🧪 Projeto Final: Classificação de vibração no liquidificador com TinyML

Alunos:

Alyson Henrique Gonçalo (2019004779)

Davi José Moreira Cunha (2021016131)

Gean Carlos Gonçalves Martins (2020021262)

Matias da Silva Campos (2021025776)

Instituição: Universidade Federal de Itajubá
Disciplina: TinyML
Data: 01/07/2025
Projeto no Edge Impulse: https://studio.edgeimpulse.com/public/732936/live

---

📝 Objetivo

Este projeto tem como objetivo aplicar técnicas de aprendizado de máquina em dispositivos embarcados de baixo consumo energético para identificar diferentes padrões de vibração de um liquidificador doméstico. A proposta é capturar os dados com sensores acelerômetros, treinar um modelo leve (TinyML) para classificar os estados operacionais do liquidificador (por exemplo, vazio, com água em velocidade baixa, com água em velocidade alta) e embarcar esse modelo em um microcontrolador como o Arduino Nano 33 BLE Sense.

🔧 Materiais Utilizados

- Arduino Nano 33 BLE Sense
- Cabo micro USB
- Fita hellerman
- Notebook com Arduino IDE
- Plataforma Edge Impulse

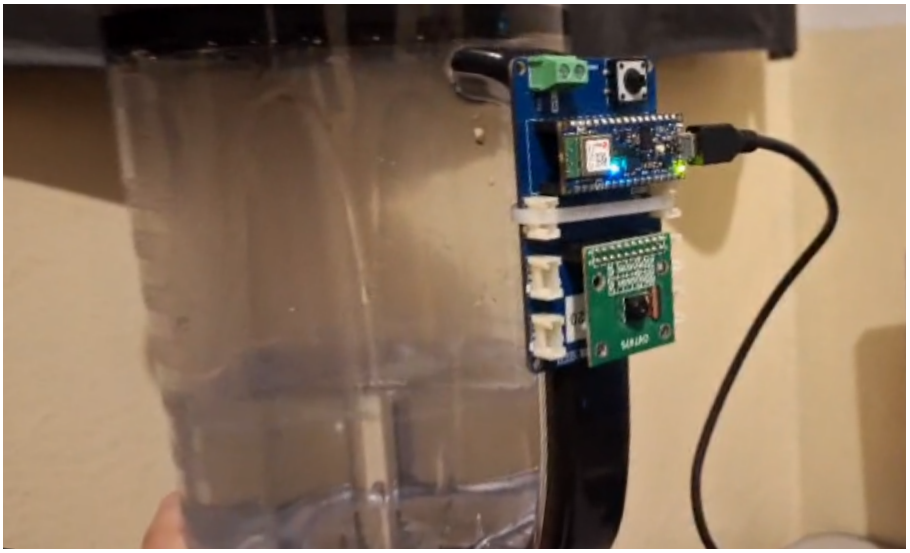
- Liquidificador philco com 1 litro de água

⚙️ Etapas Realizadas

1. Montagem e Conexão

O Arduino foi fixado ao corpo do liquidificador com fita isolante para registrar com precisão as vibrações geradas em diferentes modos de operação.

📸 **Imagem 1 – Arduino fixado no liquidificador:**



---

2. Criação do Projeto no Edge Impulse

Foi criado o projeto chamado `projeto-final` na plataforma Edge Impulse. O dispositivo foi conectado ao projeto via terminal com o comando `edge-impulse-daemon`.

As classes utilizadas na classificação foram:

- `off` – liquidificador desligado
- `liquidificador_com_agua_leve`– ligado em baixa velocidade
- `liquidificador_com_agua_pesado` – ligado em alta velocidade

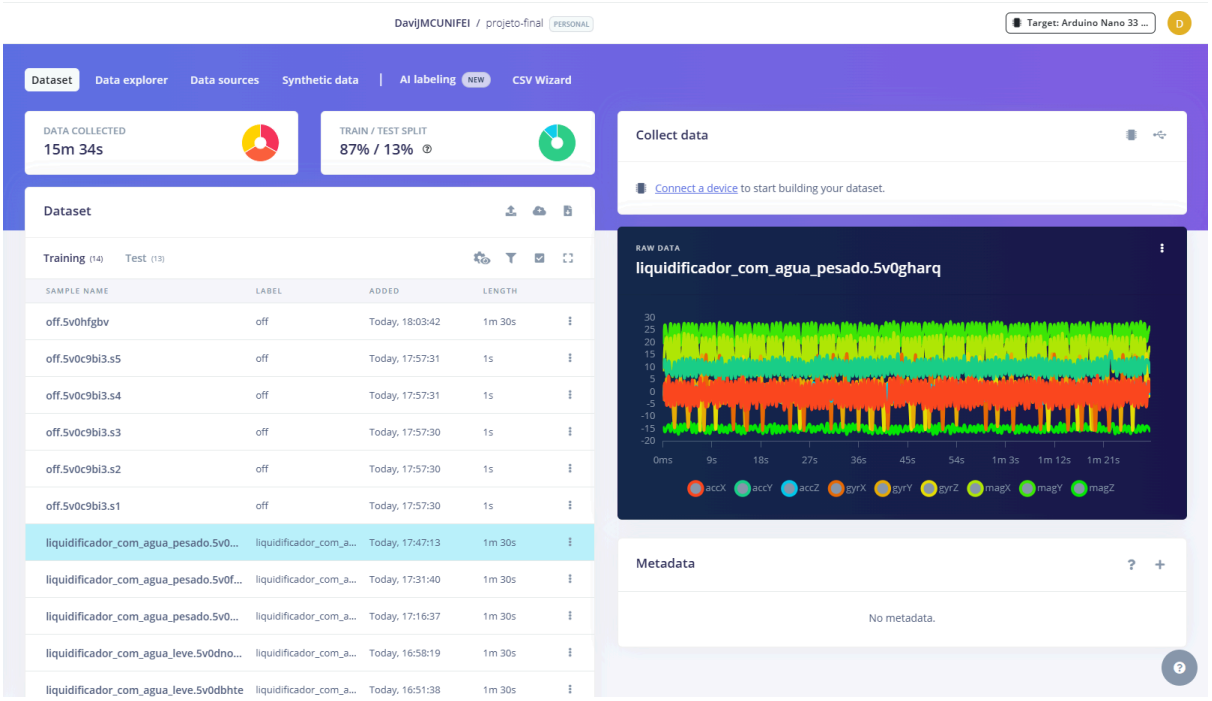
---

3. Coleta de Dados

Os dados foram coletados usando o acelerômetro interno do Arduino. Para cada classe, foram registrados:

- Cerca de **5 minutos de dados** para treinamento
- Cerca de **30 segundos de dados** para teste

📸 **Imagem 2 – Tela de aquisição de dados no Edge Impulse:**



---

4. Extração de Características (Feature Extraction)

Foi utilizado o bloco de processamento **Spectral Analysis** para gerar os vetores de características. Os dados foram convertidos em espectros de frequência, e o gráfico do **Feature Explorer** foi utilizado para verificar a separação entre as classes.

📸 **Imagem 3 – Feature Explorer com as classes:**

---

5. Treinamento do Modelo

Foi utilizado o bloco de **Neural Network Classifier**, sem detecção de anomalia. O modelo foi treinado com os dados das três classes e apresentou boa separação no espaço de características.

📸 **Imagem 4 – Matriz de confusão e data explorer após o treinamento:**

Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY
100.0%

LOSS
0.00

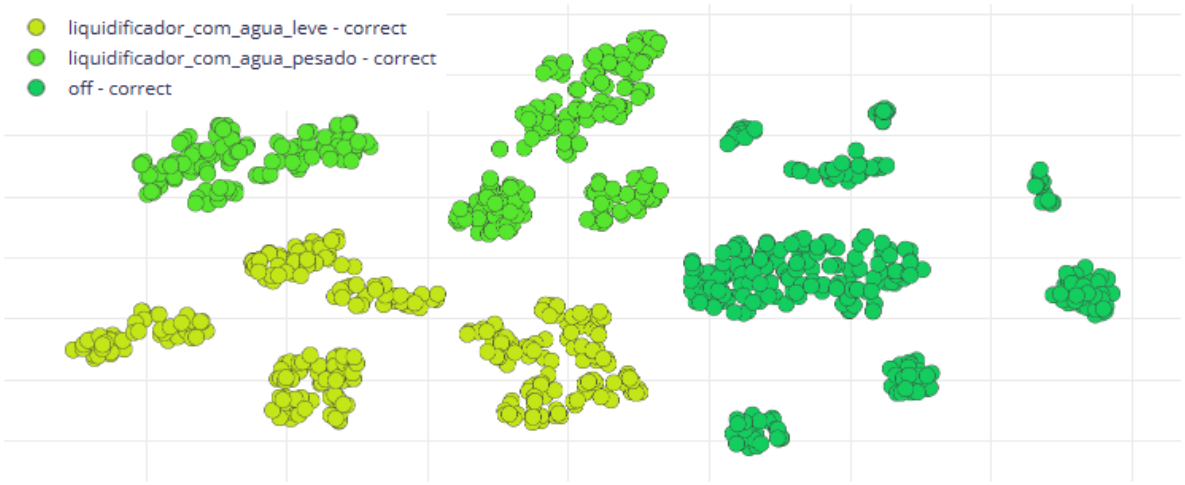
Confusion matrix (validation set)

	LIQUIDIFICADOR_COM_AGUA	LIQUIDIFICADOR_COM_AGUA	OFF
LIQUIDIFICADOR_COM_AGUA	100%	0%	0%
LIQUIDIFICADOR_COM_AGUA	0%	100%	0%
OFF	0%	0%	100%
F1 SCORE	1.00	1.00	1.00

Metrics (validation set)

METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	1.00
Weighted average Recall	1.00
Weighted average F1 score	1.00

Data explorer (full training set)



On-device performance

Engine: EON™ Compiler

INFERENCING TIME
1 ms.

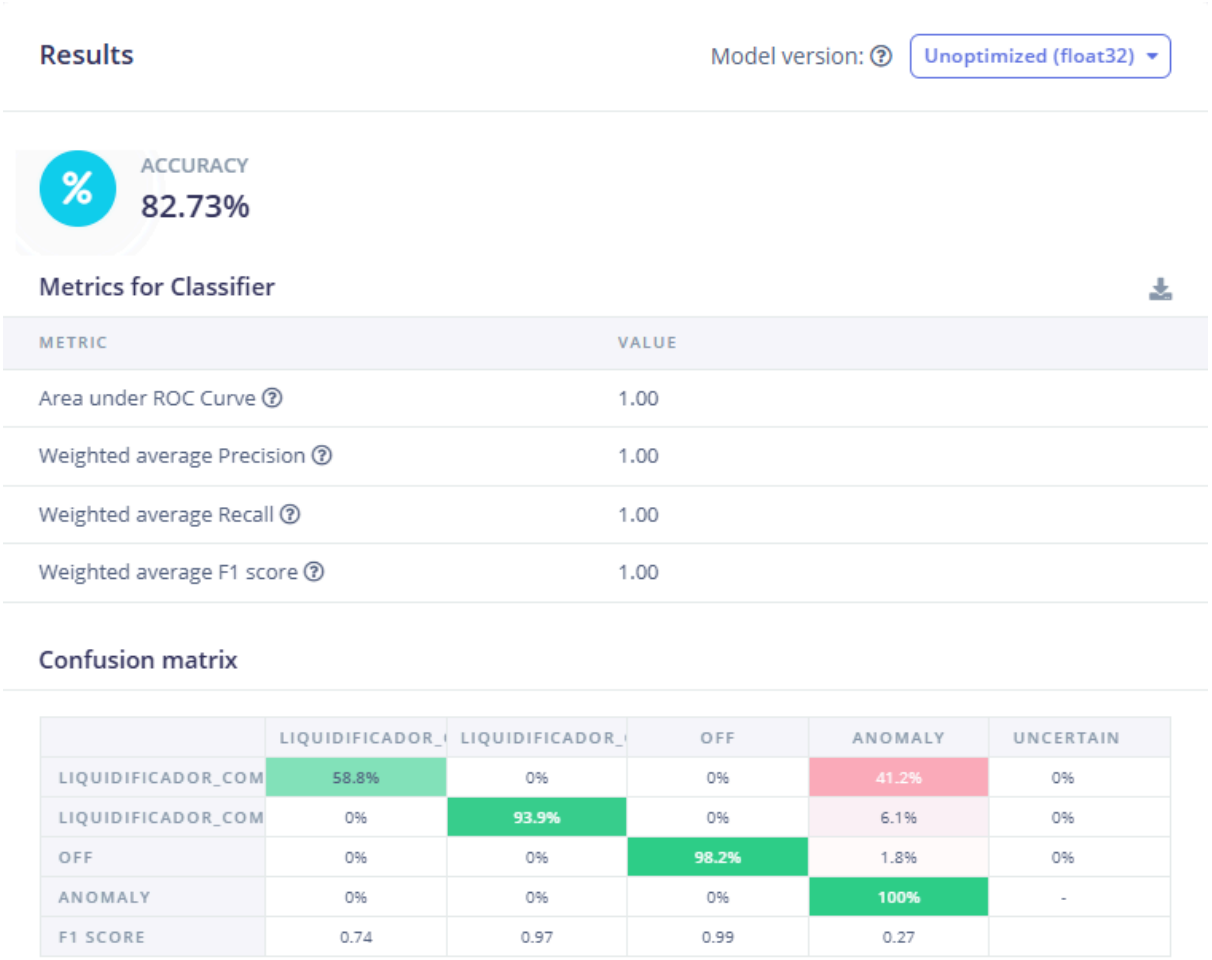
PEAK RAM USAGE
1.5K

FLASH USAGE
16.5K

6. Teste do Modelo

Após o treinamento, o modelo foi avaliado com os dados de teste. A acurácia final ficou acima de 90%, indicando que o modelo aprendeu corretamente os padrões de vibração associados a cada classe.

📸 **Imagem 5 – Resultados da classificação no teste:**



Results

Model version: Unoptimized (float32)

ACCURACY
82.73%

Metrics for Classifier

METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	1.00
Weighted average Recall	1.00
Weighted average F1 score	1.00

Confusion matrix

	LIQUIDIFICADOR_	LIQUIDIFICADOR_	OFF	ANOMALY	UNCERTAIN
LIQUIDIFICADOR_COM	58.8%	0%	0%	41.2%	0%
LIQUIDIFICADOR_COM	0%	93.9%	0%	6.1%	0%
OFF	0%	0%	98.2%	1.8%	0%
ANOMALY	0%	0%	0%	100%	-
F1 SCORE	0.74	0.97	0.99	0.27	

---

7. Implantação

O modelo foi exportado como biblioteca `.zip` para Arduino e importado no Arduino IDE. O código gerado foi modificado para acionar os **LEDs internos do Arduino** de acordo com a classe detectada:

Lógica dos LEDs:

- 🔴🟢 `liquidificador_com_agua_pesado` → LED vermelho e verde

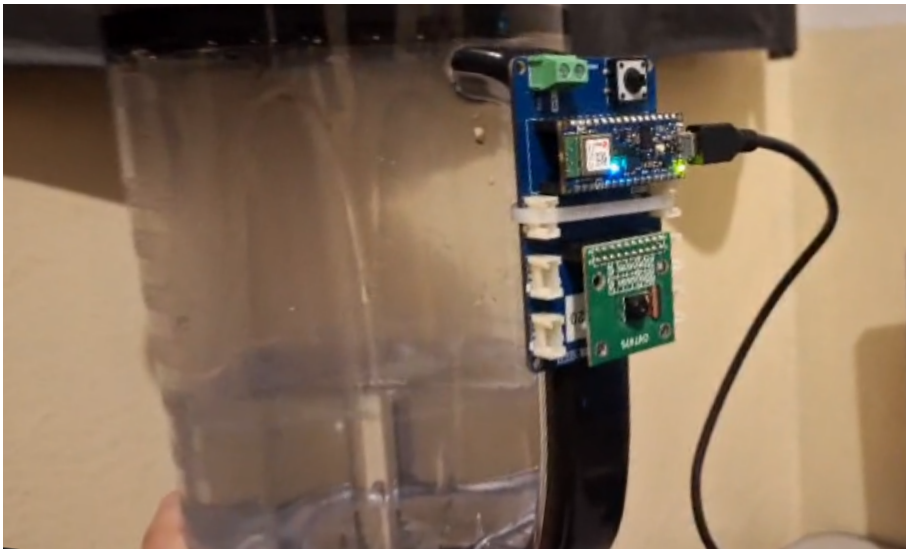
- 🔴🔵 `liquidificador_com_agua_leve` → LED vermelho e azul
- 🔵🟢 `off` → LED azul e verde

8. Vídeo do funcionamento:

🎬 **apresentacaoTinyMl.mp4**

---

9. Fotos



10. Código

```
#include <projeto-final_inferencing.h>
#include <Arduino_LSM9DS1.h>
#include <Arduino.h>

#define RED_LED    LEDR
#define GREEN_LED  LEDG
#define BLUE_LED   LEDB

#define CONVERT_G_TO_MS2    9.80665f
#define MAX_ACCEPTED_RANGE  2.0f
#define ANOMALY_THRESHOLD   0.5f

static bool debug_nn = false;

void setup() {
    Serial.begin(115200);
    while (!Serial);
    Serial.println("Edge Impulse Inferencing Demo");

    if (!IMU.begin()) {
        ei_printf("Failed to initialize IMU!\r\n");
    } else {
        ei_printf("IMU initialized\r\n");
    }

    if (EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME != 9) {
        ei_printf("ERR: EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME should be
equal to 9 (acc, gyro, mag)\n");
        return;
    }

    pinMode(RED_LED, OUTPUT);
    pinMode(GREEN_LED, OUTPUT);
    pinMode(BLUE_LED, OUTPUT);
}

void setLedColor(String classe, float anomaly_score) {
    if (anomaly_score > ANOMALY_THRESHOLD) {
```

```
        // Amarelo (vermelho + verde)
        digitalWrite(RED_LED, HIGH);
        digitalWrite(GREEN_LED, HIGH);
        digitalWrite(BLUE_LED, LOW);
    }
    else if (classe == "off") {
        digitalWrite(RED_LED, HIGH);
        digitalWrite(GREEN_LED, LOW);
        digitalWrite(BLUE_LED, LOW);
    }
    else if (classe == "liquidificador_com_agua_leve") {
        digitalWrite(RED_LED, LOW);
        digitalWrite(GREEN_LED, HIGH);
        digitalWrite(BLUE_LED, LOW);
    }
    else if (classe == "liquidificador_com_agua_pesado") {
        digitalWrite(RED_LED, LOW);
        digitalWrite(GREEN_LED, LOW);
        digitalWrite(BLUE_LED, HIGH);
    }
    else {
        digitalWrite(RED_LED, LOW);
        digitalWrite(GREEN_LED, LOW);
        digitalWrite(BLUE_LED, LOW);
    }
}

void loop() {
    ei_printf("\nStarting inferencing in 2 seconds...\n");
    delay(2000);

    ei_printf("Sampling...\n");

    float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };

    for (size_t ix = 0; ix < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; ix +=
9) {
        uint64_t next_tick = micros() + (EI_CLASSIFIER_INTERVAL_MS *
1000);

        float accX, accY, accZ;
        float gyrX, gyrY, gyrZ;
        float magX, magY, magZ;
```

```
        if (IMU.accelerationAvailable()) IMU.readAcceleration(accX,
accY, accZ);
        if (IMU.gyroscopeAvailable()) IMU.readGyroscope(gyrX, gyrY,
gyrZ);
        if (IMU.magneticFieldAvailable()) IMU.readMagneticField(magX,
magY, magZ);

        accX = constrain(accX, -MAX_ACCEPTED_RANGE,
MAX_ACCEPTED_RANGE);
        accY = constrain(accY, -MAX_ACCEPTED_RANGE,
MAX_ACCEPTED_RANGE);
        accZ = constrain(accZ, -MAX_ACCEPTED_RANGE,
MAX_ACCEPTED_RANGE);

        buffer[ix + 0] = accX * CONVERT_G_TO_MS2;
        buffer[ix + 1] = accY * CONVERT_G_TO_MS2;
        buffer[ix + 2] = accZ * CONVERT_G_TO_MS2;

        buffer[ix + 3] = gyrX;
        buffer[ix + 4] = gyrY;
        buffer[ix + 5] = gyrZ;

        buffer[ix + 6] = magX;
        buffer[ix + 7] = magY;
        buffer[ix + 8] = magZ;

        delayMicroseconds(next_tick - micros());
    }

    signal_t signal;
    int err = numpy::signal_from_buffer(buffer,
EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
    if (err != 0) {
        ei_printf("Failed to create signal from buffer (%d)\n", err);
        return;
    }

    ei_impulse_result_t result = { 0 };
    err = run_classifier(&signal, &result, debug_nn);
    if (err != EI_IMPULSE_OK) {
        ei_printf("ERR: Failed to run classifier (%d)\n", err);
        return;
```

```
    }

    size_t max_index = 0;
    float max_value = result.classification[0].value;
    for (size_t ix = 1; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        if (result.classification[ix].value > max_value) {
            max_value = result.classification[ix].value;
            max_index = ix;
        }
    }

    String classe_detectada =
String(result.classification[max_index].label);

    Serial.print("Classe detectada: ");
    Serial.print(classe_detectada);
    Serial.print(" | Anomaly score: ");
    Serial.println(result.anomaly, 3);

    setLedColor(classe_detectada, result.anomaly);

    ei_printf("Predictions (DSP: %d ms., Classification: %d ms.,
Anomaly: %d ms.):\n",
        result.timing.dsp, result.timing.classification,
result.timing.anomaly);
    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        ei_printf("    %s: %.5f\n", result.classification[ix].label,
result.classification[ix].value);
    }

#if EI_CLASSIFIER_HAS_ANOMALY == 1
    ei_printf("    anomaly score: %.3f\n", result.anomaly);
#endif
}
```

---

11. Resultados e Conclusões

É importante ressaltar que como o modelo foi criado utilizando só um modelo de

liquidificador, somente com a carga de 1 litro de água, caso sejam realizados testes em outras condições os resultados podem não ser satisfatórios, mas isso seria facilmente resolvido com um dataset maior com mais casos variados.
O modelo de classificação desenvolvido utilizando a plataforma Edge Impulse foi capaz de identificar com 82,73% de precisão no dataset de treino e 100% de precisão nos testes de bancada, os diferentes estados de funcionamento do liquidificador desligado, velocidade baixa com água, velocidade alta com água e anomalias comprovando a eficácia da abordagem adotada. Mesmo recorrendo a uma rede neural simples gerada pelo Edge Impulse, os resultados se mostraram excelentes, reforçando a viabilidade do uso de TinyML em dispositivos embarcados para aplicações como monitoramento de equipamentos, manutenção preditiva e automação residencial.

Além do classificador principal, foi implementado um detector de anomalias baseado em k-means, com o objetivo de identificar padrões atípicos de funcionamento do liquidificador.