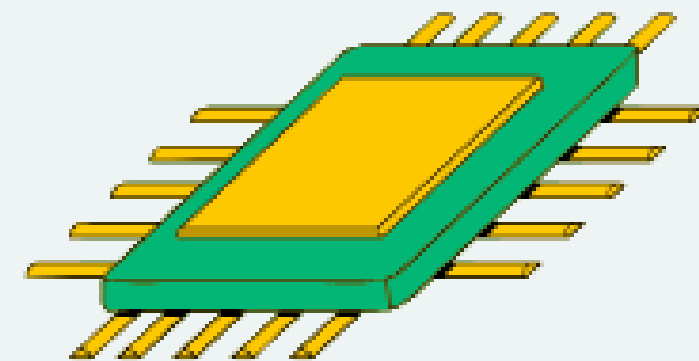


TinyML Risk Indicator on SPDuino

LOGISTIC MODEL → TINY DEPLOYMENT

SARAH RACHEL



Presentation Outline

1. Introduction
2. Motivation & Use-case
3. Demo Overview
4. Hardware: SPDuino (board) & Peripherals
5. TinyML Pipeline (Step-by-Step Execution)
6. On-Device Implementation Details (SPDuino sketch)
7. Role of TensorFlow in Pipeline



Introduction

- **TinyML** = running Machine Learning models on microcontrollers
- Works on devices with very low memory (16–256 KB RAM)
- Enables real-time, offline inference with no internet or cloud
- Models must be lightweight: small number of parameters, minimal compute
- **Examples:** Anomaly detection, Sound classification, Gesture recognition, Health risk indicators

1. Train model on PC (Python / TensorFlow)
2. Compress / extract model parameters
3. Deploy parameters (weights, bias, scaler values) to microcontroller
4. Run inference using basic operations like Activation Functions
5. Output through LEDs, buzzers or Serial monitors

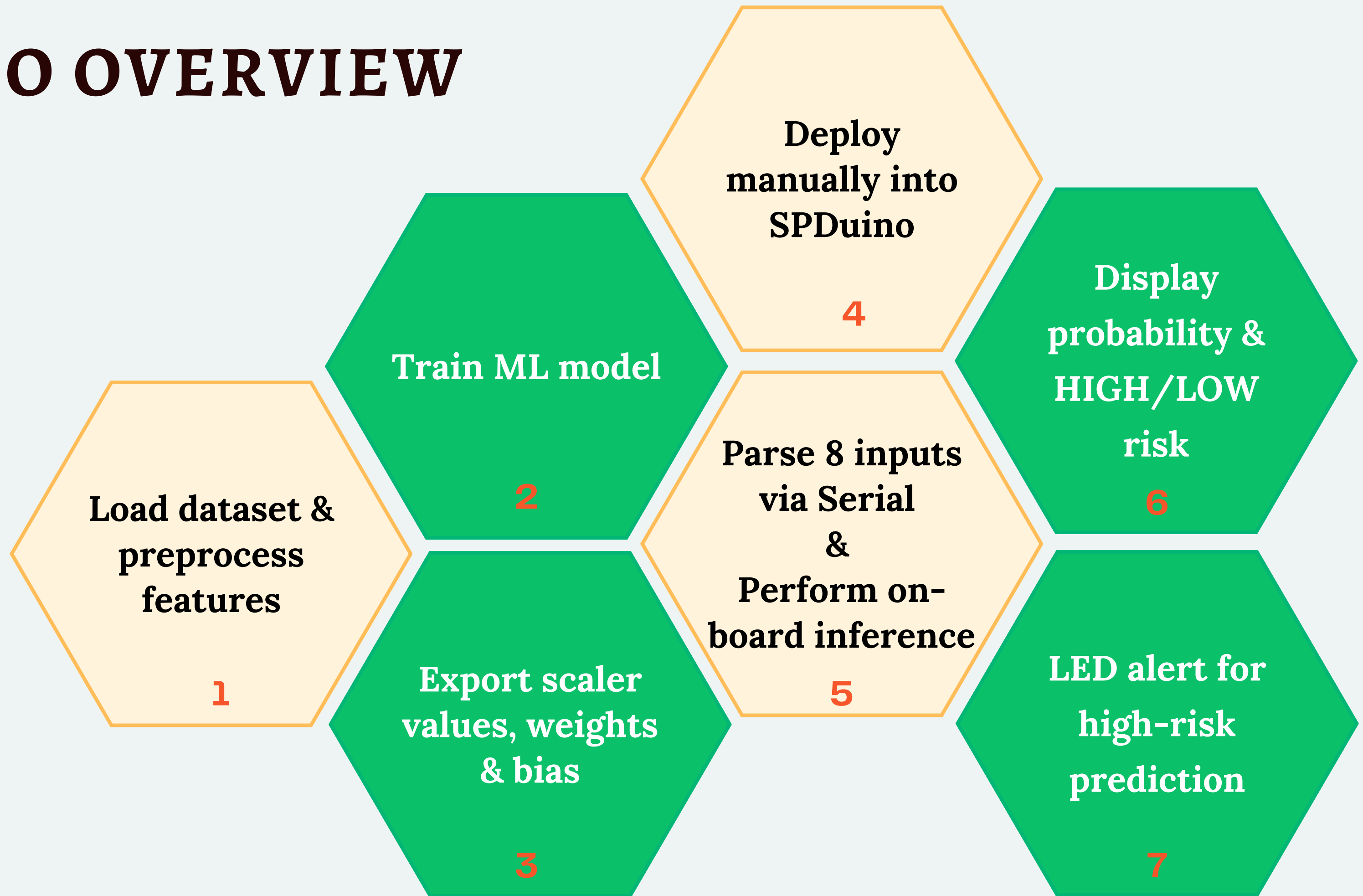
How ML works on microcontrollers??



Motivation & Use-Case

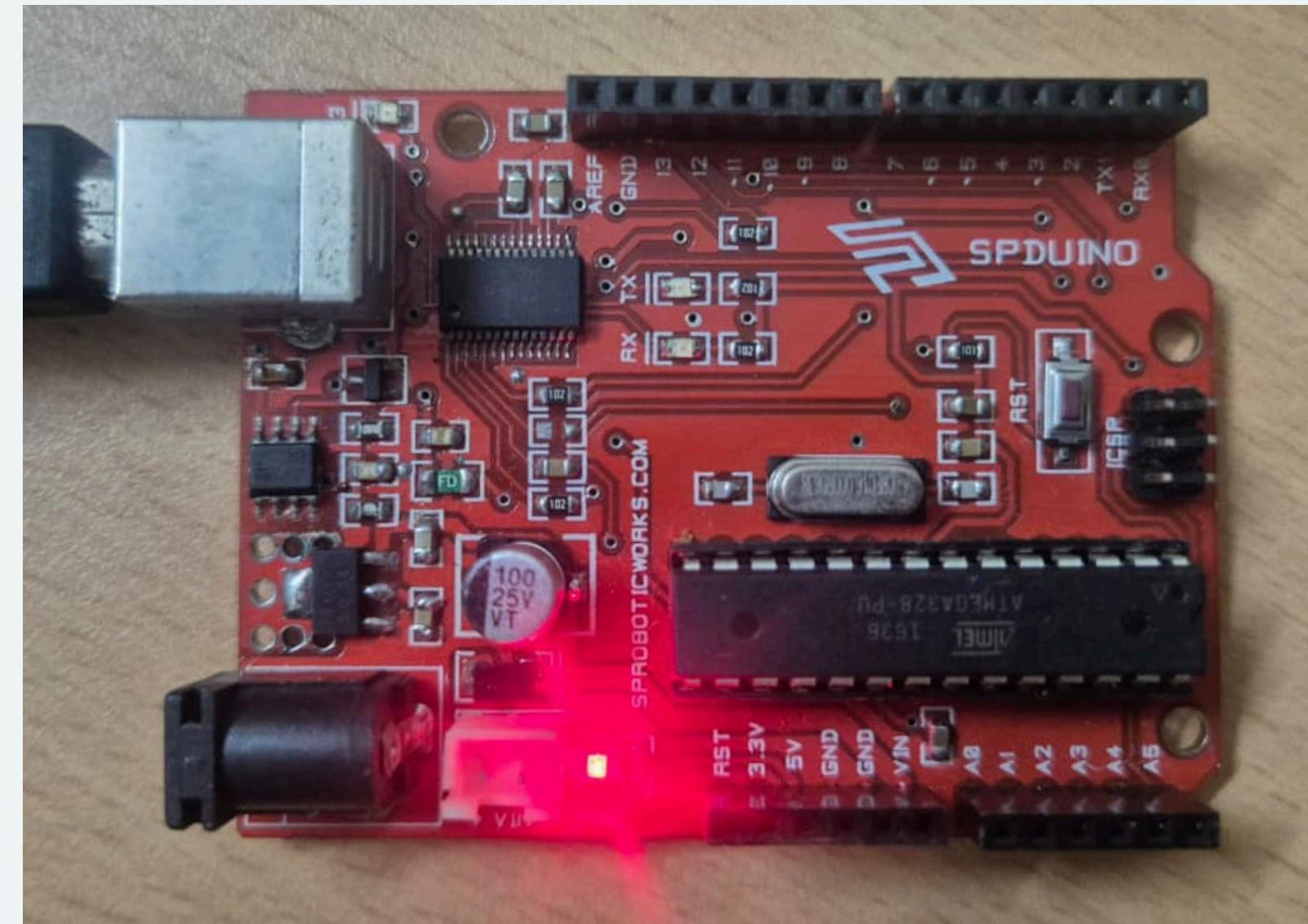
- Many real-world systems need low-cost, offline ML inference
- **Microcontrollers provide:**
 - Low power
 - High reliability
 - No cloud dependency
 - Fast response
- **Why this project?**
 - Demonstrate full TinyML workflow end-to-end
 - Build a real-time diabetes risk indicator device
 - Show how a standard ML model can run on an SPDuino board
 - Use dataset parameters (e.g., glucose, BMI, BP) to classify High / Low risk

DEMO OVERVIEW



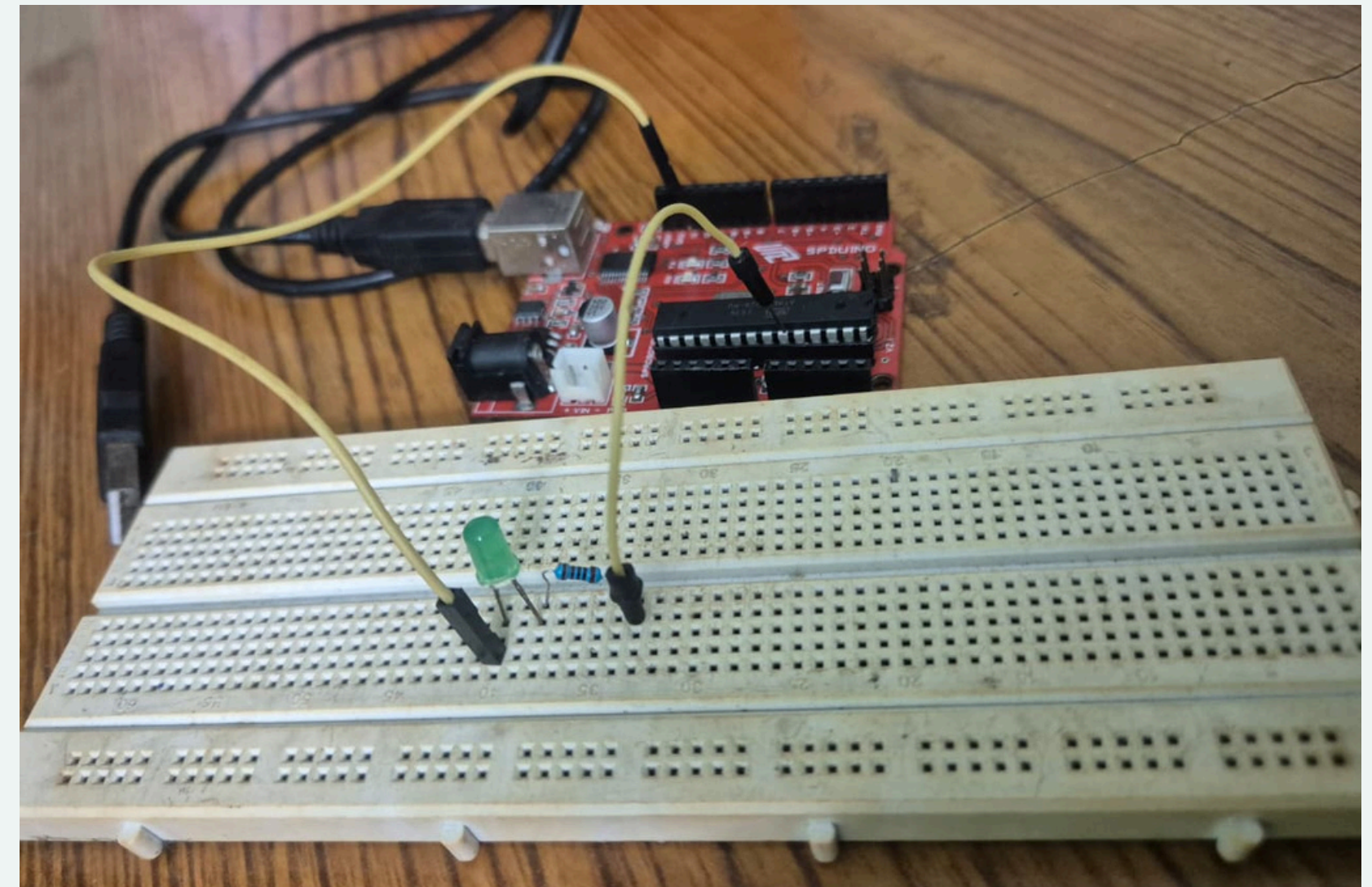
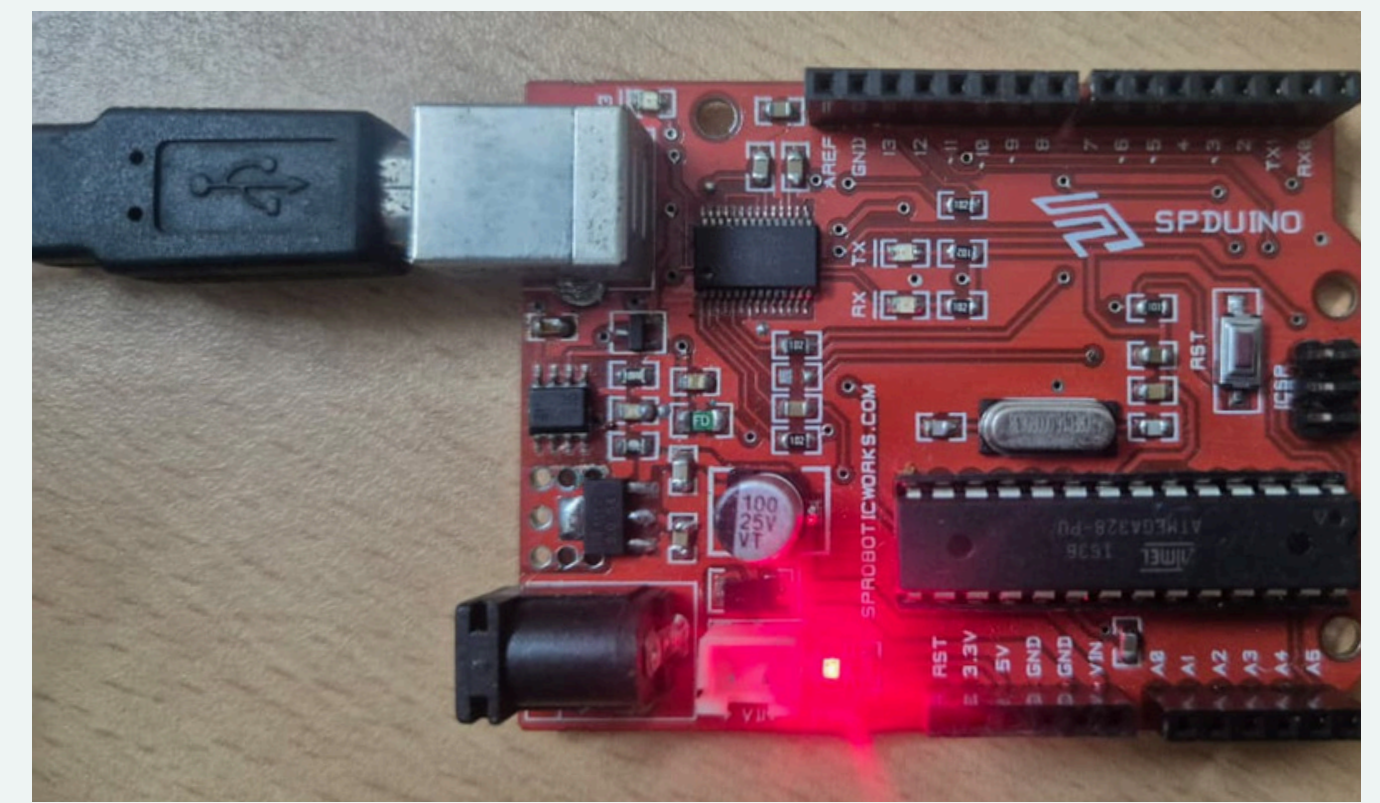
Hardware: SPDuino Board

- The **SPDuino** board is designed to work like a standard Arduino (UNO/Nano style)
- It uses an 8-bit AVR microcontroller (ATmega328P-class) so:
 - Easy to program in Arduino IDE.
 - Supports C/C++ based Arduino sketches.
 - Works on USB or small battery
 - Well-supported libraries for sensors & timers.
- Suitable for low-power ML inference such as:
 - Quantized (8-bit)
 - Optimized with TFLite Micro
 - Small footprint (<30 KB model size)
- Supports standard Serial Monitor (9600 baud)
- Sufficient flash 32Kb & SRAM 2 Kb for lightweight 1D features (MFCC).



Hardware: Peripherals

- LED + 220Ω resistor
 - HIGH risk → LED ON
 - LOW risk → LED OFF
- USB cable for power + uploading sketch
- Breadboard
- Serial Monitor for entering 8 input values
- Pin Connections:
 - LED anode → Digital Pin (e.g., D7)
 - LED cathode → resistor → GND



TINYML PIPELINE

Pima Indians Diabetes Database

Predict the onset of diabetes based on diagnostic measures

[Data Card](#) [Code \(3803\)](#) [Discussion \(54\)](#) [Suggestions \(0\)](#)

About Dataset

Context

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

1) Data Preparation

- Load dataset
- Convert to csv/excel
- Clean & scale features.


diabetes.csv > data

```
1 Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigreeFunction,Age,Outcome
2 6,148,72,35,0,33.6,0.627,50,1
3 1,85,66,29,0,26.6,0.351,31,0
4 8,183,64,0,0,23.3,0.672,32,1
5 1,89,66,23,94,28.1,0.167,21,0
6 0,137,40,35,168,43.1,2.288,33,1
7 5,116,74,0,0,25.6,0.201,30,0
8 3,78,50,32,88,31,0.248,26,1
9 10,115,0,0,0,35.3,0.134,29,0
10 2,197,70,45,543,30.5,0.158,53,1
```

TINYML PIPELINE

2) Model Prep

- Train model
- (Logistic Regression / TensorFlow)
- Export: weights, bias, scaler mean, scaler scale.

 train_diabetes.py > ...

```
1 import pandas as pd
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.model_selection import train_test_split
4 from sklearn.preprocessing import StandardScaler
5
6 df = pd.read_csv("diabetes.csv")
7 X = df.drop("Outcome", axis=1)
8 y = df["Outcome"]
9
10 # Scale data
11 scaler = StandardScaler()
12 X_scaled = scaler.fit_transform(X)
13
14 # Train-test split
15 X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
16
17 # Train Logistic Regression
18 model = LogisticRegression(max_iter=500)
19 model.fit(X_train, y_train)
20
21 # Print model parameters
22 print("Weights:", model.coef_[0])
23 print("Bias:", model.intercept_[0])
24 print("Scaler Mean:", scaler.mean_)
25 print("Scaler Scale:", scaler.scale_)
```

TINYML PIPELINE

3) Model Conversion / Optimization

- Keep model lightweight for microcontrollers
- TensorFlow Lite conversion for the Logistic Regression
- Continue with:
 - Quantization (int8)
 - Feature reduction.

```
58 # 5. TensorFlow Model (Convert LR structure) for TFLite
59 # =====
60 import tensorflow as tf
61
62 # Simple logistic regression in TF
63 tf_model = tf.keras.Sequential([
64     tf.keras.layers.Input(shape=(8,)),
65     tf.keras.layers.Dense(1, activation="sigmoid") # logistic regression
66 ])
67
68 tf_model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
69 # Train on the same scaled data
70 tf_model.fit(X_train, y_train, epochs=50, batch_size=16, verbose=0)
71 loss, acc = tf_model.evaluate(X_test, y_test, verbose=0)
72 print("TensorFlow Model Accuracy:", acc)
73 converter = tf.lite.TFLiteConverter.from_keras_model(tf_model)
74 tflite_model = converter.convert()
75
76 # Save .tflite file
77 with open("diabetes.tflite", "wb") as f:
78     f.write(tflite_model)
79
80 print("TFLite model saved as diabetes.tflite")
```

Role of TensorFlow in the Pipeline

Model Building & Training

- Create lightweight ML models (Dense layer, small NN, logistic equivalent)
- Supports large datasets with efficient training
- Easy integration with preprocessing steps

Model Optimization

- Convert trained model to TensorFlow Lite (TFLite)
- Apply quantization (int8) to reduce size
- Enables deployment on microcontrollers with limited RAM/Flash

Dataset Reduction

- Use TensorFlow data pipelines for Feature selection, Normalization, Dimensionality reduction.
- Ensures efficient preprocessing before deployment

Deployment to TinyML Devices

- TFLite model → converted to TensorFlow Lite Micro format
- Runs directly on microcontrollers like SPDuino

Ensures Consistent Predictions

- Same TF model used in Python → converted → runs identically on device

TINYML PIPELINE

3) Model Conversion / Optimization

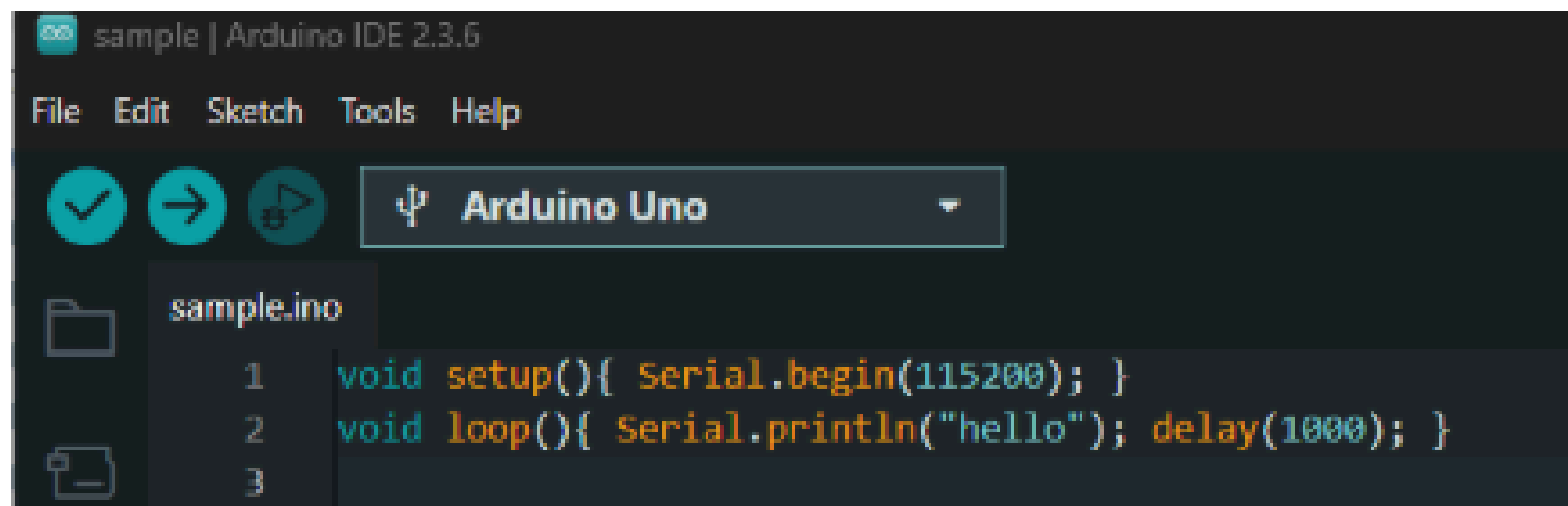
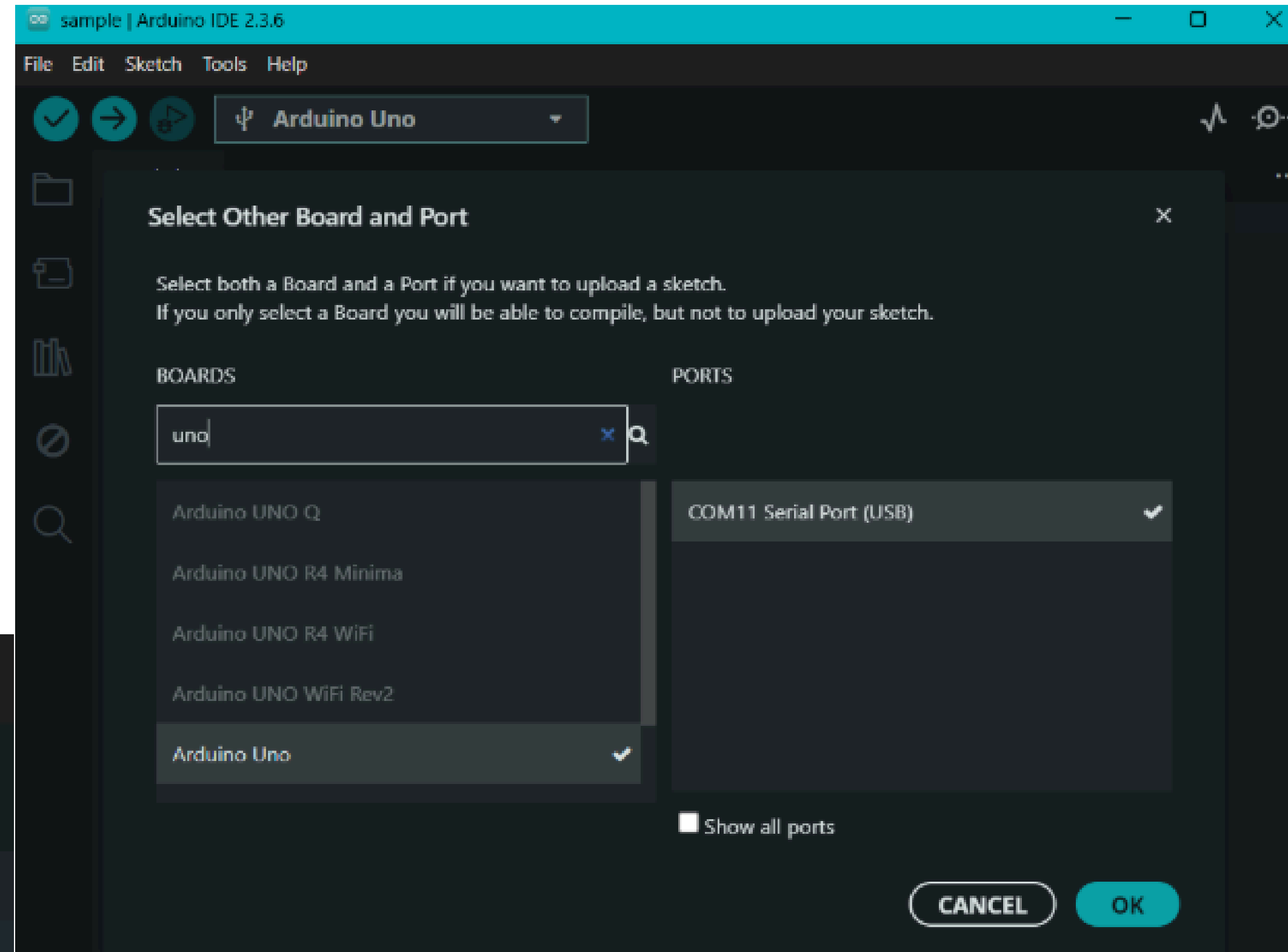
- Keep model lightweight for microcontrollers
- TensorFlow Lite conversion for the Logistic Regression
- Continue with:
 - Quantization (int8)
 - Feature reduction.

```
82 # =====
83 # 7. INT8 Quantization (recommended for microcontrollers)
84 # =====
85 converter = tf.lite.TFLiteConverter.from_keras_model(tf_model)
86 # Set quantization parameters
87 converter.optimizations = [tf.lite.Optimize.DEFAULT]
88 # Representative dataset for quantization
89 def representative_data_gen():
90     for i in range(100):
91         yield [X_train[i:i+1].astype(np.float32)]
92 converter.representative_dataset = representative_data_gen
93 converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
94 converter.inference_input_type = tf.int8
95 converter.inference_output_type = tf.int8
96 tflite_quant_model = converter.convert()
97
98 # Save INT8 quantized model
99 with open("diabetes_int8.tflite", "wb") as f:
100     f.write(tflite_quant_model)
101
102 print("Quantized TFLite model saved as diabetes_int8.tflite")
103 |
```

TINYML PIPELINE

4) Setup SPDuino

- Setup the Ardunio IDE
- SPDuino board and Peripherals
- Connect it to the System
- Create the Sketch



Output

Sketch uses 1628 bytes (5%) of program storage space. Maximum is 32256 bytes.
Global variables use 194 bytes (9%) of dynamic memory, leaving 1854 bytes for local variables.

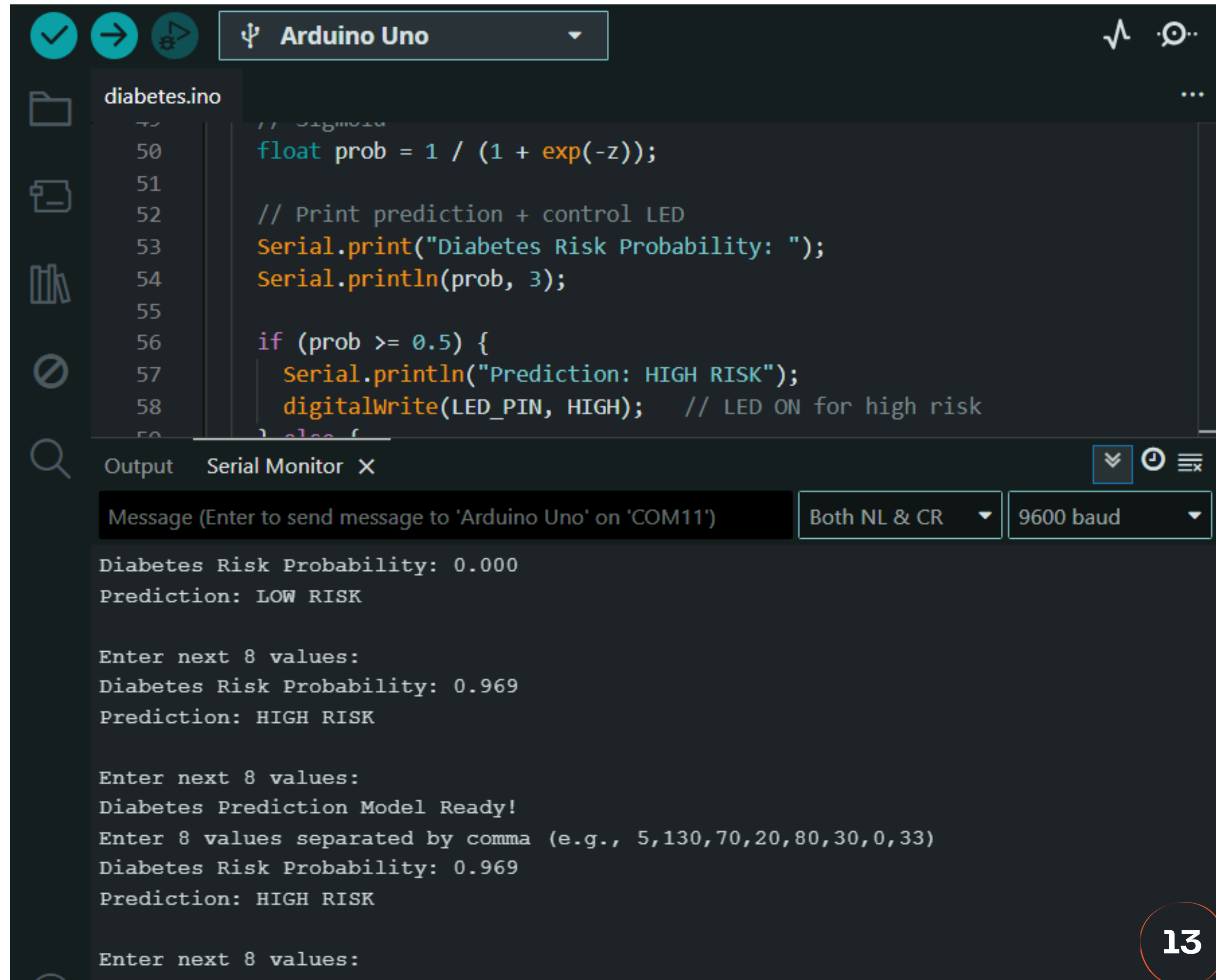
TINYML PIPELINE

5) Deployment on SPDuino

- Implement:
 - Dot product + bias
 - Sigmoid activation
 - Compile & upload to board

Eg High risks:

- 8,180,90,35,200,40,1.5,60
- 10,200,95,45,250,45,2.0,55
- 6,170,88,30,180,38,1.8,50



```
diabetes.ino
// Sigmoid
50 float prob = 1 / (1 + exp(-z));
51
52 // Print prediction + control LED
53 Serial.print("Diabetes Risk Probability: ");
54 Serial.println(prob, 3);
55
56 if (prob >= 0.5) {
57     Serial.println("Prediction: HIGH RISK");
58     digitalWrite(LED_PIN, HIGH); // LED ON for high risk
59 } else {
60     digitalWrite(LED_PIN, LOW); // LED OFF for low risk
61 }
```

Output Serial Monitor X

Message (Enter to send message to 'Arduino Uno' on 'COM11') Both NL & CR 9600 baud

Diabetes Risk Probability: 0.000
Prediction: LOW RISK

Enter next 8 values:
Diabetes Risk Probability: 0.969
Prediction: HIGH RISK

Enter next 8 values:
Diabetes Prediction Model Ready!
Enter 8 values separated by comma (e.g., 5,130,70,20,80,30,0,33)
Diabetes Risk Probability: 0.969
Prediction: HIGH RISK

Enter next 8 values:

TINYML PIPELINE

6) Testing on SPDuino

- Frame test cases in form of vectors so that they can be given to serial monitor input.

Eg High risks:

- 8,180,90,35,200,40,1.5,60
- 10,200,95,45,250,45,2.0,55
- 6,170,88,30,180,38,1.8,50

Eg of Low risk:

```
# ----- Test a single sample -----  
# Replace these with one row from diabetes.csv (8 feature values)  
v1 = 5  
v2 = 130  
v3 = 70  
v4 = 20  
v5 = 80  
v6 = 30  
v7 = 0.0  
v8 = 33  
x = np.array([[v1, v2, v3, v4, v5, v6, v7, v8]])  
x_scaled = scaler.transform(x)  
prob = model.predict_proba(x_scaled)[0, 1]  
pred = model.predict(x_scaled)[0]  
print("Python prob:", prob)  
print("Python pred:", pred)  
  
Python prob: 0.264289221072794  
Python pred: 0
```

TINYML PIPELINE

6) Testing on SPDuino

- Reset the board each time and try various test cases
- Decision rule for the activation function Sigmoid:
 - $\text{prob} \geq 0.5$
 - $\text{Prob} \geq 0.5$

```
diabetes.ino
19 }
20
21 void loop() {
22     if (Serial.available() > 0) {
23         String input = Serial.readStringUntil('\n');
24         float x[FEATURES];
25         int start = 0, end = 0;
26         for (int i = 0; i < FEATURES; i++) {
27             end = input.indexOf(',', start);
28             if (end == -1) end = input.length();
29             x[i] = input.substring(start, end).toFloat();
30             start = end + 1;
31         }
32
33         // Scale input
34         float z = 0;
35         for (int i = 0; i < FEATURES; i++) {
36             float x_scaled = (x[i] - mean[i]) / scale[i];
37             z += x_scaled * weights[i];
38         }
39     }
40 }
```

Output Serial Monitor X

Message (Enter to send message to 'Arduino Uno' on 'COM11') Both NL & CR 9600 baud

Enter 8 values separated by comma (e.g., 5,130,70,20,80,30,0,33)

Diabetes Risk Probability: 0.000

Prediction: LOW RISK

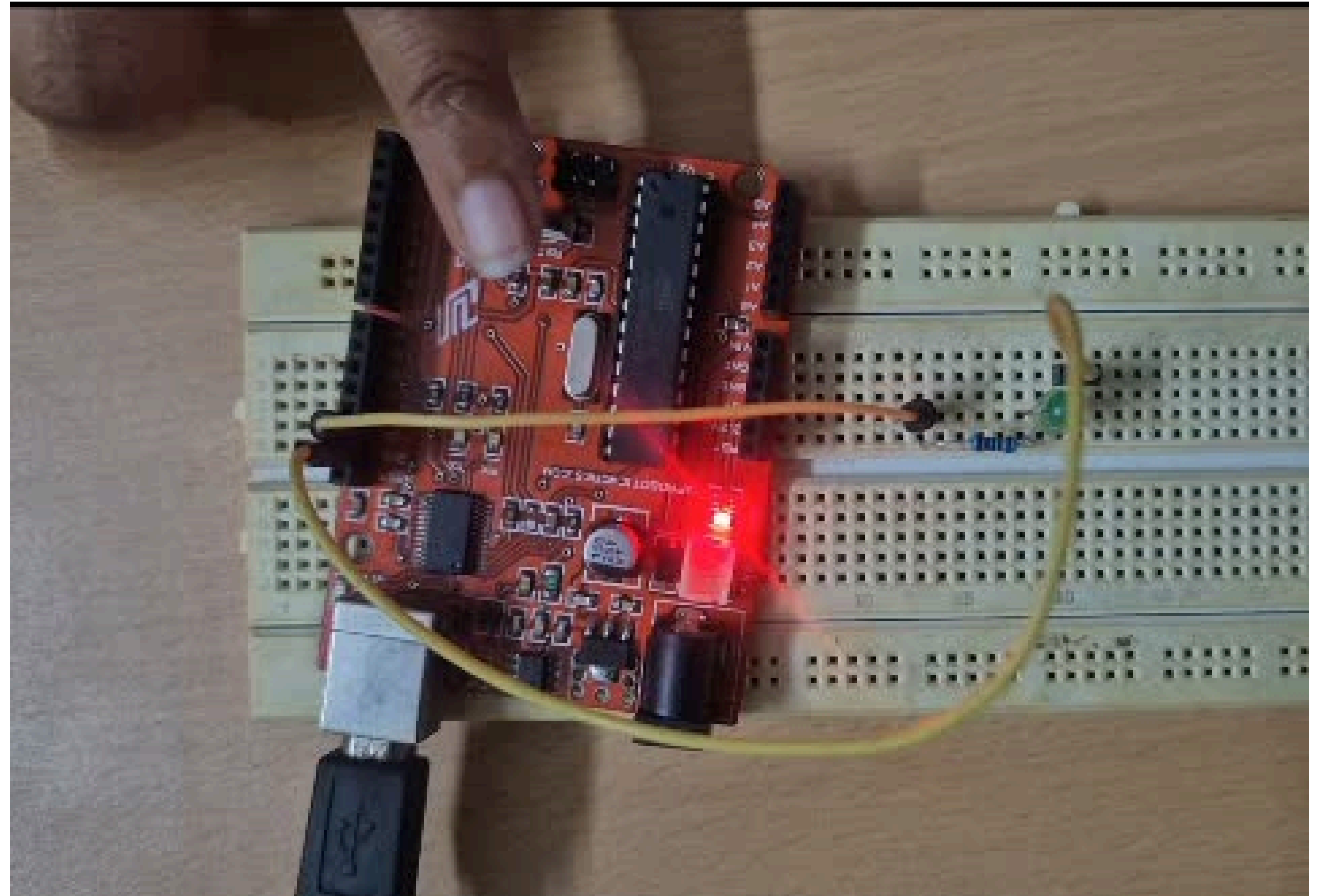
Enter next 8 values:

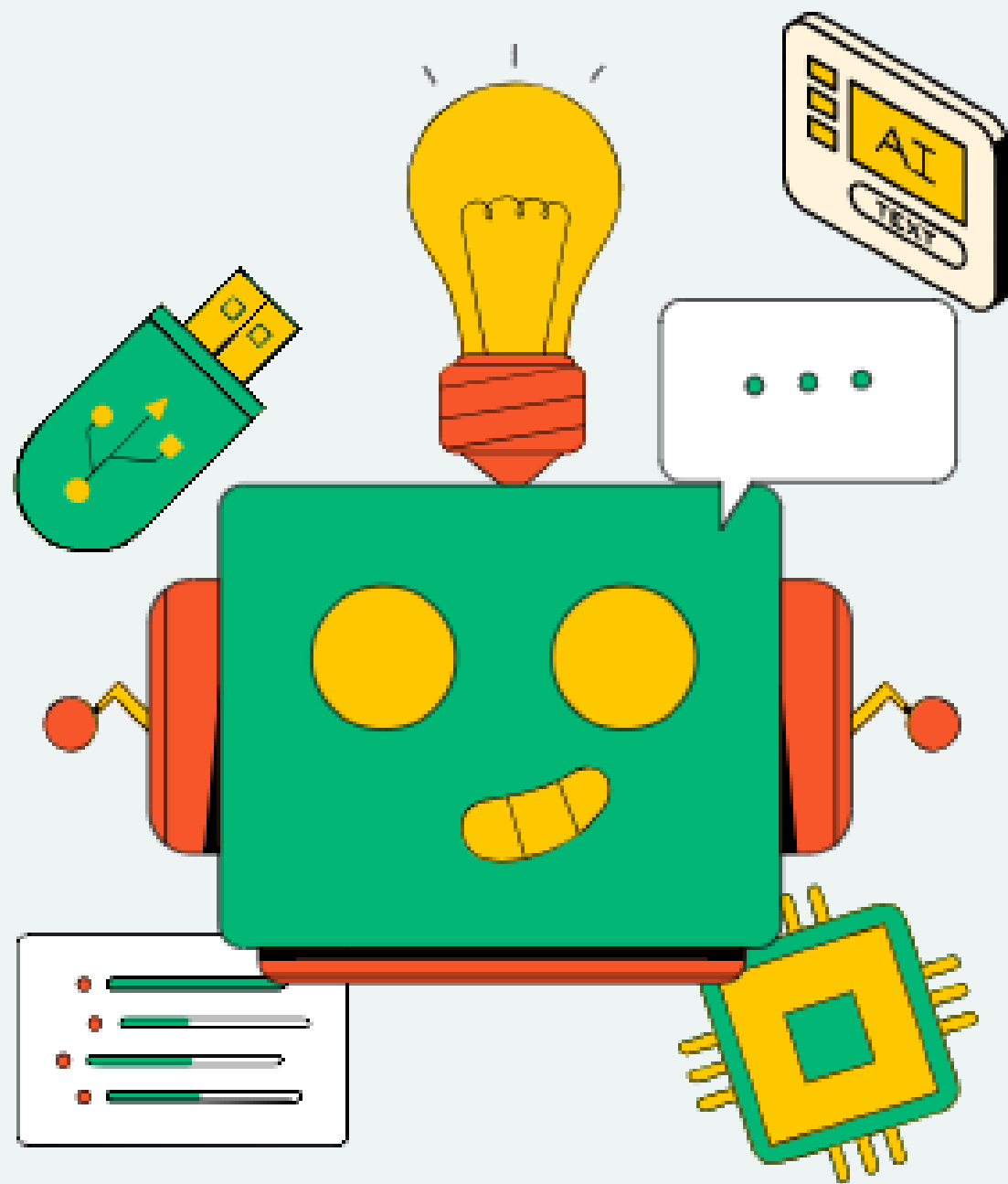
TINYML PIPELINE

7) Alerting through LED

- Read inputs
- Run model locally
- Output result (Serial + LED indicator)

If
 $\text{prob} \geq 0.5 \rightarrow \text{HIGH RISK (LED ON)}$
Else
 $\text{prob} < 0.5 \rightarrow \text{LOW RISK (LED OFF)}$





THANK YOU

