
3. 433 MHz RF Transmitter and RF Receiver:



Range in open space (meters)	RX receiver frequency (MHz)	RX Typical Sensitivity (Dbm)	RX supply current (mA)	RX Operating Voltage (V)	TX Frequency Range (MHz)	TX Supply voltage (V)	TX Out Put Power (Dbm)
100	433	105	3.5	5	433.92	3-6	4-12

The **RF transmitter** module is very simple to operate and offers low current consumption (typical. 11mA). Data can be supplied directly from a microprocessor or encoding device, thus keeping the component count down and ensuring a low hardware cost.

4. SSD1306 OLED Display Module



Resolution	V_{in} (or) V_{cc}	Display size	Display type	Dimensions
128 x 64 pixels	3.3-5 V	0.96 inch	OLED Display	2.7cm x 2.8cm

This is the **0.96" OLED display** module. It has 4 pins, amd it is made of 128x64 individual blue OLED pixels, and each one is turned on or off by the controller chip.

It works without a backlight, that is, in a dark environment, its brightness is higher compared to that of an LCD display.

5. AS7263 NIR Spectroscopy Sensor

Sensor	ADC Resolution	Commu-nication	V_{in} (or) V_{cc}	Channels (nm)	Wavelength Accuracy	AOI
Photodiode	16 bits	I2C	2.7-3.6V	610, 680, 730, 760, 810, 860.	±5 nm	20°

The **AS7263 spectrometer** detects wavelengths in the visible range at 610, 680, 730, 760, 810 and 860nm of light, each with 20nm of full-width half-max detection. The board also has multiple ways for us to illuminate objects that we will try to measure for a more accurate spectroscopy reading. There is an onboard LED that has been picked out specifically for this task, as well as two pins to solder our own LED into.

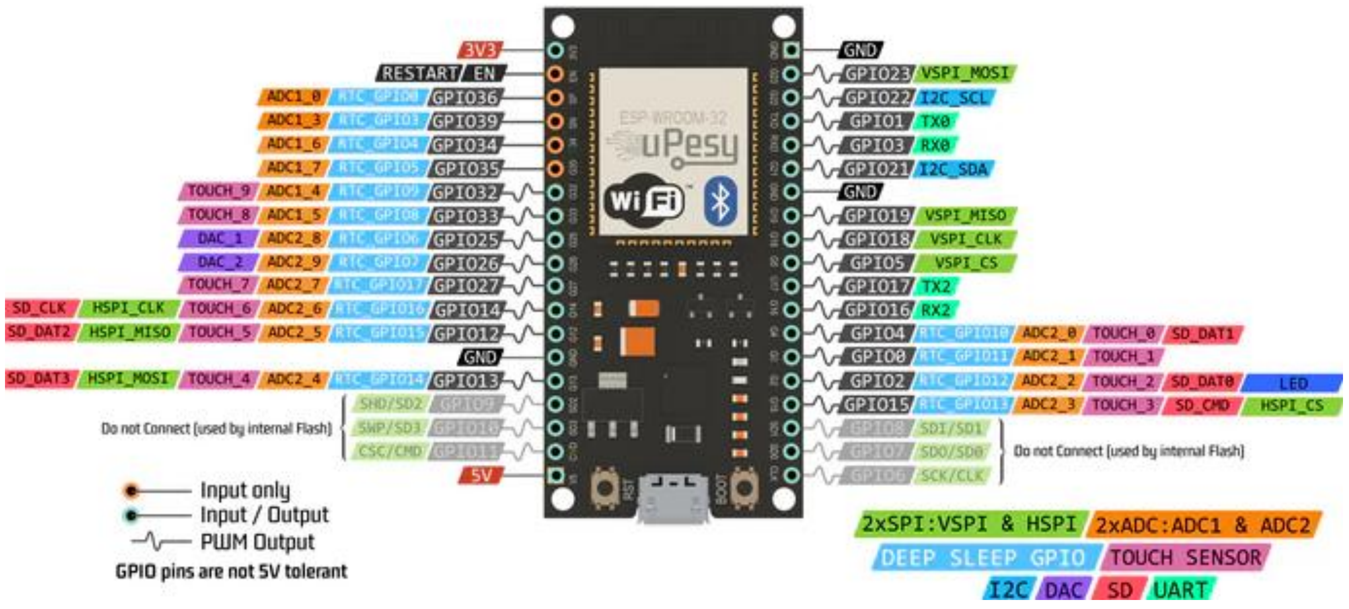
6. **MAX30100 Pulse Oximeter Heart Rate Sensor** (For testing)

Power supply	**Current Draw**	**Red LED Wavelength**	**IR LED Wavelength**	**Temperature Range**	**Temperature Accuracy**
3.3V to 5.5V	~600µA (during Measurements) ~0.7µA (during standby mode)	660 nm	880 nm	-40˚C to +85˚C	±1˚C

The **MAX30100** is a modern, integrated pulse oximeter and heart rate sensor IC, from Analog Devices. It combines two LEDs, a photodetector, optimized optics, and low-noise analog signal processing to detect pulse oximetry (SpO2) and heart rate (HR) signals.

On the right, the MAX30100 has two LEDs – a RED and an IR LED. And on the left is a very sensitive photodetector. The idea is that you shine a single LED at a time, detecting the amount of light shining back at the detector, and, based on the signature, you can measure blood oxygen level and heart rate.

7. ESP32-WROOM-32

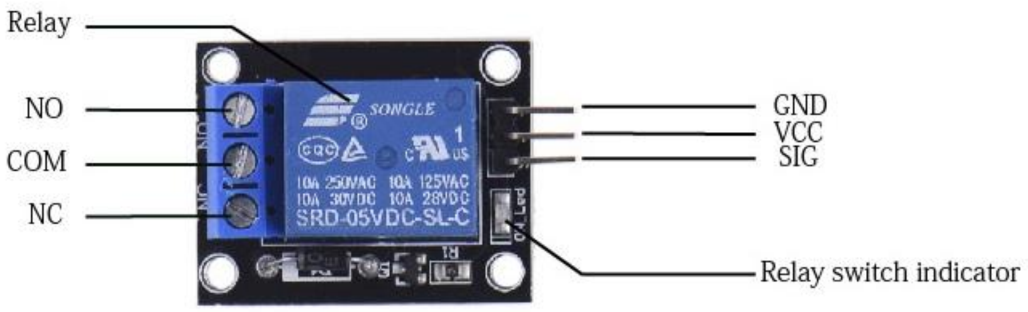


Flash Memory	Processor	Operating voltage (VDC)	Operating current (mA)	Clock frequency (MHz)	Data rate (Mbps)	SRAM Memory (KB)
4 MB	Two Low-Power Xtensa 32-bit LX6 Microprocessors	3.0 V-3.6V	80	80 to 240	150	520

ESP WROOM 32 is a powerful, generic WiFi-BT-BLE MCU module that targets a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks, such as voice encoding, music streaming, and MP3 decoding.

At the core of this module is the ESP32S chip, which is designed to be scalable and adaptive. There are 2 CPU cores that can be individually controlled or powered. In our project, the ESP32 is in full control of the receival of RF transmission from the CGM, the android application and the insulin pump operations.

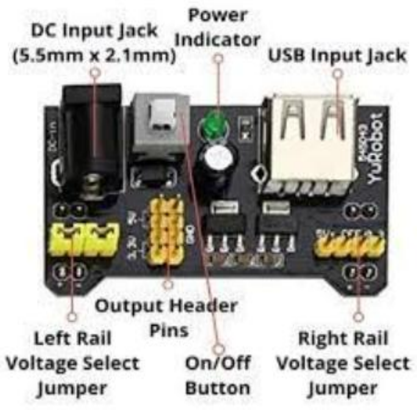
8. **5V Relay Module**



Trigger Voltage	Trigger Current	Maximum Switching Voltage
5 VDC	20 mA	250 VAC @ 10A 30 VDC @ 10 A

A **1-channel 5V control Single-Pole Double-Throw (SPDT) relay board** can be controlled directly via a microcontroller and switch up to 10A at 250 VAC. The inputs are isolated to protect any delicate control circuitry. This might be replaced with MOSFET in the final design.

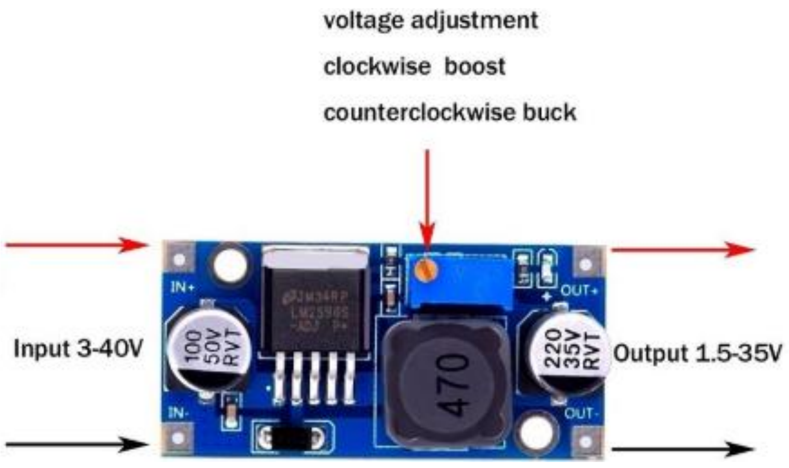
9. **3.3V/5V MB102 Breadboard Power Supply Module:**


DC Input Jack (5.5mm x 2.1mm)
Power Indicator
USB Input Jack
Output Header Pins
Left Rail Voltage Select Jumper
On/Off Button
Right Rail Voltage Select Jumper

Input Voltage	Output Voltage	Maximum output current
6.5-12 V	3.3 V/ 5V	<700 mA

It is a **3.3V/5V MB102 Breadboard Power Supply Module** which provides a dual 5V and 3.3V power rails and has a multi-purpose female USB socket. The module can also output 5V on USB connector or input through the USB connector. The idea behind the usage of this module is that, it can directly power the ESP32, as well as the 6V DC pump (at a lower 5V, hence lower speed) without having to use multiple buck converters.

10. **LM2596 DC-DC Buck Converter** (Alternative)



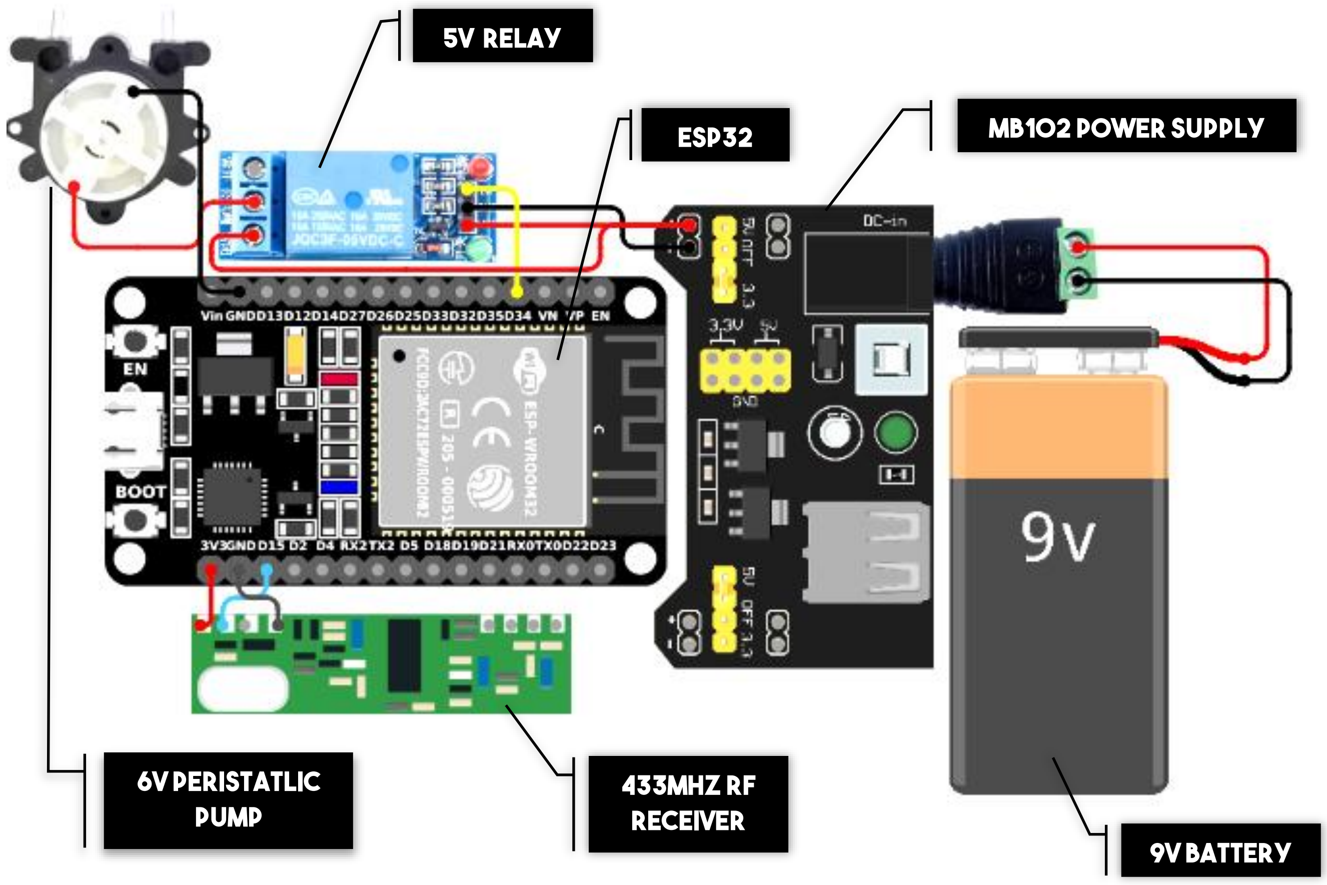
Input voltage	**Output voltage**	**Output current**	**Switching Frequency**	**Operating temperature**	**Conversion efficiency**
3-40V	1.5-35V	Rated current is 2A, maximum 3A	150 KHz	Industrial grade (-40 to +85)	92% (highest)

It is a **step-down (buck) switching regulator**, capable of driving a 3-A load with excellent line and load regulation. These devices are available in fixed output voltages of 3.3V, 5V, 12V, and an adjustable output version.

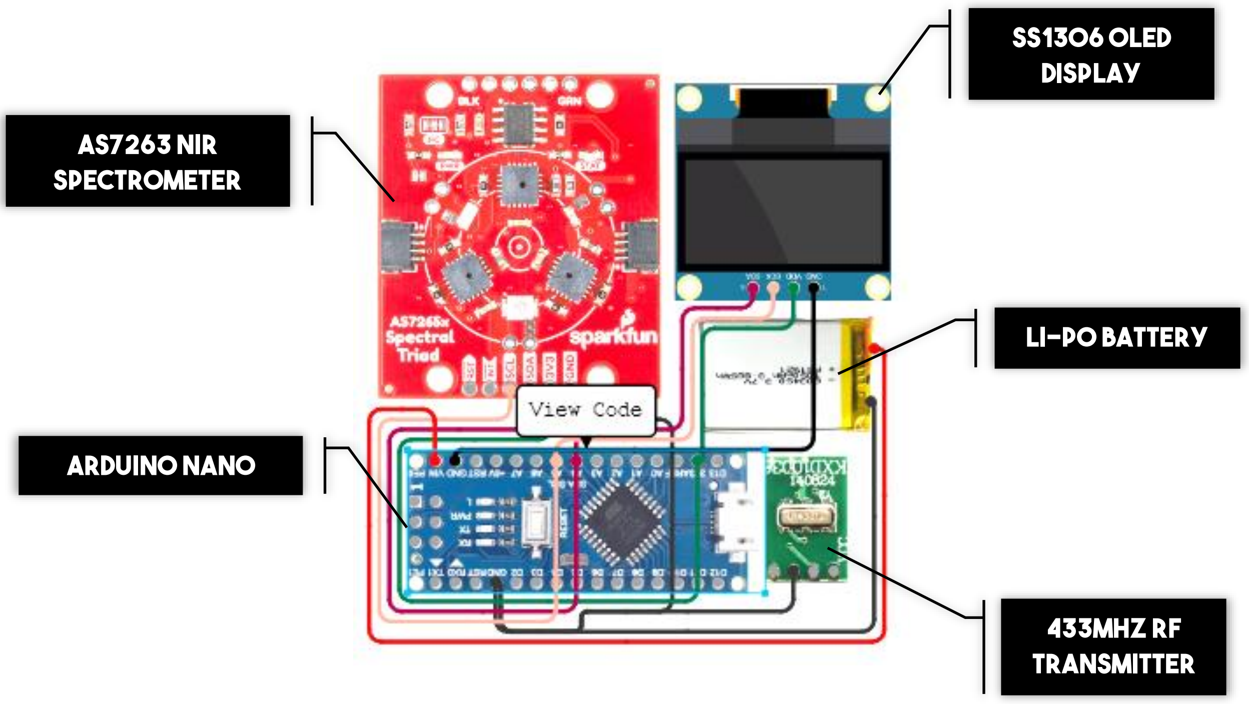
To allow more precise control over the regulation of the 9V coming from our battery, this buck converter could be used. Since the MB102 Power supply module only outputs a fixed voltage of either 5V or 3.3V, the LM2596 can be used to run the peristaltic pump at its maximum potential (6V) with precise control over the voltage.

Circuit Diagram and Pin Connections

1. **Insulin Pump,** controlled by ESP32-WROOM-32.


5V RELAY
ESP32
MB102 POWER SUPPLY
6V PERISTALTIC PUMP
433MHZ RF RECEIVER
9V BATTERY
9v
Vin GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23
EN
BOOT
ESP-WROOM32
DC-in

2. **CGM watch module,** controlled by Arduino Nano.



CODE: ESP32 - 433 MHz Receiver

```
#include <RH_ASK.h>
#include <SPI.h>

RH_ASK rf_driver(2000, 34, 22);

void setup() {
  Serial.begin(115200);
  delay(4000);
  Serial.println("ESP32 433MHz receiver");
  if (RH_PLATFORM == RH_PLATFORM_ESP32)
    Serial.println("RH_PLATFORM_ESP32");
  delay(5000);
  Serial.println("Receiver: rf_driver initialising");
  if (!rf_driver.init()) {
    Serial.println("init failed");
    while (1) delay(1000);
  }
  Serial.println("Receiver: rf_driver initialised");
}

void loop() {
  uint8_t buf[20]={0};
  uint8_t buflen = sizeof(buf);
  // Check if received packet is correct size
  if (rf_driver.recv(buf, &buflen)) {
    // Message received with valid checksum
    Serial.print("Message Received: ");
    Serial.println((char*)buf);
  }
}
```

Arduino Nano - CGM Sensor with MAX30100

```
#include <Wire.h>
#include <EEPROM.h>
#include "MAX30100_PulseOximeter.h"
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <RH_ASK.h>
#include <SPI.h>

#define ENABLE_MAX30100 1
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 32

#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#if ENABLE_MAX30100
#define REPORTING_PERIOD_MS 5000

PulseOximeter pox;
RH_ASK driver;
uint32_t tsLastReport = 0;
uint32_t tsLastSave = 0;

float glucose_records[3] = {0.0, 0.0, 0.0};

void onBeatDetected()
{
    Serial.println("Beat!");
}

void setup()
{
    Serial.begin(115200);
    Serial.println("SSD1306 128x32 OLED TEST");

    if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
        Serial.println(F("SSD1306 allocation failed"));
        for (;;);
    }

    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(20, 18);
    display.print("Pulse OxiMeter");
    int temp1 = 0;
    int temp2 = 40;
    int temp3 = 80;
    display.display();
    delay(2000);
    display.cp437(true);
    display.clearDisplay();

    Serial.print("Initializing pulse oximeter..");
#if ENABLE_MAX30100
    if (!pox.begin()) {
        Serial.println("FAILED");
        for (;;);
    } else {
        Serial.println("SUCCESS");
    }

    pox.setIRLedCurrent(MAX30100_LED_CURR_7_6MA);
    pox.setOnBeatDetectedCallback(onBeatDetected);
#endif

    load_glucose_records();
    print_glucose_records(); // Print loaded records to Serial
}
```

```
void loop()
{
#if ENABLE_MAX30100
    pox.update();
    int bpm = 0;
    int spo2 = 0;
    float glucose_level = 0.0;

    if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
        bpm = pox.getHeartRate();
        spo2 = pox.getSpO2();

        if (bpm > 0 && spo2 > 0) {
            glucose_level = 16714.61 + 0.47 * bpm - 351.045 * spo2 + 1.85 * (spo2
* spo2);
        }

        Serial.print("Heart rate: ");
        Serial.println(bpm);
        Serial.print("SpO2: ");
        Serial.println(spo2);
        Serial.print("Glucose Level: ");
        Serial.println(glucose_level);

        tsLastReport = millis();
        display_data(bpm, spo2, glucose_level);
    }

    // Save the glucose level if it is smaller than 500 every 10 seconds
    if (glucose_level < 500.0 && glucose_level > 0 && millis() - tsLastSave >
10000) {
        save_glucose_level(glucose_level);
        tsLastSave = millis();
    }
#endif
}

void display_data(int bpm, int spo2, float glucose_level)
```

```
{
    display.clearDisplay();

    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.print("BPM:");
    display.println(bpm);

    display.setCursor(0, 10);
    display.print("SpO2:");
    display.println(spo2);

    display.setCursor(0, 20);
    if (glucose_level > 500.0 || glucose_level < 0) {
        display.print("Glucose:MEAS");
    } else {
        display.print("Glucose:");
        display.println(glucose_level);
    }

    // Display last three glucose records
    display.drawLine(88, 0, 88, 32, WHITE);
    for (int i = 0; i < 3; i++) {
        display.setCursor(92, 0 + (i * 10));
        display.print(glucose_records[i]);
    }

    display.display();
}

void save_glucose_level(float glucose_level)
{
    // Shift old records
    glucose_records[2] = glucose_records[1];
    glucose_records[1] = glucose_records[0];
    glucose_records[0] = glucose_level;
    // Save records to EEPROM
    for (int i = 0; i < 3; i++) {
        EEPROM.put(i * sizeof(float), glucose_records[i]);
    }

    print_glucose_records(); // Print saved records to Serial for debugging
}

void load_glucose_records()
{
    for (int i = 0; i < 3; i++) {
        EEPROM.get(i * sizeof(float), glucose_records[i]);
    }
}

void print_glucose_records()
{
    Serial.println("Glucose Records:");
    char msg[10];
    for (int i = 0; i < 3; i++) {
        Serial.print(i);
        Serial.print(": ");
        Serial.println(glucose_records[i]);
        dtostrf(glucose_records[i], 6, 2, msg);
        driver.send((uint8_t *)msg, strlen(msg));
        driver.waitPacketSent();
        delay(1000);
    }

}
```

Interfacing MAX30100 with Arduino Nano

MAX30100 is an optic-based sensor that measures heartrate (BPM) and blood oxygen saturation (SPO2).

It communicates to the Arduino Nano via the I2C protocol.

Pin connections and descriptions:

MAX30100 Pin	Arduino Nano Pin	Description
VCC	3.3V or 5V	Power Supply
GND	GND	Ground
SCL	A5	I2C Clock Line
SDA	A4	I2C Data Line

Primary code functions:

1. **Initialization**
   - **a.** The pox.begin() function Is used for sensor initialization.
   - **b.** onBeatDetected() is a callback function used to detect whenever there is a heartbeat.
2. **Data Reading**
   - **a.** The function pox.update() updates heartrate and SPO2 readings.
   - **b.** The values are accessed using pox.getHeartRate() and pox.getSpO2().
3. **Glucose Calculation**
   - **a.** The glucose level is calculated using the formula that was derived through polynomial regression

     ```
     BGL = 16714.61 + 0.47 * bpm - 351.045 * spo2 + 1.85 * (spo2 * spo2);
     ```

   - **b.** Then, it is displayed on the OLED screen and saved to EEPROM of the Arduino Nano.

Interfacing SSD1306 with Arduino Nano

This 0.96" OLED display is used to show BPM, SPO2 and glucose levels

Pin connections and descriptions:

SSD1306 Pin	Arduino Nano Pin	Description
VCC	3.3V or 5V	Power Supply
GND	GND	Ground
SCL	A5	I2C Clock Line
SDA	A4	I2C Data Line

Primary code functions:

- The Adafruit_SSD1306 library is used to interface with the OLED.
- The display.begin() function Is used for display initialization.
- The display_data() function is used to update the values shown on the screen.

Interfacing Nano with 433MHz RF Transmitter

Once the glucose levels are calculated, the Arduino Nano sends the data over to the ESP32 wirelessly using a 433MHz RF transmitter.

Pin connections and descriptions:

Transmitter Pin	Arduino Nano Pin	Description
VCC	3.3V or 5V	Power Supply
GND	GND	Ground
DATA	D12	Data Transmission

Primary code functions:

- The RH_ASK (RadioHead) library is used for RF based data transmission.
- The driver.send() function transmits the glucose value as a string.
- waitPacketSent() ensures complete transmission before sending another value.

Interfacing ESP32 with 433MHz RF Receiver

The ESP32 receives the glucose readings via the 433MHz RF receiver and prints them to the serial monitor.

Pin connections and descriptions:

Receiver Pin	ESP32 Pin	Description
VCC	3.3V or 5V	Power Supply
GND	GND	Ground
DATA	D12	Data Transmission

Primary code functions:

- Similarly, the RH_ASK (RadioHead) library is used for RF based data transmission.
- Receiver is initialized using the rf_driver.init() function.
- The rf_driver.recv() function checks for incoming glucose data.
- Once the data is received, it is printed on the ESP32's serial monitor.

Libraries used and their purpose

Library	Purpose
Wire.h	Handles I2C communication between Arduino and its peripherals.
EEPROM.h	Used to enable storing and retrieval of glucose readings in Arduino Nano's non-volatile memory.
MAX30100_PulseOximeter.h	Manages the MAX30100 sensor for BPM and SPO2 readings.
Adafruit_GFX.h	Provides graphical functions for displaying text and patterns on all OLED screens.
Adafruit_SSD1306.h	Controls the SSD1306 OLED display specifically.
RH_ASK.h	Implements communication through ASK modulation for the 433MHz RF transmitter and receiver.
SPI.h	Required for the RH_ASK.h module to compile (dependency).

Code explanation - CGM

1. **Library Initialization**

   The imported libraries are:

   i. Wire.h
   ii. EEPROM.h
   iii. MAX30100_PulseOximeter.h
   iv. Adafruit_GFX.h
   v. Adafruit_SSD1306.h
   vi. RH_ASK.h
   vii. SPI.h

2. **Sensor & Display Setup**

   a. Baud rate is set to 115200.
   b. OLED display is initialized and start-up message is displayed.
   c. MAX30100 Pulse Oximeter is initialized.
   d. The glucose records are loaded from the EEPROM

3. **Main Loop**

   a. The MAX30100 is updated every 5 seconds, and the following calculations are made:
      i. Heartrate (BPM) and SPO2 are retrieved.
      ii. If valid, the glucose level is calculated using the formula, else it is set to MEAS.
      iii. The BPM, SPO2 and Glucose Levels are printed to the serial monitor and displayed on the OLED screen.

   b. If the glucose level is within a valid range (<500 mg/dL), it is written to the EEPROM every 10 seconds.
   c. The last glucose value is transmitted via the 433 MHz RF transmitter to the ESP32.

4. **User defined functions:**
   a. **display_data():** Updates the OLED screen with the newest readings.
   b. **save_glucose_level():** Writes the glucose level to the EEPROM.
   c. **load_glucose_records():** Retrieves the glucose level readings from the EEPROM.
   d. **print_glucose_records():** Prints the stored glucose records to the serial monitor and also transmits them via RF.

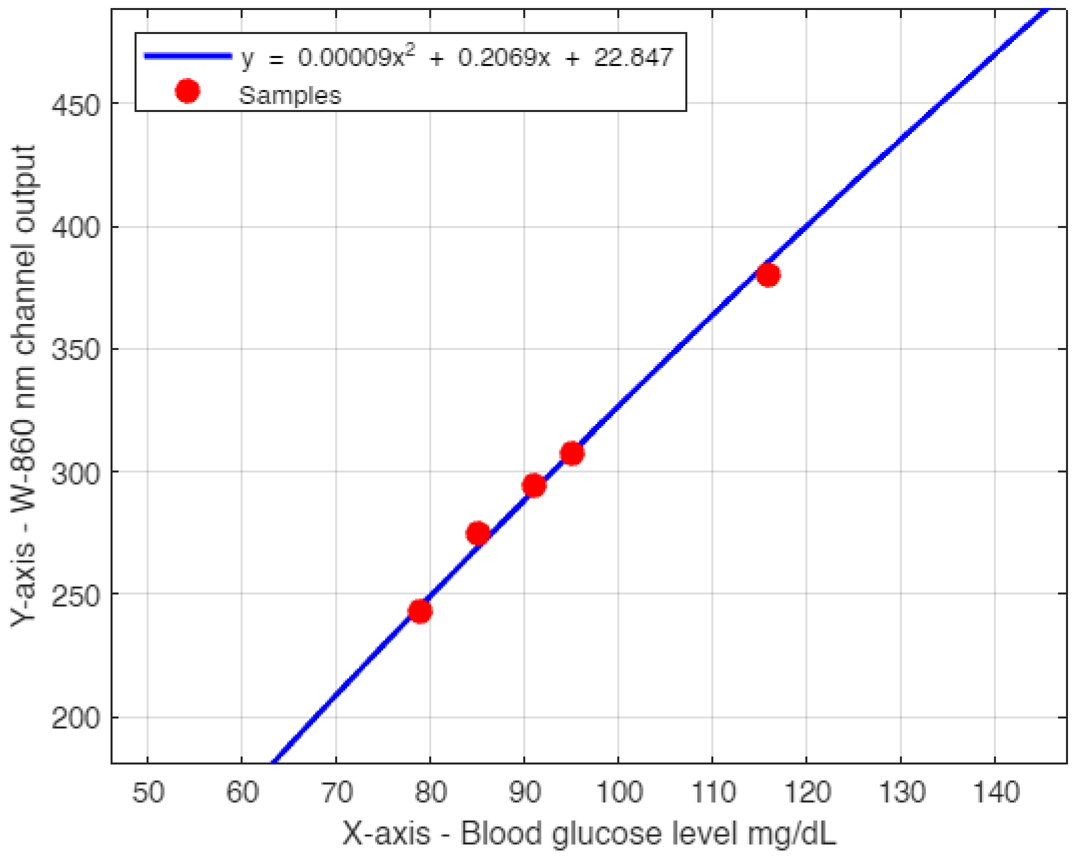
Curve-Fitting Equation

S. No.	Blood Glucose Level	W-860 nm NIR channel analog output
1	79 mg/dL	243.51
2	91 mg/dL	294.38
3	85 mg/dL	274.43
4	116 mg/dL	379.83
5	95 mg/dL	299.71

After fitting a quadratic curve through the points:

$$y = 0.00009x^2 + 0.2069x + 22.847$$

Plot of results



Synthetic Dataset Generation

MATLAB

```matlab
% Generation of dataset
x= randi([200,400],3000,1);
y_0=(.00009.*x.*x) + (0.2069.*x) + 22.847;
y=y_0+normrnd(4,0.3,3000,1);
dataset=[x y];


% Normalize the input features and target values
[input_x_norm, input_ps] = mapminmax(x', 0, 1);
input_x_norm = input_x_norm';
[y_0_norm, output_ps] = mapminmax(y', 0, 1);
y_0_norm = y_0_norm';


% Split data into training (80%) and testing (20%) sets
rng(42);     % to make the distribution between test and train fixed
cv = cvpartition(size(input_x_norm, 1), 'HoldOut', 0.2);
XTrain = input_x_norm(cv.training, :);
YTrain =dlarray( y_0_norm(cv.training, :));
XTest = input_x_norm(cv.test, :);
YTest = (y_0_norm(cv.test, :));


% Convert to dlarray format
XTrain = dlarray(XTrain);
XTest = dlarray(XTest);
```

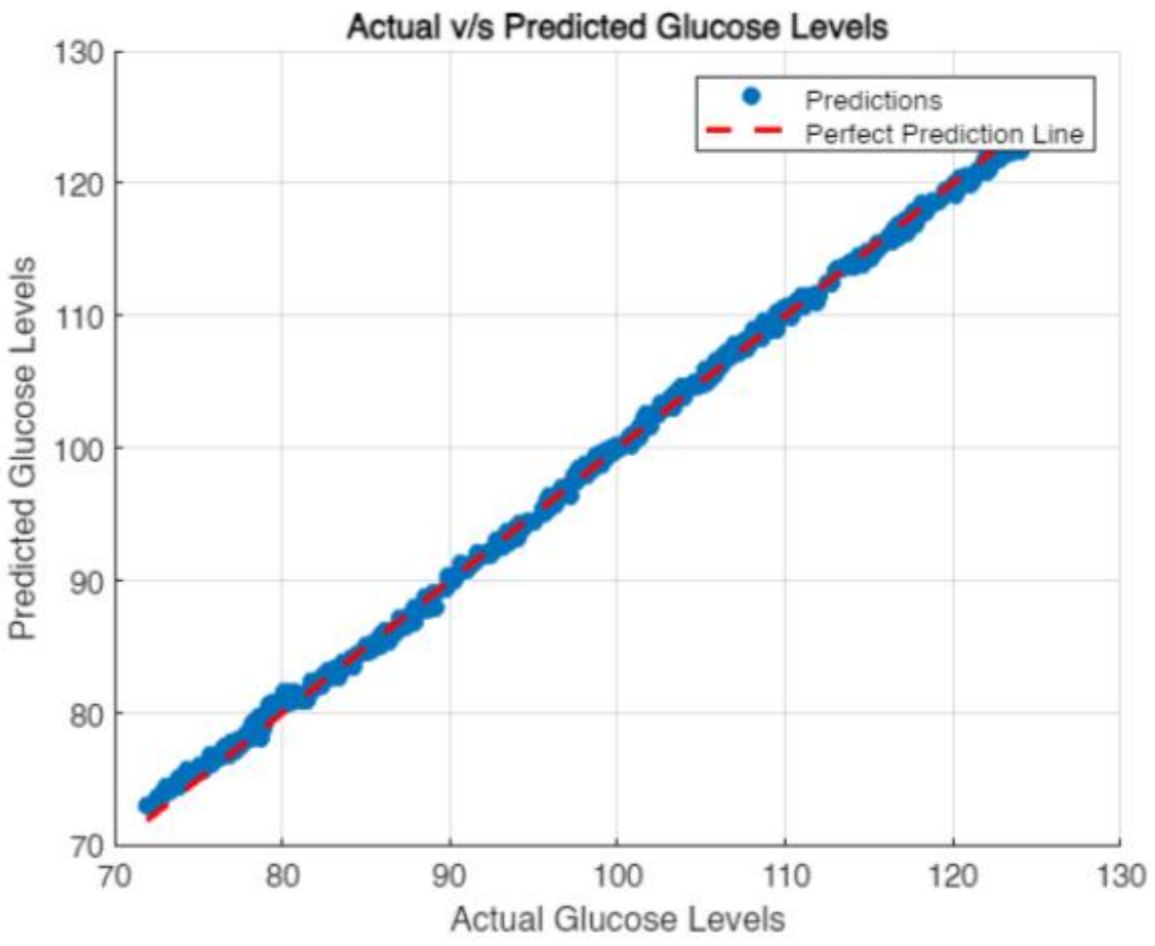
Neural Network Model Training

MATLAB

```matlab
% Building a neural network
net = dlnetwork;
layers = [
    featureInputLayer(1)
    fullyConnectedLayer(64)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(128)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(128)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(256)
    batchNormalizationLayer
    reluLayer
fullyConnectedLayer(256)
    batchNormalizationLayer
    reluLayer
    dropoutLayer(0.4)
fullyConnectedLayer(256)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(64)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(1)
    ];
net = addLayers(net, layers);

plot(net);
% analyzeNetwork(net);

  % defining the loss function - HUBER LOSS
function loss = dlhuber(y_pred, y_true, delta)
    e = y_pred - y_true;
    a = abs(e);
    mask = a <= delta;

    % Quadratic part (resembling L2)
    square_loss = 0.5 * e.^2;

    % Linear part (resembling L1)
    linear_loss = delta * (a - 0.5 * delta);

    % loss function equation
    loss = sum(mask .* square_loss + (~mask) .* linear_loss) / numel(e);
end


% training the model
options = trainingOptions("adam", ...
    "Plots", "training-progress", ...
    "MaxEpochs", 200, ...
    "InitialLearnRate", 0.001, ...
    "LearnRateSchedule", "piecewise", ...
    "LearnRateDropFactor", 0.1, ...
    "LearnRateDropPeriod", 50, ...
    "L2Regularization", 0.0005, ...
    "ValidationData", {XTest, YTest}, ...
    "ValidationFrequency", 90, ...
    "ValidationPatience", 10, ...
    "Shuffle", "every-epoch", ...
    "MiniBatchSize", 96);

huberLoss = @(y_pred, y_true) dlhuber(y_pred, y_true, 0.1);   % loss function

[net, info] = trainnet(XTrain, YTrain, net, huberLoss, options);

% predicting the values on the test data
y_pred_norm = predict(net, XTest);
```

```
% inverse to get the actual glucose values from the normalized data
y_pred = mapminmax('reverse', y_pred_norm.extractdata', output_ps)';
y_true = mapminmax('reverse', YTest', output_ps)';


% accuracy calculation
perc_error = abs((y_true - y_pred) ./ y_true) * 100;
accuracy_perc = 100 - mean(perc_error);


fprintf('Accuracy: %.2f%%\n', accuracy_perc);


% Plot actual vs predicted values
figure;
scatter(y_true, y_pred, 'filled');
hold on;
min_val = min([y_true; y_pred]);
max_val = max([y_true; y_pred]);
plot([min_val, max_val], [min_val, max_val], 'r--', 'LineWidth', 2);
hold off;
xlabel('Actual Glucose Levels');
ylabel('Predicted Glucose Levels');
title('Actual v/s Predicted Glucose Levels');
legend('Predictions','Perfect Prediction Line');
grid on;
```

```
Accuracy: 99.53%
```



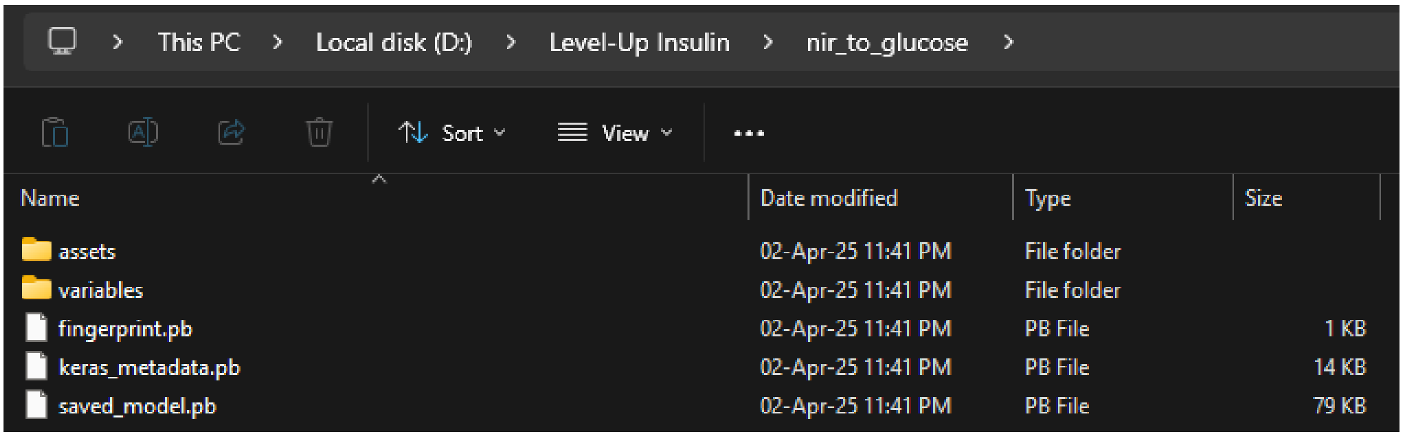
Exporting as TensorFlow Model

MATLAB

```
exportNetworkToTensorFlow(net,"nir_to_glucose")
```

Output:

- **saved_model.pb** → Core architecture/blueprint of the model
- **fingerprint.pb** → Cached hashes of the model
- **keras_metadata.pb** → For Keras compatibility


This PC > Local disk (D:) > Level-Up Insulin > nir_to_glucose >
Sort
View

Name	Date modified	Type	Size
assets	02-Apr-25 11:41 PM	File folder	
variables	02-Apr-25 11:41 PM	File folder	
fingerprint.pb	02-Apr-25 11:41 PM	PB File	1 KB
keras_metadata.pb	02-Apr-25 11:41 PM	PB File	14 KB
saved_model.pb	02-Apr-25 11:41 PM	PB File	79 KB

Converting from TF to TFLite Model

In order to be able to fit into an ESP32, the model must be quantized and converted into a TFLite (TensorFlowLite) model.

Python – VS Code

```python
import tensorflow as tf

# Load SavedModel
saved_model_dir = "nir_to_glucose"
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)

# Optimize for size (Quantization)
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Convert to TFLite
tflite_model = converter.convert()

# Save TFLite Model
with open("nir_to_glucose.tflite", "wb") as f:
    f.write(tflite_model)

print("TFLite model conversion successful! File saved as 'nir_to_glucose.tflite'")
```

TFLite to C header file (".h" file)

To be able to use the TFLite model in the ESP32, we must convert it into a “.h” file. Then, it can be included in our code in Arduino IDE and be uploaded the board.

Python – VS Code

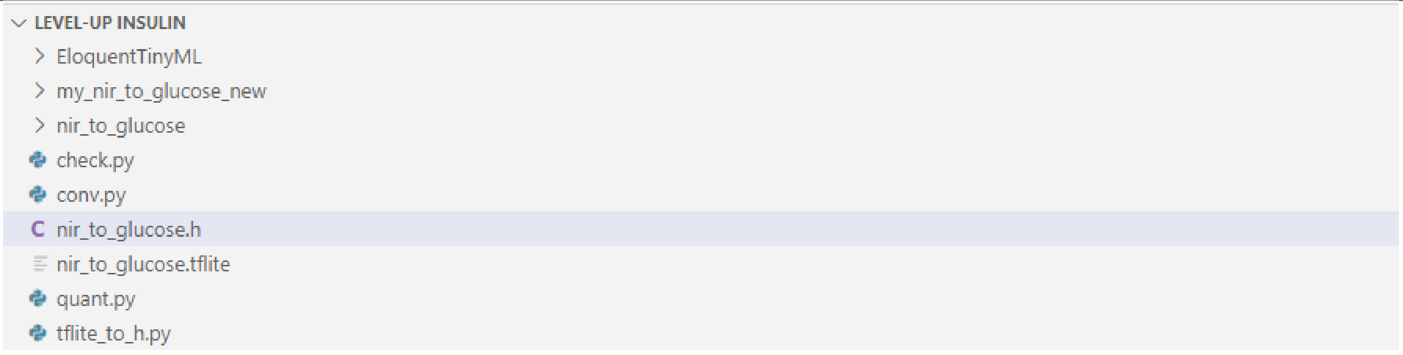
```python
import numpy as np
import os

def convert_tflite_to_header(tflite_path, output_header_path):
    with open(tflite_path, 'rb') as tflite_file:
        tflite_content = tflite_file.read()


    hex_lines = [', '.join([f'0x{byte:02x}' for byte in tflite_content[i:i+12]]) for i in 
range(0, len(tflite_content), 12)]


    hex_array = ',\n  '.join(hex_lines)


    with open(output_header_path, 'w') as header_file:

        header_file.write('const unsigned char model[] = {\n  ')
        header_file.write(f'{hex_array}\n')
        header_file.write('};\n\n')

if __name__ == "__main__":
    tflite_path = 'nir_to_glucose.tflite'
    output_header_path = 'nir_to_glucose.h'

    convert_tflite_to_header(tflite_path, output_header_path)
    convert_tflite_to_header(tflite_path, output_header_path)
```

Output:

We end up with `nir_to_glucose.tflite` and `nir_to_glucose.h` files.

Model size: 20 KB

VS Code



```
LEVEL-UP INSULIN
  > EloquentTinyML
  > my_nir_to_glucose_new
  > nir_to_glucose
  check.py
  conv.py
  nir_to_glucose.h
  nir_to_glucose.tflite
  quant.py
  tflite_to_h.py
```

Uploading to ESP32

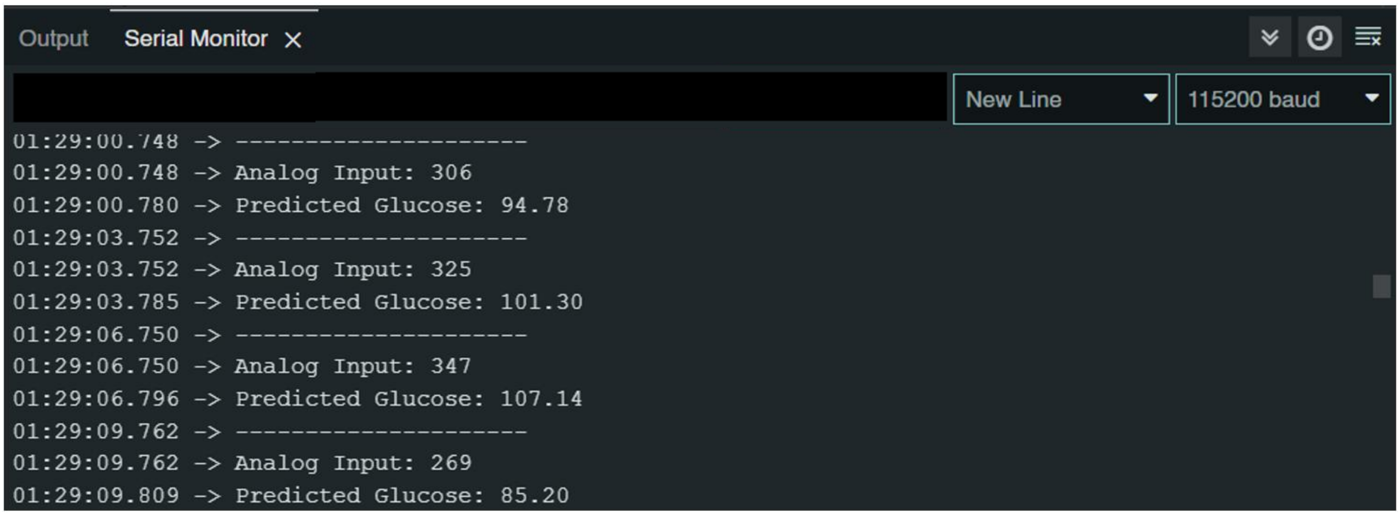
Included Libraries:

- **EloquentTinyML** → This library is to simplify the deployment of Tensorflow Lite for Microcontroller models. It provides an interface to load a model and run inferences.
- **TFLM_ESP32** → Dependency library for EloquentTinyML.
- **nir_to_glucose.h** → Our neural network model.

Arduino IDE

```
#include "nir_to_glucose.h"
#include <tflm_esp32.h>
#include <eloquent_tinyml.h>

#define NUM_OF_INPUTS 1
#define NUM_OF_OUTPUTS 1

#define ARENA_SIZE 2*1024

Eloquent::TinyML::TfLite<NUM_OF_INPUTS, NUM_OF_OUTPUTS, ARENA_SIZE> tf(nir_to_glucose);

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

void loop() {
   float x = random(250, 951);
   float input[1] = { x };
   float predicted = tf.predict(input);
   Serial.println("---------------------");
   Serial.print("Analog Input: ");
   Serial.println(x);
   Serial.print("Predicted Glucose: ");
   Serial.println(predicted);
   delay(3000);
}
```

Serial Monitor:



```
01:29:00.748 -> ---------------------
01:29:00.748 -> Analog Input: 306
01:29:00.780 -> Predicted Glucose: 94.78
01:29:03.752 -> ---------------------
01:29:03.752 -> Analog Input: 325
01:29:03.785 -> Predicted Glucose: 101.30
01:29:06.750 -> ---------------------
01:29:06.750 -> Analog Input: 347
01:29:06.796 -> Predicted Glucose: 107.14
01:29:09.762 -> ---------------------
01:29:09.762 -> Analog Input: 269
01:29:09.809 -> Predicted Glucose: 85.20
```